FOREWORD OF THE SCIENTIFIC EDITOR OF THE FIVE-VOLUME CYCLE OF MONOGRAPHS

This volume of a five-volume cycle of monographs for a specialist in the field of strength calculations stands somewhat apart. A researcher or a strength engineer usually does not think about algorithms, and even more so about the most complex mathematical apparatus, on the basis of which the algorithms for constructing (generating) grids for the numerical solution of strength problems are developed. This is probably due to the fact that if a universal or niche CAE (Computer Aided Engineering - engineering analysis system) is used for calculations, then it contains an industrial grid generator. If the solution of a particular complex research problem requires the development of an independent program, then the existing grid generator is used, since this is almost always enough to solve a particular problem. At the same time, it should be understood that such a program is usually workable only in the hands of a researcher. The need for

a deeper dive into the problem arises when developing a fully functional CAE, for example, for strength engineering analysis, capable of solving complex problems with finite deformations and their redistribution. Such problems describe mechanical processes, with finite deformations under loading, in which the boundaries and boundary conditions change. Such processes include, for example: the formation of stress concentrators during loading; changing the material properties of a body part; forced, from the point of view of the mechanics of a deformable solid body, an increase (change) in the mass of the body. Such problems are solved on the basis of the theory of multiple superimposition of large deformations (vols. I–III). Academician L.I. Sedov,

noting more than a third of a century ago its practical significance in the future. It should be noted that in our time, with the development of additive technologies, the creation of new materials, with an increased need to more accurately evaluate supercritical loading scenarios and product life, the industrial implementation of such problems has become a practical necessity.

---

1) *Levin V. A.* Multiple imposition of large deformations in elastic and viscoelastic bodies / Preface. acad. L. I. Sedova. — M.: Nauka, Fizmat lit, 1999. — 223 p.

In this case, a very high functionality is required from the mesh generator. This is not only the basic functionality - the ability to build structured and some types of unstructured grids, but also, in particular, automatic adaptation of the grid to changes.
increasing geometry (topology) of the computational domain and to stress/strain fields in the process of solution, including anisotropic meshes that make it possible to take into account the induced anisotropy in the vicinity of stress concentrators. This is important, for example, in dynamic problems in which it is necessary to

ensure high accuracy of the solution at the wave front. To implement this functionality, it is necessary, in particular, to solve the problem of mesh rebuilding near the curvilinear boundaries of the body; control of anisotropic adaptation of meshes through their local modifications; restoration of body geometry using a deformed mesh; automatic mesh rebuilding to improve the convergence of iterative algorithms for solving essentially nonlinear problems under large deformations and distortions of the original body geometry; automatic rebuilding (improving properties) of a finite element mesh based on the original mesh containing strongly elongated

elements and/or elements with "excessively sharp" corners. This volume is devoted to the description of the solution of such problems. Most of the research in this area, due to the specialization that has arisen in modern science, unfortunately ends at the stage of the research code and rarely gets industrial implementation. The authors of the volume were able to bring them to an alienable software product that has proven its usefulness in research tasks and is able to complement the functionality of industrial mesh generators 1). Academician G.I. Marchuk

It should be noted that in the four years that have elapsed from the coordination of the structure of the five volume cycle to its release, a serious trend has emerged in the technologies for solving the problems of strength engineering analysis using the methods of isogeometric analysis [29, 33, 44, 58, 84]. The essence of the approach is to use the linear space as the basis functions, within which the solution of the original boundary value problem in partial derivatives is sought, for example, by the method of weighted Galerkin residuals, inhomogeneous rational B-splines (NURBS), simultaneously specifying the geometry of the considered body (CAD -model; in practice, this is a joint use of CAD and CAE). In this case, there is no need to introduce an additional basis consisting of shape functions given on the finite element discretization of the original model

1) This integration is done in CAE FIDESYS (www.cae-fidesys.com).

and, as a consequence, the construction, often nontrivial, of unstructured grids is not

required. It should be noted that at present, effective methods have been developed for the geometric description of the boundaries of three-dimensional bodies (curved linear surfaces) using NURBS, and, as a result, isogeometric analysis is successfully used, for example, in solving problems for shell structures 1). H This approach in industrial problems of strength engineering analysis is generally impossible because of the unsolved problem of constructing a basis from three dimensional NURBS for regions of arbitrary shape [32]. When (if) this problem is overcome, it is likely that a significant part of the trivial standard strength calculations will be carried out using isogeometric analysis. Complex problems associated with the analysis of defect behavior, the development of prefracture zones, changes in boundaries and boundary conditions, and body mass will still require the use of mesh generators. And, probably, a symbiosis of these or similar approaches will be implemented in industrial packages. We can assume the emergence of a computational block with the conditional name

"sampling generator", combining these approaches 2). A methodologically similar process is observed when the method of spectral ele one hundred of the finite element method (see vols. I, II) 3).

The method

external approximations 4). This method is a kind of logical generalization of the finite element method, expanding and supplementing its main ideas, including the weak formulation and the search for a solution in Sobolev subspaces. Without going into details, we note that, within the framework of the method of external approximations, the concept of a generalized finite element is introduced, which in problems of the strength of assemblies and complex composite structures is actually their component part — in fact, each assembly detail can be described by one gene

1) *Cottrell JA, Hughes TJR, Bazilevs Y.* Isogeometric Analysis: Toward Integration of CAD and FEA. — NY: Wiley & Sons, 2009.

2) Therefore, the scientific editor of this series of monographs proposed to prepare the second part of vol. IV (as a separate volume) on the use of isogeometric analysis in the numerical solution of problems in the mechanics of a deformable solid body.

3) The described combination of numerical discretization methods is implemented in the industrial strength engineering analysis package CAE FIDESYS (www.cae-

fidesys.com). 4) *Apanovich VN* Method of external finite element approximations. - Minsk: Highest. school, 1991. - 170 p.; *Aubin J.P.* Approximation of elliptic boundary value problems. — NY: Wiley-Interscience, 1972. See

finite element of arbitrary geometric shape. Another difference from the classical finite element with isoparametric discretization of the geometry and fields of unknown functions inside the element is the possibility of specifying an arbitrary set of basis functions that are in no way related to the geometry of the generalized element; the only requirement for choosing a basis is completeness in the corresponding Sobolev subspace. An unsolved problem from the point of view of mass industrial use is the insufficiently accurate description of complex boundaries. Note that, from a practical point of view, for the user, the method of external approximations is similar to isogeometric analysis, allowing one to perform calculations directly on the initial geometry of the body without any modifications and additional actions to prepare the model for calculation. Considering all of the above, the editor of the cycle expects the creation within the next decade in industrial packages of strength

engineering analysis of some combinations of “sampling generators” and solvers that combine existing and emerging methods for solving strength problems. In conclusion, I would like to note the deep immersion of the authors of the volume into the problems of strength at finite strains and their redistribution, their practical

participation in their solution 1), careful study of the examples given in the book and proposed by the editor of the cycle 2). This volume will be of interest to both users and developers of packages for strength engineering analysis, as well as useful to senior students and graduate students of specialized specialties.

Moscow, Moscow State University M. V. Lomonosov June 11, 2016

Professor V. A. *Levin*

1) In particular, automatic mesh adaptation in problems of elastic wave propagation under finite deformations over a damaged material (vol. I) was performed by A.A. Danilov.

2) That is why the author of the preface considered it right for himself not to be part of the team of authors, but to confine himself to selecting examples and discussing the structure of the volume.

AUTHOR'S PREFACE

We started working on the problem of constructing quasioptimal grids for the numerical solution of partial differential equations more than 15 years ago. Initially, it was obvious to us that such grids should be unstructured and may contain triangles or tetrahedra that are strongly elongated in some direction. Analysis of the properties of numerical solutions on such grids required the development of new algorithms and special software. The purpose of the book is a detailed discussion of various practical tools that are in demand in engineering applications and necessary for efficient automated construction of unstructured adaptive computational grids, both regular and anisotropic. One of our practical observations is that the fast and reliable construction of computational grids requires a combination of three different methods. The advancing

front method constructs a mesh fairly quickly in 95–99% of the computational domain. The Delaunay triangulation method requires more computational costs for constructing each element of the grid, but allows one to complete the grid in the remaining problematic subdomains. Finally, the shape of the mesh elements in the problematic subdomains is improved using the anisotropic meshing method, which has the highest computational complexity. The algorithms that form the basis of our reliable meshing technology are presented in Chapters 3 and 5. Optimization of computational costs in dynamic problems assumes that the initial mesh changes over time depending on the dynamics of the solution. Chapter 4 presents methods for constructing adaptive dynamic meshes through their multilevel hierarchical refinement and coarsening. In the remaining chapters and appendix, we discuss various methods for controlling the local size and shape of mesh elements using scalar and tensor metrics, as well as various numerical error estimates. The book is intended for developers of engineering analysis systems, engineers and mathematicians whose activities are related to the construction of computational grids. Our colleagues have provided us with considerable assistance in writing the book. A. Aguzal made a decisive contribution to the theoretical analysis of the properties of numerical solutions on anisotropic grids. The mathematical methods

developed jointly with him formed the basis of a number of algorithms implemented in our open source software,

Ani2D and Ani3D packages. We are grateful to A.V. Vershinin and A.V. Plenkinu, who made a significant contribution to the development of a method for improving a given surface mesh. We thank V.A. Garanzhu, R.V. Garimello, V.M. Goloviznina, N. Deby, V.G. Dyadechko, P.L. George, I.V. Kapyrina, T.K. Kozubskaya, V.D. Liseikin, D.R. Madison, Yu.M. Nechepurenko, K.D. Nikitina, I.B. Petrova, A.V. Plenkina, I.A. Sazonova, P.E. Farrell, F. Hesh and A.Yu. Chernyshenko for discussions of algorithms for constructing computational grids and comments that allowed us to improve the book. For the invaluable contribution to testing our software, we are grateful to the students of the Moscow Institute of Physics and Technology and Moscow State University, as well as to many users around the world who sent their comments.

The work on the book took several years and a lot of personal time. We are deeply grateful to our families for their support during this time.

Moscow, Russia and Los Alamos, USA
December 3, 2014

Yu.V. Vasilevsky
A.A. Danilov
K.N. Lipnikov
V.N. Chugunov

Chapter 1

INTRODUCTION

This book is the fourth volume of the cycle "Nonlinear Computational Mechanics of Strength". The main goal of this cycle is to present various mathematical models, numerical methods, and algorithms that form the basis of engineering analysis (SIA) systems and make it possible to describe the behavior of a deformable solid body under finite deformations, when boundaries and boundary conditions change during its loading, discretely or continuously. conditions. An important application of SIA is strength problems, the solution of which is associated with numerical simulation of the development of a defect in a solid, taking into account the occurrence of prefracture zones and phase transitions at the defect tip under the action of mechanical stresses. Numerical solution of this kind of problems requires rich tools for constructing computational grids. The fourth volume of the series is devoted to the description of various meshing technologies.

The main purpose of the book is to present a number of modern algorithms for constructing computational grids for engineering applications, in particular, those related to strength problems. We confine ourselves to considering only unstructured grids, which can be constructed in almost any computational domain, given both in the plane and in space. Technologies for constructing structured grids (i.e., grids with an ijk-structure of connectivity between nodes) based on constructing a mapping of some canonical domain into a given computational domain are less automated than technologies for constructing unstructured grids. A number of monographs and scientific collections published both in our country and abroad are devoted to methods and algorithms for constructing structured grids [1, 10, 14–16, 22, 56, 59, 61, 68, 69]; we refer the interested reader to these sources.

This book is a summary of many years of experience of the authors in the practical construction of unstructured computational grids. The main purpose of the book is to discuss the various tools used in the construction of such grids. We see the area of application of the presented tools primarily in the development and implementation of new grid generators in engineering applications. The target audience of the book is SIA developers, engineers and mathematicians whose activities are related to the construction of computational grids, i.e. those who directly program or use grid generators.

The book by no means claims to be an exhaustive review and theoretical analysis of all methods for constructing unstructured grids and the data structures used for this. The reader can study these issues using monographs [8, 11, 18, 36, 50, 52, 54] and the works of specialized domestic (NUMGRID-2004, 2006, 2008, 2010, 2012) and international (International Meshing RoundTable, 1992–2014) conferences (see, for example, [76, 77]). The yet unpublished monograph by V.A. Garangi "Numerical geometry and construction of computational grids", containing rich theoretical material on the subject under consideration. The framework of our book is formed by a set of algorithms implemented by us in the ani2D and ani3D software packages, which are freely available on the Internet: www.sf.net/projects/ani2d; www.sf.net/projects/ani3d. The content is a discussion of the presented algorithms, including practical results of their use.

This book has a number of features that distinguish it from already published monographs. First, it represents *all the stages* of the technological chain of construction of the computational grid, starting from the specification of the area and ending with the procedures for improving and adapting the grid. In the monographs known to the authors [18, 36, 50, 54], one or several stages of the technological chain are considered. Second, we consider *automated technologies* for reliable mesh generation that ensure the trouble free operation of algorithms. This quality is extremely important when building unstructured 3D meshes in complex areas. Reliability of 3D meshing is achieved by a combination of several methods, since no individual method alone can provide an acceptable quality mesh. Thirdly, we give *a detailed description* of the algorithms used, which makes it possible to understand the features of the technical implementation of the algorithms in our programs. An algorithmic view of the methods for constructing computational grids is also characteristic of the book [18]. The features of the book listed above and software implementations of the described algorithms make it a convenient guide not only for users of our software, but also for developers of new generation libraries.

grids.

Most of the geometric models considered in the book are typical examples in strength problems: a two-dimensional or three-dimensional body with holes or cracks. When a significant mechanical load is applied to such bodies, it is very important to calculate internal stresses and predict prefracture zones, which is apparently impossible without the use of adaptive meshes. For example, in the case of crack propagation, it is necessary to dynamically rebuild the mesh, ensuring its thickening to the current position of the crack edge. For this, it is necessary: firstly, to construct an initial mesh that adaptively thickens to the edge of the crack at the initial moment

time; secondly, to rebuild the grid at each time step, condensing it to a new calculated position of the crack edge and coarsening it in other zones. In the case of calculation of prefracture zones in bodies with cavities, it is necessary to adapt the mesh to the vertices of defects and track the boundaries of prefracture zones and, possibly, rebuild the mesh depending on the implementation of phase transitions. We do not present the results of calculations for problems of deformation of a solid body, but we present meshes that can be used to solve such problems, as well as possible technologies for their construction.

Let us briefly list the main advantages of the technologies developed by us for constructing and adapting unstructured computational grids. The concepts we use in the introduction will be rigorously defined in chapter 2 of the book.

The book proposes a set of algorithms and software tools for the reliable construction of 2D and 3D grids with regular cells in regions of complex shape, which can be specified in various ways. For example, a three-dimensional region can be defined as a set of union, subtraction, and intersection operations of geometric primitives such as an ellipsoid, box, cone, etc. The region boundary can also be defined as a set of parameterized surface pieces whose boundaries are also parameterized. The most common approach to defining an area is to use a design automation system (CAD) 1), which is an integral part of many CIAs. The interaction of the grid generator with CAD, which stores the internal parametric representation of the region boundary, is provided by a special software interface. The constructed unstructured meshes can be refined using local mesh refinement methods. In this case, triangular or tetrahedral cells selected by

the user are divided into two subcells, as well as some neighboring cells, which guarantees the conformity of the resulting mesh. When applied several times, this procedure provides multi-level local grinding, during which the quality of new cells can only slightly deteriorate. Moreover, refined meshes allow multi-level local coarsening according to rules set by the user, thereby providing the construction of dynamically adaptable meshes that track the moving features of the mesh solution.

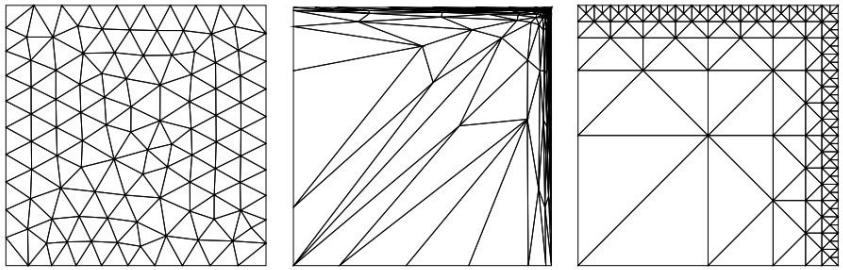
In addition to hierarchical refinement and coarsening, unstructured meshes can be completely rebuilt through a sequence of local mesh modifications. This approach has the greatest potential compared to hierarchical refinement.

1) Wanglophone literature CAD (Computer Aided Design).

meshes, because it allows you to build anisotropic meshes. Anisotropic adaptation reduces or increases the size of cells in preferred directions, which is reasonable for solutions with anisotropic features. The set of local operations is very limited. In the 2D case, it suffices to implement just five basic operations: inserting a node onto an edge, replacing an edge, shifting a node, deleting an edge, and deleting a node. In the three-dimensional case, the number of basic operations is seven: inserting a node onto an edge, direct and reverse replacement of a face by an edge, replacing an edge by a polygonal face, shifting a node, deleting an edge, and deleting a node. The same basic operations are used to fix and detangle meshes. Such meshes arise, for example, when solving deformation problems using dynamic meshes that move

along with the deformable body.

Examples of an unadapted unstructured grid, an anisotropic adapted grid, and a hierarchical locally refined grid are shown in Figs. 1.1. The choice between locally refined and completely unstructured meshes depends on the problem. In a number of problems of elasticity and hydrodynamics, especially in problems with strong singularities, an anisotropic unstructured mesh makes it possible to achieve an acceptable accuracy of the mesh solution on a much smaller number of cells than a regular mesh. On the other hand, the construction of an anisotropic grid requires more computational effort.



A *V*

b Fig. 1.1. Examples of computational grids: *(a)* unadapted unstructured, *(b)* anisotropically adapted, *(c)* hierarchical locally refined

chennaya

Automated algorithms for adaptive rebuilding of computational grids require control over the properties of these grids. You can control the local hierarchical rebuilding of the grid using the indicators of a posteriori error, examples of which we give. To construct unstructured grids, it is necessary to determine continuous functions that characterize the local properties of the cells, for example, their desired size at each point of the region. A posteriori error estimates underlie the construction of such functions. Functions,

that control the isotropic refinement or coarsening of the mesh are scalar, and the functions that control the anisotropic adaptation are tensor, since they must specify both the cell size and the direction of its elongation. The correct setting of the control function is an important condition for the efficiency of the computational grid. We propose new methods for constructing these functions and show their efficiency both theoretically [24, 26] and practically.

Finally, another advantage of our approach is that

that the technologies for constructing computational grids listed above are implemented in freely distributed software packages, and therefore can be tested by any user. Let's move on to a brief description of the contents of the book. Chapter 2 is

introductory for the non-specialized reader. In § 2.1 we give the main definitions of unstructured grids considered in the book, and also introduce common notation. The general name for the grids generated by our technological chain is conformal simplicial grids. *A simplex* is a triangular cell for 2D meshes or a tetrahedral cell for 3D meshes. In § 2.2 we discuss the main properties of simplicial grids that can be used in applications. Section 2.3 discusses the data structures and fast algorithms underlying the technology for generating unstructured meshes.

Chapter 3 is devoted to the construction of unstructured simplicial grids in arbitrary domains with a piecewise smooth boundary. Section 3.1 discusses several ways *of specifying* the boundary of a region. The set of methods covers the widest range of areas encountered in applications. In §§ 3.2 and 3.3, two main methods for constructing two-dimensional meshes are discussed - the Delaunay triangulation method and the advancing front method. In § 3.4 the advancing front method is transferred to the construction of surface triangulations, which is the initial stage in the generation of a tetrahedral mesh. If the boundary of the three-dimensional area is already defined by surface triangulation, then this stage of the technological chain can be omitted. If the quality of a given surface mesh is poor, it may be necessary to improve it, as discussed in § 3.5. Sections 3.6 and 3.7 are devoted to questions of automatic generation of tetrahedral meshes with a given trace on the boundary. Chapter 4 covers the technology of multilevel hierarchical local refinement and coarsening of simplicial grids, which preserves the regularity of simplex cells. In this chapter, as in all the following, it is assumed that the initial mesh in the region has been built. Section 4.1 gives the basic principles of hierarchical mesh refinement. Sections 4.2 and 4.3

describe in detail the algorithms of the bisection method for triangular and tetrahedral meshes. In § 4.4, the technology of multilevel coarsening of a locally refined mesh is considered,

which opens the way to the construction of a sequence of grids with dynamic adaptation, described in § 4.5.

Chapter 5 presents the most general technology for adaptive rebuilding of simplicial meshes by means of their local modifications. The flexibility and generality of the technology is due to the fact that, firstly, the properties of the adaptive rebuilt mesh are practically independent of the properties of the initial mesh and, secondly, this approach allows the construction of anisotropic adaptive meshes. Section 5.1 presents the basic principles of organizing mesh rebuilding algorithms based on its local modifications. Sections 5.2 and 5.3 describe in detail all types of local operations on 2D and 3D meshes, respectively. The sequence of local modifications of a tetrahedral mesh can be computationally expensive, so in § 5.4 we describe a parallel version of the technology under consideration. In addition to adaptability, a set of local mesh modifications also provides the ability to correct and unravel meshes, which is discussed in § 5.5. Although small, Chapter 6 provides the reader with important practical information about controlling the properties of unstructured meshes. Sections 6.1 and 6.2 successively consider methods for controlling the properties of regular and anisotropic simplicial grids. These sections describe the construction

of the necessary scalar and tensor functions, respectively. Adaptation of computational grids is an important link in the technological chain for constructing grids with the optimal arrangement of nodes and edge connections between them, which minimize the error of the grid solution. Some important practical issues of grid adaptation are considered in the appendix. Section A.1 considers the rebuilding of grids near curvilinear boundaries. Inaccurate approximation of curvilinear boundaries

can severely limit the accuracy of grid solutions. Section A.2 describes a posteriori error estimation methods for controlling the properties of hierarchical locally refined adaptive grids. Section A.3 presents various approaches to controlling the anisotropic adaptation of meshes by means of their local modifications.

Chapter 2

BASIC CONCEPTS

This chapter will introduce the basic concepts needed to construct triangular and tetrahedral meshes. We also recall the main results of graph theory, which will be used in subsequent chapters. Finally, we describe the basic data structures needed to implement the locally refined and unstructured meshing algorithms presented in the book.

§ 2.1. Triangular and tetrahedral meshes

A plane *triangle* is the simplest polygon having three vertices v1, v2, v3 and three edges e12, e23, e13. *The edge* eij is a segment, where 1 i<j 3. If all three vertices of a and vj lie on the same straight line, then it is triangle connecting a pair of vertices vi called *degenerate.* Let a Cartesian coordinate system (x, y) be given on the plane, in which any point

vi can be represented by a vector vi with two components xi and yi. A directed edge eij connecting two vertices vi and vj can be represented by the vector eij = vj ÿ vi. (Here and below we will use regular font for points or edges as such, and bold font for points or edges as vectors.) The triangle ÿ is the convex hull of the three vertices v1,

v2 and v3, i.e. the set of points

$$\ddot{y}=x:x=\sum_{i=1}^{3}\ddot{y}_i v_i,\ \ddot{y}_i \geq 0,\ \sum_{i=1}^{3}\ddot{y}_i = 1 \quad .$$

The algebraic area of a triangle ÿ is the quantity

$$S\ddot{y} = \frac{1}{2}\det\{e12,e13\} \ddot{y} \frac{1}{2}((x2 \ddot{y} x1)(y3 \ddot{y} y1) \ddot{y} (x3 \ddot{y} x1)(y2 \ddot{y} y1)), \quad (2.1.1)$$

where

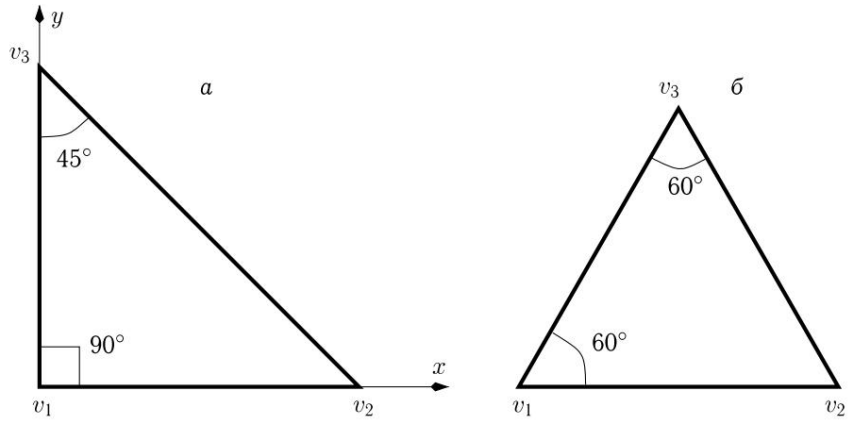
det{e12,e13} denotes the determinant of a matrix with two columns e12 and e13. Due to the properties of the determinant, the algebraic area changes sign when any two vertices in the triangle ÿ are interchanged. *The area* of a triangle is the absolute value of the algebraic plane
spare.

A canonical triangle is a triangle in which two edges of unit length coincide with the unit vectors of the chosen Cartesian

coordinate systems. *An equilateral* triangle is a triangle with equal edge lengths. The diameter of a triangle ÿ, like any polygon, is the value

$$\text{diam}(ÿ) = \max |x ÿ y|. \quad \textbf{xÿÿ,yÿÿ}$$

Due to the convexity of the triangle, its diameter coincides with the length of the largest edge. Examples of canonical and equilateral triangles are shown in fig. 2.1.



Rice. 2.1. Canonical *(a)* and equilateral *(b)* triangles

The area of a canonical triangle is 1/2, and the area of an equilateral triangle of diameter 1 is

$$Sÿ = ÿ\overline{34}\,. \qquad (2.1.2)$$

A triangle is said *to be regular* if the ratio of its area and the area of an equilateral triangle of the same diameter is on the order of 1. There are several equivalent measures for the regularity of a triangle. Let |e| denotes the length of the edge e of the triangle ÿ, and Pÿ is its perimeter: Pÿ = |e12| + |e13| + |e23|. In this book we consider the following measure of the regularity, or quality of the shape, of a triangle:

$$Qÿ = 12ÿ3\,|Sÿ|\frac{}{P2ÿ}\cdot \qquad (2.1.3)$$

This formula is based on the following property: an equilateral triangle has the largest area among all triangles with a fixed perimeter. It follows from this definition that Qÿ is a positive value not exceeding 1. Moreover, Qÿ = 1 only for an equilateral triangle.

The relationship between the quality of the shape of a triangle and the parameters of its geometric shape is illustrated in Table. 2.1 for two examples. The data in the first two lines correspond to the extension of an equilateral triangle into an acute isosceles triangle. The triangle stretch factor s is 1 for an equilateral triangle and grows linearly with the triangle's largest height. Note that the quality of the shape of a prolate triangle decreases approximately in inverse proportion to the stretch factor s.

The data in the last two lines correspond to an isosceles triangle with a varying angle with equal sides. An angle less than 90ÿ corresponds to the extension of the triangle, and an angle greater than 90ÿ corresponds to its flattening. We note in particular that the quality of the shape of a right triangle is quite high and equals approximately 0.9.

T a b l e 2.1

Changes in the quality of the shape when stretching (second line) and flattening (fourth line) an isosceles triangle

s	1	2	4	8	16	32	64	128	256
Qÿ	1.0	0.849	0.563	0.325	0.174	0.090	0.046	0.023	0.012
ÿ	10ÿ	30ÿ	50ÿ	70ÿ	90ÿ	110ÿ	130ÿ	150ÿ	170ÿ
Qÿ	0.382	0.820	0.983	0.986	0.892	0.738	0.548	0.336	0.113

In practice, we recommend building grids with elements for which Qÿ 0.7. For the examples above, this means that s ÿ (1, 2,9) and ÿ ÿ (23ÿ,114ÿ). A triangular mesh, or *triangulation* ÿh of a polygonal (polygonal) domain ÿ, is a finite set of triangles ÿi, i = 1, ... , N, such that:

— the region ÿ is completely covered by these triangles; — the boundary of the domain ÿÿ is completely covered by triangle edges; The interiors of the triangles do not intersect. This mesh definition is quite general and includes both conformal and nonconformal triangulations (a triangulation is said to be *conformal* if the intersection of any two triangles is either their common vertex, their common edge, or the empty set). A triangulation is called *uniform* if it consists of equilateral triangles, and *quasi-uniform* with step h if all its triangles are close (in measure Qÿ) to an equilateral triangle of diameter h. Note that uniform grids do not exist in three dimensions.

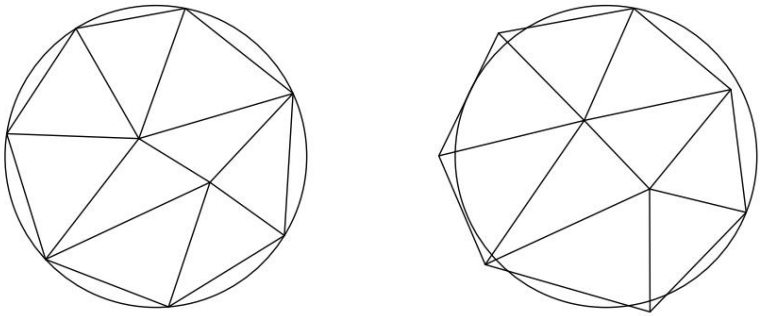
A triangulation is *regular* if it consists of regular triangles. For triangular regular grids, the following inequalities hold:

$$c \quad \frac{ÿÿ}{\text{diam}(ÿ)} < 1, \qquad (2.1.4)$$

where ÿ ÿ is the radius of the inscribed circle, and the constant c does not depend on the grid triangle. In contrast to a quasi-uniform grid, a regular grid can contain triangles that differ greatly in diameter, although the diameters of any two adjacent triangles do not differ much, i.e., are close (in measure Q ÿ). For example, for the regular grid shown in Fig. 1.1c, we have c = (ÿ 2 ÿ 1)/2. The measure of the regularity of the grid ÿh, or its quality Q(ÿh), can be the smallest of the qualities of the form Q ÿi of the triangles ÿi composing the grid: Q(ÿh) = min

$$1i N \quad Q ÿi .$$

The concept of triangulation of a domain ÿ with a piecewise smooth curvilinear boundary is a natural generalization of the concept of triangulation of a polygonal domain. The main difference is that the boundary of the region is approached by the mesh edges, and both incomplete coverage of the near-boundary part of the region and the mesh going beyond the boundaries of the region are allowed, see Fig. 2.2.



а *б*

Rice. 2.2. The boundary nodes of triangulation lie on the curvilinear boundary *(a)* and near this boundary *(b)*

In many practical applications, the boundary nodes of a triangulation lie on a curvilinear boundary (see Figure 2.2). This ensures that the boundary of the domain is approximated by triangulation edges with the second order of accuracy with respect to the length of the edge, i.e., the distance from the boundary edge of the grid e to the curvilinear boundary is a value of the order of |e| 2. The concept of triangulation can be generalized for a

region defined on a two-dimensional surface in three-dimensional space, as well as

for a closed two-dimensional surface which is the boundary of a three-dimensional body. In this case, the grid nodes must have three coordinates (x, y, z), the edge of the triangle is a curved segment lying on a two-dimensional surface, and the absolute value of the algebraic area (2.1.1) is only an approximation of the integral area of the triangle. Tetrahedral meshes in space are defined by analogy with triangular meshes in the plane. *A tetrahedron* in

three dimensions is the convex hull of four vertices v1, v2, v3, and v4. If all the vertices of a tetrahedron lie in the same plane, then it is called *degenerate. An edge* of a tetrahedron is a segment eij connecting and a *face* of a tetrahedron is a triangle fijk a pair of vertices vi and vj , with vertices vi, vj and vk.

Let us define some Cartesian coordinate system (x, y, z) in which any point vi is represented by a vector vi with three components xi, yi, zi. A directed edge connecting two vertices vi and vj can be represented by the vector eij = vj ÿ vi. The orientation of the face with vertices vi, vj, and vk is given by the normal vector nijk = eij × eik. The normal vector can be directed both inside and outside the tetrahedron. The definition of a tetrahedron can be formalized as follows. A tetrahedron with vertices v1, v2, v3 and v4 is a set

points

$$ÿ=x:x= \quad \sum_{i=1}^{4} ÿivi, ÿi \; 0, \quad \sum_{i=1}^{4} i = 1 \quad .$$

Algebraic, or oriented, *volume of* the tetrahedron ÿ on

is called the quantity

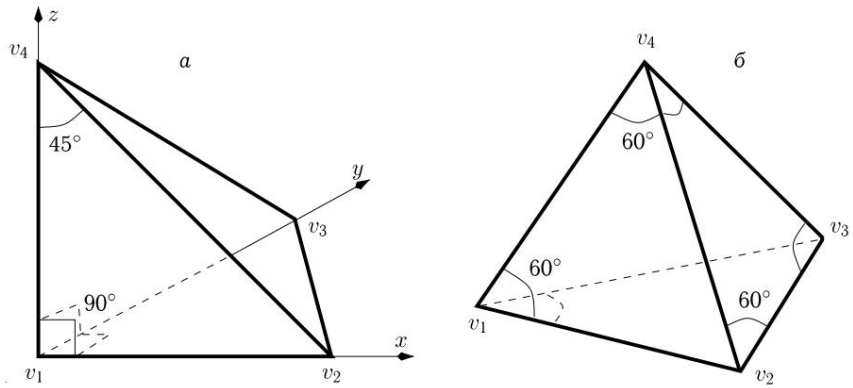
$$Vÿ = 6 \; \frac{1}{} \; \det\{e12,e13,e14\}, \qquad (2.1.5)$$

where det{e12,e13,e14} denotes the determinant of a matrix with three columns e12, e13 and e14. The algebraic volume changes sign when any two vertices in the tetrahedron ÿ are interchanged. *The volume* of a tetrahedron is the absolute value of the algebraic volume.

A canonical tetrahedron is a tetrahedron in which three edges coincide with the unit vectors of the chosen Cartesian coordinate system. *An equilateral, or regular,* tetrahedron is a tetrahedron with equal edge lengths. The diameter of a tetrahedron ÿ is the length of its largest edge. Examples of canonical and regular tetrahedra are shown in fig. 2.3. The volume of a canonical tetrahedron is 1/6, and the volume of a regular tetrahedron of diameter 1 is

$$Vÿ = \frac{1}{6 ÿ 2} = . \qquad (2.1.6)$$

A tetrahedron is called *regular* if the ratio of its volume and the volume of a regular tetrahedron of the same diameter is



Rice. 2.3. Canonical and regular tetrahedra

order 1. There are several equivalent measures of tetrahedron regularity. Let Pÿ be the sum of the lengths of the edges of the tetrahedron ÿ, i.e., Pÿ = |e12| + + |e13| + |e14| + |e23| + |e24| + |e34|. In this book we consider the following measure of the regularity, or shape quality, of a tetrahedron:

$$Q\ddot{y} = 64\ \ddot{y}\ 2\ \frac{|V\ddot{y}|}{P\ddot{y}^3}. \qquad (2.1.7)$$

It follows from the definition of Q ÿ that the quality of the shape of a tetrahedron is a positive value not exceeding 1. It can be shown that Q ÿ = 1 only for a regular tetrahedron. In what follows (chaps. 2–5) sometimes instead of “quality of form” we will just write "quality". The relationship between the value of the quality of a tetrahedron and its shape in two specific examples is shown in Table. 2.2. The data in the first two lines correspond to the extrusion of a regular tetrahedron in the direction of one of its heights. The stretch factor s is 1 for a regular tetrahedron. A comparison with a similar table for triangles shows a much faster decrease in quality with increasing s. For larger values of s, doubling s results in

T a b l e 2.2

Changes in quality upon modification of regular (second row) and canonical (fifth row) tetrahedra

s	1	2	4	8	16	32	64
Q ÿ	1.0	0.785	0.398	0.148	0.046	0.013	0.003
ÿ	61ÿ	70ÿ	90ÿ	110ÿ	130ÿ	150ÿ	170ÿ
ÿ	84.2ÿ	71ÿ	54.7ÿ	41.4ÿ	29.1ÿ	17.4ÿ	5.78ÿ
Q ÿ	0.468	0.999	0.804	0.575	0.386	0.223	0.073

to a decrease in the quality of the tetrahedron by a factor of 3–4. Therefore, in practice, we recommend constructing meshes with quality Q(ÿh) 0.2.

The data in the last three rows of the table correspond to the tetrahedron, which is obtained from the canonical tetrahedron by changing the three equal dihedral angles at the vertex v1 (see Fig. 2.3). These angles can vary within a limited range ÿ ÿ (60ÿ, 180ÿ), where the extreme values correspond to degenerate tetrahedra. In the penultimate row of the table, we present the values of ÿ for three other equal dihedral angles. The quality of a rectangular tetrahedron is about 0.8. Similarly, Table. 2.1, as ÿ tends to 180ÿ, the quality of the tetrahedron also decreases. However, the limited dihedral angles in a tetrahedron do not mean its high quality: in the limiting case, when ÿ ÿ 60ÿ and ÿ ÿ 90ÿ, the quality Q ÿ tends to zero. This shows a significant difference between the quality of a tetrahedron and the quality of a triangle, where the limited angles means its high quality. The tetrahedron needs both dihedral and planar angles to be limited.

A polyhedral (polyhedral) region is a region with a piecewise smooth boundary represented by a finite set of flat faces. A tetrahedral mesh or *tetrahedralization* ÿh of a polyhedral region ÿ is a finite set of tetrahedra ÿi, i = 1, ... , N, such that: — the region ÿ is completely covered by these tetrahedra; — the boundary of the region is completely covered by the faces of tetrahedra; — the interiors of the tetrahedra do not intersect. This mesh definition includes both conformal and non-conformal meshes. A tetrahedralization is said *to be conformal* if the intersection of any two tetrahedra is either their common vertex, or their common edge, or their common face, or the empty set. A tetrahedralization is said to *be quasi-uniform* with step h if all its tetrahedra are close in measure Q ÿ to a regular tetrahedron of diameter h. Note that space cannot be covered by regular tetrahedra, i.e., there are no uniform tetrahedral grids.

A tetrahedralization is *regular* if it consists of regular tetrahedra. The inequalities (2.1.4) hold for regular grids. The measure of regularity Q(ÿh) of the tetrahedralization ÿh can be the least of the qualities Q ÿi of the tetrahedra ÿi that make up the mesh:

$$Q(\ddot{y}h) = \min_{1i\ N} Q\ \ddot{y}i. \qquad (2.1.8)$$

Regular triangulations and tetrahedrizations are widely used in engineering calculations, since discretizations on regular grids have a number of advantages, such as discretization stability, acceptable conditionality of generated matrices, and the existence of efficient search algorithms. The concept of tetrahedralization of a domain with a curvilinear piecewise smooth boundary is a natural generalization of the concept of tetrahedralization

polyhedral region, in which the boundary of the region is approached by the boundary faces of the mesh. In this case, both incomplete coverage of the border part of the region and the grid going beyond the boundaries of the region are allowed. In practice, as a rule, the boundary nodes of tetrahedralization lie on the boundary of the region, which makes it possible to approximate this boundary with the second order of accuracy with respect to the face diameter.

At the end of the section, we present the notation used in what follows for various grid elements. The term "simplex" is used for the common name for the triangle and tetrahedron. The symbol ÿ is used to denote the computational domain, and ÿh is used for its triangulation or tetrahedrization. The superelement ÿ(v) denotes the set of simplices with a common vertex v. The symbol ÿ(ÿ) denotes the superelement, or the set of simplices intersecting with the simplex ÿ. Thus,

$$ÿ(ÿ) = \bigcup_{vi ÿ ÿ} ÿ(vi).$$

Similarly, the superelement ÿ(e) denotes the set of simplices with a common edge e. If v ÿ e ÿ ÿ, then

$$ÿ(v) ÿ ÿ(e) ÿ ÿ(ÿ).$$

Let d denote the dimension of the space: d = 2 for the plane and d = 3 for the space. A grid element (simplex) with vertex d + 1 is denoted as ÿ(vk1 , ..., vkd+1), or i = 1, ... for short, mi vki, denote a ÿk1...kd+1 . Similarly, we will use f(vk1 , vk2 , vk3) or fk1 k2 k3 to triangular face in space, and e(vk1 , vk2) or ek1 k2 to denote an edge. For each type of grid element, depending on the context, we will use the notation given in Table. 2.3:

T a b l e 2.3

Symbol table for grid elements

Grid element as an object		as a vector
vertex	v vi	v vi
edge —	e eij	e eij ÿ vj ÿ vi
facet ⌐	fijk	nf nijk ÿ eij × eik

§ 2.2. Properties of grids and elements of graph theory

The relationship between graphs and grids makes it possible to use some results of graph theory in studying the properties of grids and performing operations on grids. Approximate ratios between the numbers of nodes, edges and mesh elements given

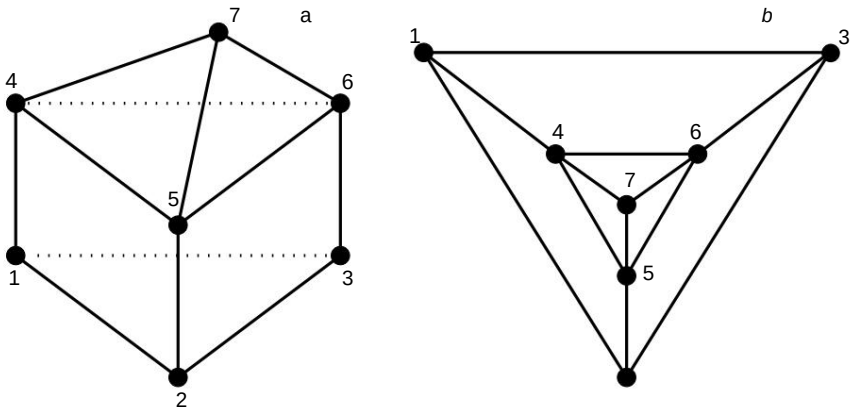
below are useful for estimating the computer memory required to store the grid. The most famous result of graph theory

applicable to the analysis of grids is Euler's formula relating the numbers of vertices, edges, and faces of any convex polyhedron. Let Nv, Ne, Nf denote, respectively, the numbers of vertices, edges, and faces of some polyhedron. Note that the faces of a polyhedron can be arbitrary polygons. Then

$$Nf = Ne ÿ Nv + 2. \quad (2.2.1)$$

For example, for a tetrahedron Nv = Nf = 4, Ne = 6, and for a cube Nv = 8, Nf = 6, Ne = 12. Based on the Euler formula

(2.2.1), we derive the relation between the number of nodes, edges and triangles in an arbitrary two-dimensional conformal triangulation. To do this, we note that any conformal triangulation of some simply connected domain can be represented as a surface of some polyhedron spread out on a plane, in which one of the faces has been removed (see Fig. 2.4). The polyhedron shown is the union of a ÿ4567 tetrahedron and a triangular prism. With such spreading, it is assumed that the removed face corresponds to the addition of the considered triangulation to a plane or an infinite cell.



2 Fig. 2.4. Polyhedron *(a)* and its flattened surface, lower face removed *(b)*

Therefore, for a conformal triangulation of a simply connected domain, ratio

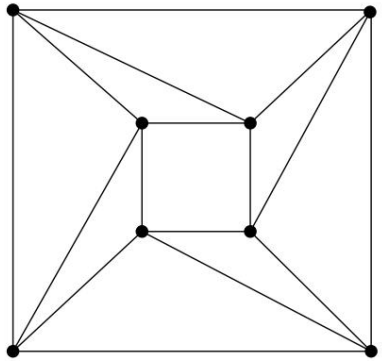
$$Nf = Ne ÿ Nv + 1.$$

In the case of a nonsimply connected domain, the conformal triangulation will contain Nh nontriangulated polygons, each of which corresponds to a region cut out from the domain. Such regions

are given, for example, by obstacles in flow problems or voids in a sample in problems of solid body mechanics. Each of these polygons contributes to the number of faces in the Euler formula, but must not contain triangles of the considered triangulation. Thus, the modification of the Euler formula (2.2.1) for the case of conformal triangulation of a nonsimply connected domain has the form

$$Nf + Nh = Ne ÿ Nv + 1. As \quad (2.2.2)$$

an illustration, we will use the triangular grid shown in fig. 2.5.



Rice. 2.5. Triangular mesh in square hole area: Nv = 8, Ne = 16, Nf = 8, Nh = 1, Nei = 8 and Neb = 8

Formula (2.2.2) can be rewritten using the number of boundary edges instead of the number of all triangulation edges. The boundary edge eb belongs to only one triangle, in contrast to the interior edge ei, which belongs to two triangles. Therefore, between the numbers of boundary edges Neb, internal edges Nei, and the number of triangles Nf in a conformal triangulation, there is the following relationship:

$$2Nei + Neb = 3Nf . \quad (2.2.3)$$

Using formulas (2.2.2) and (2.2.3), we obtain the relation for the number Nf :

$$Nf = 2Nv ÿ Neb + 2Nh ÿ 2. \quad (2.2.4)$$

Since Neb 3 and Nv 3, from (2.2.4) we obtain an unimprovable an estimate of the number of triangles for Nh = 0:

$$Nf 2Nv ÿ 5. \quad (2.2.5)$$

Since the number of triangles in a grid with Nh > 0 is less than in a grid with triangulated voids, estimate (2.2.5) is also valid for the general case Nh 0. When constructing dynamic grids, we can

allocate the memory necessary for operation, using only one parameter - the maximum allowable number of triangulation nodes.

From formulas (2.2.2) and (2.2.5) we obtain an unimprovable estimate for the number of edges

$$Ne 3Nv ÿ 6. \quad (2.2.6)$$

In the vast majority of practical cases, the number of nodes in a conformal triangulation significantly exceeds the number of boundary edges Neb and the number of cut components Nh; therefore, in practice, there are approximate relations

$$Nf ÿ 2Nv and Ne ÿ 3Nv,$$

expressing the asymptotic dependence between the numbers of triangles, edges, and nodes of the conformal triangulation. Note that

formula (2.2.4) was derived under the assumption that ÿh is a triangulation of a connected domain defined on the plane. For triangulations of closed surfaces defined in space, such as the boundaries of three-dimensional bodies, this formula may not be true. In this case, it is necessary to use the topological characteristics of the surface based on the classification of closed two dimensional surfaces as a finite sum of spheres and tori. For example, if on a connected closed surface it is possible to draw Ng nonintersecting closed curves that do not cut the surface into disconnected components, then such a surface is homeomorphic to a sphere with Ng handles, and its triangulation is true [3, 17]

$$Nf = 2Nv + 4 Ng ÿ 4.$$

Let us now consider tetrahedral meshes and use the Euler formula (2.2.1) to derive the relation between the number of tetrahedra Nt, edges Ne, and nodes Nv in a tetrahedralization of a connected domain satisfying the following constraint: for each node v, the set of tetrahedra containing v (incident to v) forms a polyhedron homeomorphic to a sphere. Recall that each such polyhedron is called a superelement and is denoted by ÿ(v), or ÿ for short. The introduced restriction is not burdensome in practice. Let us define the characteristic function ÿb for the boundary of the grid domain. Let ÿb(v) = 1 if the node v lies on the

boundary of ÿh, and ÿb(v) = 0 for an internal node of the domain. Let us introduce the number of faces Nf (ÿ), edges

Ne(ÿ), nodes Nv(ÿ) on the surface of the polyhedron ÿ, the number of tetrahedra Nt(ÿ) in ÿ, the number of edges Ne(v) incident to v, and the number of faces Nb(ÿ) in ÿ lying on the boundary ÿh and containing v. Then:

$$Nv(ÿ) = Ne(v) + ÿb(v), Nf (ÿ) = Nt(ÿ) + Nb(ÿ),$$

and formula (2.2.1) is written for ÿ as follows:

$$Nt(ÿ) + Nb(ÿ) + Ne(v) + ÿb(v) = Ne(ÿ) + 2. \quad (2.2.7)$$

Using formula (2.2.3), on the surface ÿ we have the identity

$$2Ne(ÿ) = 3(Nt(ÿ) + Nb(ÿ)),$$

which, taking into account (2.2.7), gives

$$Ne(v) = \frac{1}{2} Nt(ÿ) + \frac{1}{2} Nb(ÿ) ÿ ÿb(v) + 2.$$

Summing over all grid nodes ÿh and taking into account that

$$\sum_v Ne(v) = 2Ne, \qquad \sum_v Nt(ÿ) = 4\,Nt,$$

we get

$$Ne = Nt + 4 \quad \frac{1}{\sum_v} Nb(ÿ(v)) ÿ \quad \frac{1}{2} \sum_v ÿb(v) + Nv.$$

Let Nfb and Nvb be the number of boundary faces and the number of boundary nodes the tetrahedralization ÿh, respectively. Because the $\sum_v Nb(ÿ(v)) = 3Nfb$ of

And $\sum_v ÿb(v) = Nvb$, That

$$Ne = Nt + Nv + \frac{3}{4} Nfb - \frac{1}{2} Nvb.$$

Since Nv 4, Nfb Nvb , the set of edges and the maximum possible Ne is bounded by the pessimistic estimate

$$Ne \quad \frac{1}{2} Nv(Nv ÿ 1), \tag{2.2.8}$$

That

$$Nt \quad \frac{1}{2} Nv(Nv ÿ 3) ÿ \quad \frac{3}{4} Nfb + \frac{1}{2} Nvb \quad \frac{1}{2} Nv(Nv ÿ 3). \tag{2.2.9}$$

However, by imposing additional restrictions, one can obtain a more optimistic estimate. If for each node v its degree (the number of edges incident or converging in it) is bounded by some value ÿ, i.e., Ne(v) ÿ, then

$$Ne \quad \frac{ÿ}{2} Nv$$

And

$$Nt \quad \frac{ÿ}{2} ÿ 1Nv.$$

Thus, in tetrahedralizations with a uniformly limited degree of sites, the number of tetrahedra is estimated from above in terms of a linear function of the number of sites. We note that there are tetrahedralizations in which Nt N2, however, the methods considered in this book generate grids with a uniformly bounded degree of knots. Here ab means that the ratio a/b is about 1. In unstructured tetrahedra with a large number of tetrahedra and nodes,

where the vast majority of vertices are internal, the approximate is true ratio

$$Nt ÿ 5.5\,Nv.$$

Euler's formula can be generalized to the multidimensional case as follows. Consider a d-dimensional convex polytope (polytope) consisting of a set of k-dimensional faces, k = 0, ... , d ÿ 1. By a k-dimensional face we mean here a convex open set lying in some k-dimensional hyperplane. The vertex of the polyhedron is a 0-dimensional face, the edge is a 1-dimensional face, and the polyhedron itself is a d-dimensional face. For convenience, we introduce a (ÿ1)-dimensional face. Let nk denote the number of k-dimensional faces of a polyhedron, where each polyhedron corresponds to exactly one (ÿ1)-dimensional face, and the d-dimensional face coincides with the polytope itself. Thus, by definition, n ÿ1 = 1 and nd = 1. The generalized Euler formula for d-dimensional polytopes is as follows:

$$\sum_{k=ÿ1}^{d} (ÿ1)^k n_k = 0.$$

The relationship between d-dimensional and 3-dimensional definitions is:

$$n2 = Nf\ ,\ n1 = Ne,\ n0 = Nv.$$

Graph theory is applicable to the analysis of the properties of triangular grids. One of the consequences of the theory of planar graphs is the statement about the possibility of unraveling any triangulation, which is a "tangled" network of triangles bordering in pairs through their integer edges, into a conformal triangulation. Let us explain this statement and also explain the term "disentanglement of triangulation". *A graph* G = (V , E) is a set of vertices V and a set of edges E, not necessarily straight lines, where each edge from

E connects a pair of vertices from V . *A planar* graph is an undirected graph that can be laid on a plane without self-intersections. A planar graph drawn on a plane is called *a planar* graph. A face of a planar graph is a bounded singly connected region whose boundary is given by the edges of the graph in the form of a simple cycle v1v2, v2v3, ..., vkv1, where all vertices enter twice. If all faces of a planar graph are triangles (i.e., k = 3), then such a graph is called *triangulated.* Graph triangulation is a procedure for complementing the set of edges E to obtain a triangulated graph. According to Rado's theorem [80], any polygon with a boundary that has the form of a simple closed broken line with a finite number of links,

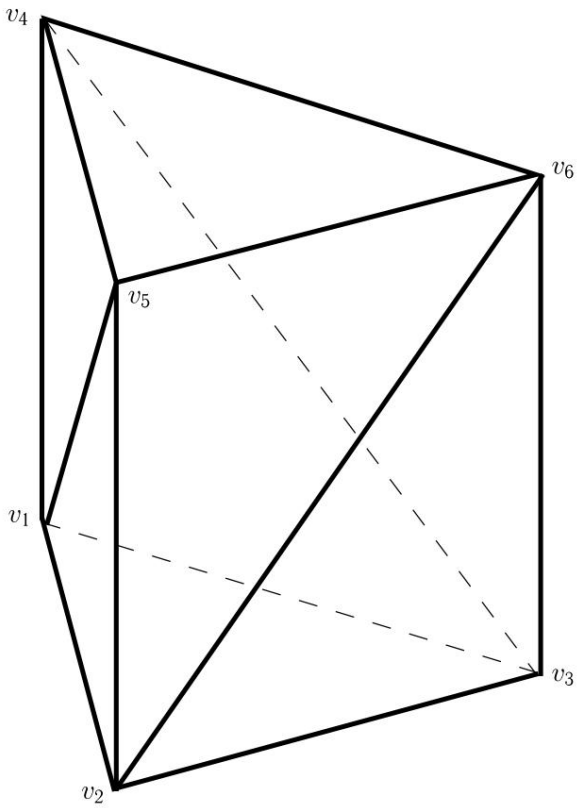
allows conformal triangulation without adding new points. In terms of graph theory, this statement looks as follows. **Theorem 2.2.1.** *Any planar graph with straight edges*

can be triangulated. This assertion, like Rado's theorem, proves the existence

definition of conformal triangulation for polygonal regions.

Building or rebuilding a triangulation can lead to a non-conformal mesh. Under the simple unraveling of the mesh, we mean the movement of its nodes, preserving the edge connections between the nodes, in order to obtain a conformal triangulation. Farey's theorem [47] guarantees the existence of such a displacement of nodes for a grid defined by a planar triangulated graph.

Theorem 2.2.2. *Any planar graph has a planar representation in which all edges are represented as line segments.*



Rice. 2.6. Schonhardt prism with triangular base v1 v2 v3

There are criteria for the planarity of graphs [63]; therefore, for triangular grids that satisfy such criteria, there is a procedure for simply unraveling the grid into a conformal triangulation.

Unfortunately, the above analysis cannot be extended to the case of conformal tetrahedrizations. For example, in the area with a given surface grid shown in Fig. 2.6, it is impossible to construct a conformal tetrahedralization without adding additional internal nodes. Vgl. In Section 6, we consider more complex mesh disentanglement algorithms that change its topology and are therefore applicable to disentangle both triangulations and tetrahedrals.

Another important concept in the theory of planar graphs is the concept of *a dual* graph. For a planar graph G, the dual graph is Gdual with the following properties: — each vertex of Gdual is

associated with a face of G; — each edge Gdual is

associated with the edge G; — if the edge G separates

two faces fi and fj , then the corresponding edge of the dual graph Gdual connects the vertices Gdual associated with fi and fj . The

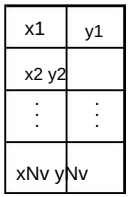
concept of a dual graph is used both in some methods for constructing triangular grids and Voronoi grids, and in methods for discretizing differential equations [45, 74].

§ 2.3. Data structures and fast algorithms

To store the grid and perform any operations on it, you need to use various data structures. There are a large number of different data structures for storing a grid, oriented to different purposes [18, 50, 54]. The enumeration and comparative analysis of these data structures is beyond the scope of this book, and we will limit ourselves to only those structures that are often used when working with triangulations and are easily transferred to tetrahedralizations. The minimal and simplest representation of a triangular grid is a list of coordinates of numbered nodes (xi, yi) and a

list of numbered triangles, each of which is given by the indices of three nodes (i1, i2, i3). In this case, the coordinates are stored in a real two-dimensional array V rt(2, Nv), and the triangles are stored in an integer two-dimensional array Tri(3, Nf).

Finite element methods can also use two-dimensional integer arrays of edges Edge(2, Ne) and boundary edges Bnd(2, Neb) that store node indices. These structures are auxiliary as they can be created automatically from the Tri array.



Rice. 2.7. Structured list V rt(2, Nv) to store node coordinates

All the listed arrays are *structured lists* or tables with a given row length (see Figure 2.7). Structured lists also include information about the numbers of triangles adjacent to each triangle. The construction of such a list is discussed in Chap. 4. The algorithm for iterating over the elements of a structured list is very simple. It consists of two nested loops over the rows and columns of the corresponding two-dimensional array,

as shown in Algorithm 1 for a Tri structured list.

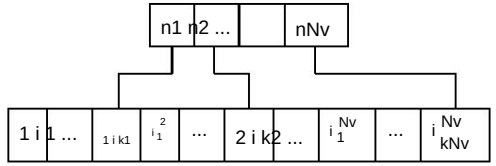
Algorithm 1. Enumeration of elements of the structured list Tri 1: **for** i = 1, Nf **do for** j = 1, 3

```
do return Tri(j, i) –
2:          number of j-th vertex
                of triangle i 3: 4: end for 5: end for
```

In what follows, we will use the notation XY for an array, where a grid object x ÿ X contains references to the nearest grid objects y ÿ Y such that x ÿ y = ÿ. For example, TV is another notation for the two-dimensional array Tri(3, Nf), where T is a triangle and V is a grid point.

A number of applications need data that is reasonable to store as a flat list. For example, data on the numbers of triangles containing a grid node (converging at a common vertex) are characterized by the variability in the number of triangles assigned to each node. Of course, such data can also be stored as a two-dimensional array VT(Kmax, Nv), where Kmax is the maximum possible number of triangles converging at a common vertex. However, such a strategy can significantly increase the amount of computer RAM reserved for data, since Kmax can significantly exceed the grid average value, and most of the VT array will be filled with zeros.

The flat list is represented by two linear integer arrays. In the case under consideration, the first linear array iV T(3Nf) contains only nonzero values from the array VT, sorted by rows. The second linear array nV T(Nv) contains the indices of the elements of the array iV T, which are the last in each row of the array VT (see Fig. 2.8). This format is similar to the sparse string format commonly used for storing sparse matrices.



Rice. 2.8. An unstructured list for holding triangles that share a common vertex. Array nV T(Nv) on top, array iV T(3Nf) on bottom

Algorithm 2. Iterating over the elements of an unstructured list iV T, nV T 1: Let n1 = 1 2: **for** i = 1, Nv **do**

```
n2 = nV T(i) 3: for n
= n1, n2 do k = n ÿ n1 +
          1 return iV T(n)
4:
is the number of the kth triangle
5:              containing node i
6:

7: end for n1 = n2
8:        + 1 9: end for
```

The number of non-zero elements in each row of the VT array is calculated as follows:

$$nV\,T(i) \text{ ÿ } nV\,T(i \text{ ÿ } 1),\ i > 1;\ nV\,T(1),\ i = 1.$$

The algorithm for iterating over the elements of an unstructured list consists of two nested loops, presented in Algorithm 2.

Note that the number of elements in the iV T array is equal to the number of elements in the TV array, and this is not accidental. The unstructured list iV T, nV T is the inverse of the structured list TV and must therefore contain the same number of elements. Similarly, the structured list TV can be viewed as the inverse of the unstructured list iV T, nV T.

There are simple algorithms with linear complexity for creating reverse lists. For example, the construction of the list iV T, nV T is described in Algorithm 3.

Algorithm 3. Building an unstructured list iV T, nV T 1: Initialize nV T(1) = 2: **for** n = 1,

```
Nf do for i = 1,3 do i1 = TV (i,n) nV      ···   = nV T(Nv) = 0
T(i1) = nV T(i1) + 1 5:
3:        6: end for 7: end
4:              for 8: for n = 2,
                Nv do 9: nV T(n) = nV T(n) +
nV T(n ÿ 1) 10:
end for 11:
store nLast = nV T(Nv)
12: for n = 1, Nf do for i = 1,3 do i1 = TV (i,n)
and k = nV
T(i1) iV T(k) = n nV T(i1) = k ÿ 1 end for
17: 18: end for 19: for
13:       n = 1, Nv ÿ 1 do
14:             20: nV T(n) = nV T(n + 1) 21: end for
15:             22: restore nV
16:             T(Nv) = nLast
```

The process of building or rebuilding a grid is associated with its constant modifications, and, consequently, with constant changes in data structures. One of the important problems that arise in the implementation of grid algorithms is the problem of fast search within a dynamic data structure. Let us first consider the problem of maintaining *an ordered list of* elements of some one-dimensional dynamic

array of real numbers Q(N). Suppose this array is sorted in ascending order of elements:

$$Q(i) \; Q(i + 1). \qquad (2.3.1)$$

Recall the classical Algorithm 4 for searching for an interval containing the value x in an ordered array of numbers Q(N). This algorithm is based on the bisection (or bisection, or dichotomy) method. The arithmetic complexity of the bisection method is proportional to log2 N.

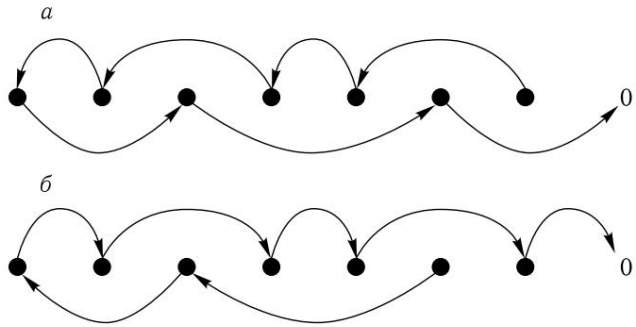
Algorithm 4. Finding an interval in an ordered list using the bi sections

```
1: l = 1, r = N 2:
while r ÿ l = 1 do l + r 2
3:        i =  ——
4:        if x Q(i) then r = i else l = i
          5: 6: end if
7: end while
8: if Q(l) x Q(r)
then found=.TRUE. 9: else found=.FALSE. 10: end
if
```

In problems where the size of the array Q and the values of its elements are constantly changing, maintaining its physical order is inefficient, so assumption (2.3.1) is no longer valid. For example, inserting a new element in the second position requires a shift of (N ÿ 1) elements. To specify the order, a linked list is used, based on a two-dimensional array Lst(2, N) such that

$$Q(Lst(1, i)) \; Q(i) \; Q(Lst(2, i)), \qquad 1<i<N,$$

where Q(Lst(1, i)) and Q(Lst(2, i)) are the nearest (in value) to Q(i) elements of the real array Q. An array Q, for which the linked list Lst is known, will be called implicitly *ordered list.* The beginning Q(ib) and the end Q(ie) of an implicitly ordered list are defined by the equalities Lst(1, ib) = 0 and Lst(2, ie) = 0. The example shown in fig. 2.9 shows that the beginning and end of the list can be located in adjacent cells of the array Q.



Rice. 2.9. Implicitly ordered list: *a* - Lst(1, i) links, *b* - Lst(2, i) links

Changes in the array of numbers Q involve adding or removing an element and, accordingly, increasing or decreasing N by one, as well as changing the value of some element without changing N. These operations are easily implemented using the linked list Lst, if you know what position you need place a new element or move a modified element. For example, the procedure for removing element i from a linked list is presented in Algorithm 5.

Algorithm 5. Removing element i from the linked list Lst

```
1: p = Lst(1, i), n = Lst(2, i) 2: if
p = 0 then Lst(2, p) = n 3: end if
4: if n =
0 then Lst(1, n ) = p 5: end if

6: N = N ÿ 1
```

When an element of an implicitly ordered list is removed, the corresponding memory location in the array Q remains in place. Since the linked list Lst no longer refers to this memory location, it becomes a gap in the array Q. To avoid an uncontrolled increase in the number of gaps, you need to constantly fill them when new elements are added to the list Q, that is, you need to store the gap list in an additional linear array.

The procedure for adding an element i to a linked list is also simple and is presented in Algorithm 6. The procedure for transferring an element of a linked list from one position to another is obtained by successive application of Algorithms 5 and 6.

Algorithm 6. Adding element i after the p-th element of the linked list Lst

```
1: Lst(2, i) = Lst(2, p), n = Lst(2, i)
2: Lst(1, i) = p
3: Lst(2, p) = i 4:
if n = 0 then Lst(1, n) = i
5: end if
6:N=N+1
```

Quickly finding the position where you want to insert or move an element of an implicitly ordered list requires additional structure. Indeed, sequential enumeration of the elements of the array Q using Lst(2, i) pointers to the next element, until finding the desired position, requires O(N) assignment and comparison operations, which is an unacceptably expensive procedure. The bisection method is not applicable to the array Q because the array is not explicitly ordered. The bisection method is also not applicable to the linked list Lst, since the middle of the list is unknown.

Our fast search algorithm is based on the introduction of additional pointers to the elements of the Lst(2, N) array. These pointers are stored in a one-dimensional array P tr(M), where M is the number of pointers. The pointers P tr(i) split the array Q into blocks with lengths approximately equal to log2 N, so that

$$Q(P\ tr(i))\ Q(P\ tr(j)),\ 1\ i<j\ M.$$

For convenience, we assume that P tr(1) = ib and P tr(M) = ie. Thus, the number of pointers M is approximately equal to N/ log2 N. The array P tr specifies some order in the implicitly ordered list Q. The array of elements Q(P tr(i)) is an explicitly ordered subset of the array Q, so the bisection method can be applied to it. The list of pointers P tr specifies *a block ordering* for Q. An array Q for which P tr and Lst are known will be called *a block ordered* list. The fast search for an interval containing the value x in a block-ordered list of real numbers Q is performed in two steps, as shown in Algorithm 7. Step 1 of this algorithm is a bisection method for an array of pointers Ptr, and its arithmetic cost is O(log2 M) = O(log2N). After that, all elements of the i-th block are sequentially traversed until the desired interval is found. Therefore, the arithmetic cost of the second stage is also O(log2 N).

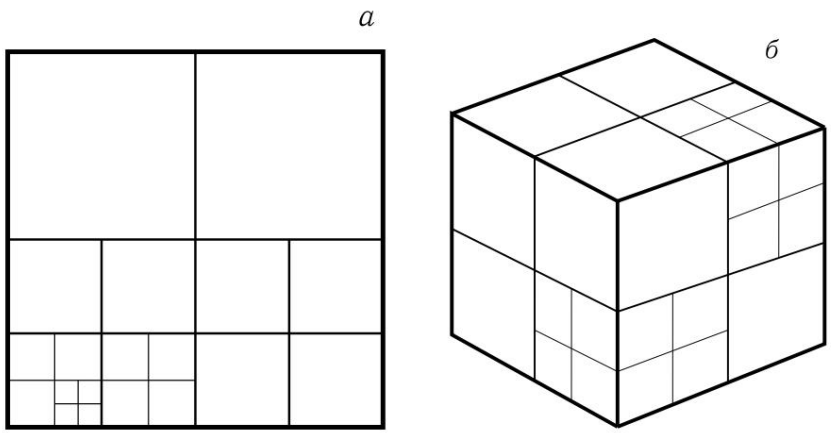
Algorithm 7. Finding an interval in a block-ordered list

```
1: Apply Algorithm 4 to find the i-th block containing x, i.e.
   Q(P tr(i)) x Q(P tr(i + 1))
2: i1 = P tr(i), i2 = Lst(2, i1) 3:
while x>Q(i1) and i2 = 0 do i1 =
4:      i2, i2 = Lst(2, i1) 5: end
while 6:
return i1, i2
```

Changes in the real array Q require periodic updating of the array of pointers P tr to maintain a fixed block length. These pointers are shifted to the preceding or following elements of the linked list Lst. Changing the i-th pointer may lead to the need to change the (i ÿ 1)-th and (i + 1)-th pointers. In the worst case, the total number of assignment operations at this step can reach M. Significant reduction in calls to the pointer update procedure is achieved by introducing a floating block length. As long as all block lengths are in the interval [0.75 log2 N, 1.25 log2 N], the P tr(i) pointer update procedure is not applied. Efficiency of the proposed method

maintaining a dynamic block-ordered list of real numbers is discussed in Chap. 5 when constructing adaptive grids.

Many operations with unstructured meshes are based on the rapid localization of various mesh elements (edges, faces, cells) in a given neighborhood. The most convenient auxiliary data structure for quickly searching for triangulation (or tetrahedralization) elements is a quaternary (or octal) tree structure. Grid analogues of quaternary and octal trees are shown in Figs. 2.10. Let's consider a quaternary tree in more detail.



Rice. 2.10. Grid analogues of quaternary *(a)* and octal *(b)* trees

A quaternary tree is structured like a normal binary tree; the difference is that each vertex does not have a pair of children, but four. Each vertex of such a tree is responsible for some part of the unit square [0, 1] × [0, 1]. In this case, the root of the tree is responsible for the entire square, and its descendants are each responsible for their quarter of the parent square. When passing from the parent to the children, the square cell of the parent is divided into 4 equal parts and distributed among the children. The level of a vertex is the number of edges in the path from this vertex to the root of the tree. Thus, the level of the root is 0, and the level of its direct descendants is 1. At each vertex, we store the list of triangulation elements assigned to it, which is assumed to belong to [0, 1] × [0, 1]. To select the vertex to which the element should be assigned, we apply Algorithm 8.

Algorithm 8. Choice of vertex of a quaternary tree

1: Find the midpoint x and diameter d of the grid element 2:
Find the smallest number n such that d 2ÿn 3: Find the
vertex of the quaternary tree at level n that is responsible for the square
containing x

Let us consider the advantages of such a structure when searching for an element of a triangular grid that intersects with some given circle B lying inside the unit square. For each triangle ÿi, we define the minimal circle Bi containing it. Then if ÿi intersects B, then Bi intersects B. Obviously, B and Bi intersect if and only if the distance between their centers ÿi is less than the sum of their radii R and Ri. Note that Ri 2ÿni where ni is the level of the vertex of the quaternary tree to which the triangle ÿi is assigned. The quaternary tree structure makes it , possible to very quickly find all triangles for which ÿiR + 2ÿni , since the cost of finding one element of the grid is proportional to log4 Nf actions. Only for such triangles does it make sense to check their intersection with the circle B. If the considered triangulation does not belong to the unit square, then it can be displayed by translating and scaling into the unit square before performing any actions. Algorithms for fast localization of grid elements based on quaternary and octal trees are effective for regular triangulations and tetrahedralizations. The efficiency of using these data structures for operations with adaptive grids is discussed

in Chap. 3.

Chapter 3

CONSTRUCTION OF UNSTRUCTURED GRID IN ARBITRARY AREAS

In this chapter, we describe two methods for constructing unstructured simplicial meshes: the advancing front method and the Delaunay triangulation method. When constructing tetrahedral meshes, each method has its own advantages and disadvantages, so the most effective strategy for constructing such meshes is to combine both methods. Particular attention will also be paid to methods for improving a given surface mesh, since its quality strongly affects the operation of methods for constructing a spatial mesh.

§ 3.1. Methods for defining a computational domain

For automatic construction of a simplicial (triangular or tetrahedral) mesh, information about the geometric model of the computational domain is required. Currently, several approaches are used to represent geometric models in computer memory. The most universal approach is used in design work automation systems (CAD). A geometric model within a CAD system can be represented in many ways. The model can be composed of simple shapes - geometric primitives, such as a circle, a square, a ball, a cube, a cylinder. The model can also be represented as a polygon or polyhedron, in which case only the boundary of the object is specified as a broken line on the plane or a set of triangles in space. The division of the boundary into segments or triangles is called *the discretization of the* domain boundary. Curved lines can be used instead of segments, and curved surfaces can be used instead of flat triangles. To store geometric information about the curvilinear components of an object, as a rule, parametrizations are used

ÿ : t ÿ (x, y, z), ÿ : (p, s) ÿ (x, y, z).

A widely used way to specify parameterization in CAD is to use non-uniform rational B-splines (NURBS - non-uniform rational B-spline). A combination of three approaches is possible: part of the model can be parameterized using

NURBS, the other part is given by the discretization of its boundary, and the third part is defined by primitives. Through the operations of union, intersection, addition and subtraction, the final geometric model is obtained.

The classical approach to the construction of simplicial grids is reduced to the initial construction of a discrete boundary of the domain, followed by the construction of a simplicial grid inside the domain based on the available boundary discretization. Thus, the problem of constructing simplicial grids is reduced to two separate problems: constructing a discrete boundary for a given geometric model and constructing a simplicial grid inside a domain with a given discrete boundary. The task of constructing boundary discretization can be solved in different ways, depending on the form in which the geometric model is specified. Generally speaking, at the second stage, information about the geometric model may not be available. When solving the second problem, an important requirement is the preservation of the trace of the simplicial grid on

the boundary of the domain. Failure to comply with this condition may lead to the impossibility of performing calculations on the resulting grid or require additional work on the processing of the resulting grid from the user. For example, if the constructed mesh is later merged with another mesh, then keeping a trace of the mesh on their common boundary will allow us to obtain a conformal common mesh. When specifying the boundary conditions in the model problem, keeping the trace of the grid will make it possible to avoid additional reinterpolation of the boundary data. In some cases, it may be acceptable for the user to slightly deviate from the specified trail at the boundary. The trace of the built grid on the boundary can be smaller than

the one specified initially. In this case, refinement is allowed only by adding new nodes on the boundary and splitting the elements into smaller ones. In this case, if necessary, the user can restore conformity at the junction with a neighboring mesh using an appropriate partition of the boundary elements of the second mesh.

From the point of view of automating the meshing process, a universal approach is to accurately preserve the trace of the mesh at the boundary of the region. In this case, the quality of the resulting simplicial mesh can be limited by the quality of the boundary discretization. For example, in the three-dimensional case, prolate triangles at the boundary of the domain significantly limit the quality of the adjacent tetrahedra. *To obtain high quality tetrahedral meshes, high quality surface triangulation is required.* In the two-dimensional case, the boundary of the region is a set of several lines, in the

general case, curves. The discretization of a curved line parametrized by a function ÿ : t ÿ (x, y), t ÿ [t0, t1] is constructed using standard methods. Sample step along the curve

either selected by the user or calculated automatically, for example, based on an estimate of the local curvature. In the 3D case, constructing a good

surface triangulation is a more difficult problem. In most cases, the entire surface of an object can be divided into several parts, each of which is either a part of a plane or a part of a parameterized surface ÿ: (p, s) ÿ (x, y, z). The boundaries of these parts are generally curved lines in three-dimensional space that have their own parametrization ÿ : t ÿ (x, y, z). At the first stage, discretization of curved boundaries (lines) in three-dimensional space is constructed. Next, for each piece of the surface, three angulations are constructed with a trace on the boundary coinciding with the constructed discretization. By preserving the trace of surface meshes of different pieces of the surface on a common curvilinear edge, the conformality of the common surface mesh is guaranteed. The method of constructing a surface triangulation using the advancing front method will be discussed in detail in § 3.4. In some cases, the geometric model of the area may already be defined as a surface triangulation. The quality of elements in surface triangulation can be very low, which is especially true for meshes obtained by exporting a CAD model. Further construction of a tetrahedral mesh with poor surface triangulation will inevitably lead to the appearance of bad

tetrahedra in the final mesh. In § 3.5 we will consider an approach that makes it possible to reconstruct surface meshes while preserving the geometric features of the region. The same approach can be used to obtain finer or coarser surface meshes.

based on what is available.

As an alternative way of specifying the geometric model, the indicator or characteristic function can be used

$$ÿ : (x, y, z) ÿ \{0, 1\}.$$

The point is inside the model if ÿ(x, y, z) = 1. There are methods [64] for obtaining the polyhedral boundary of the region specified using the indicator function. As a rule, to improve the quality of the obtained discrete boundary, its additional processing is necessary. The convex region can also be defined by a cloud of points in space. Often in

this case the user is interested in a simplicial mesh with nodes at given points. If the initial data of the model problem are known and given at specific points, then a grid with nodes at these points will allow one to avoid additional reinterpolation of the initial data. As a rule, in this case, the algorithms for constructing the Delaunay triangulation are used, which are considered in the next section.

§ 3.2. Construction of the Delaunay triangulation

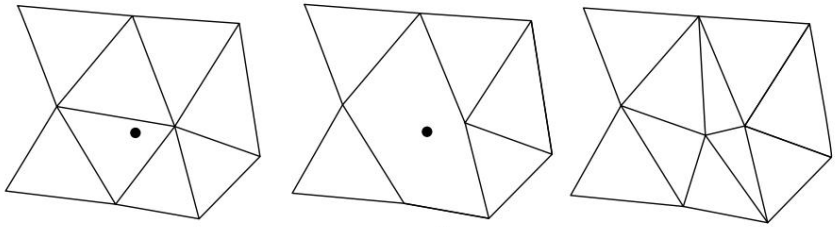
Consider the problem of constructing a triangulation on a given set of nodes V = {v1, ..., vn}. We will say that a triangulation satisfies *the Delaunay condition* if for any triangle inside the circle circumscribed around it there are no vertices of other triangles. *By a Delaunay triangulation* we mean a convex triangulation satisfying the Delaunay condition. A Delaunay triangulation is unique if no four nodes in V lie on the same circle.

There are several methods for constructing the Delaunay triangulation. In addition to direct methods for constructing a Delaunay triangulation, it can also be obtained from any other triangulation by successively rearranging adjacent pairs of triangles that do not satisfy the Delaunay condition into pairs of triangles that satisfy this condition.

In this section, a simple iterative algorithm for constructing a Delaunay triangulation will be briefly considered. The triangulation problem can be formulated as follows: let there be a partially constructed triangulation in the domain, to which a new node is added; it is required to complete the triangulation in the entire area.

In the mentioned algorithm, the position of the new node relative to the existing triangulation is first determined, and then, depending on the result, certain actions are performed. If the new node hits some edge, then it is split into two edges and adjacent triangles are split into two smaller ones. If a new node falls inside any triangle, then the latter is split into three smaller triangles. If the node does not fall inside the triangulation, then the boundary edges of the current triangulation are found, with which it can form new triangles. The proposed algorithm can be simplified (see Algorithm 9) by adding several auxiliary nodes in advance and constructing an initial Delaunay triangulation that completely covers the set of nodes V. In this case, each new node will lie inside the Delaunay triangulation. In all cases, the new triangles and their neighbors are tested for the Delaunay condition. If this condition is violated, a local reconstruction of the triangulation is performed, which consists in the following. When a new node is added, all triangles for which the Delaunay condition is violated are found and removed, and

new triangles are constructed inside the resulting polygon with the participation of the new node (see Fig. 3.1). The resulting triangulation will satisfy the Delaunay condition [18]. When constructing a Delaunay triangulation for a given two-dimensional region, it is necessary to determine the set of nodes V lying inside and on the boundary of this region; for this set, the Delaunay triangulation is constructed. In this case, the following simple algorithm can be used as an iterative algorithm for adding a new node inside the region: we choose the longest edge in the current triangulation



а *б* *в*

Rice. 3.1. Building a Delaunay triangulation by an iterative algorithm: *a* - adding a new node, *b* - deleting triangles, *c* - building new triangles

Algorithm 9. Building a Delaunay triangulation 1: Add

three auxiliary nodes that form the enclosing
triangle of set V
2: **for all** nodes v ÿ V **do** Add
a node v to the triangulation ÿh 3: 4: Find
the set of triangles ÿ for which the Delaunay condition is violated and remove
them from ÿh Construct new triangles
5: connecting v with the edges of the boundary ÿ ÿ 6: **end for** 7: Delete
auxiliary
nodes and
their corresponding triangle
Nicky

and put a new node in its middle. To control the uniformity and density of nodes in the area, more
complex methods.

The proposed algorithm for constructing a Delaunay triangulation constructs a triangulation for the convex hull of a set of nodes. If the desired region is nonconvex, then it is necessary either to use the methods of constructing Delaunay triangulations with restrictions [18] or artificially complete the triangulation by adding new nodes to the boundary of the region and then deleting triangles lying outside the desired region. When using the second approach, the trace of the mesh for convex regions is preserved, while for non-convex regions it can be refined.

The Delaunay condition naturally generalizes to tetrahedralization in three dimensions. *The Delaunay tetrahedrization* for a finite set of knots V = {v1, ..., vn} is such a conformal partition of the convex hull of the set V into tetrahedra such that for any tetrahedron inside the sphere circumscribed around it there are no other knots from V. A simple iterative algorithm for constructing a triangulation Delaunay can also be used to construct Delaunay tetrahedra in three dimensions [54]. In contrast to 2D triangular meshes, in the 3D case there are such polyhedral

domains for which it is impossible to construct a conformal tetrahedral mesh with a given trace on the boundary without adding new nodes. An example of such a region is given in § 2.2 in fig. 2.6. Several methods are known for preserving the boundary of a nonconvex domain in 3D space: local modifications of the mesh [34], mesh refinement [42], and construction of a constrained Delaunay tetrahedralization [81].

A distinctive feature of the Delaunay tetrahedrization is the presence of so-called *slivers* —strongly degenerate tetrahedra whose vertices lie almost in the same plane and almost on the same circle. For such tetrahedra, the Delaunay condition can be satisfied, but the tetrahedra themselves will be of very poor quality. To improve the quality of the mesh, local displacements of nodes are used. In practice, when implementing the algorithms for constructing the Delaunay triangulation, it is

necessary to pay special attention to data structures and algorithms for quickly searching for triangles or tetrahedra near a given point. A good overview of data structures and algorithms is given in the book [18] and in § 2.3. Special attention also deserves the accuracy of calculations when checking the fulfillment of the Delaunay condition. Some of these issues will be considered in the next paragraph.

§ 3.3. Construction of a triangulation by the advancing front method

In this section, we will propose an algorithm that constructs conformal triangular grids for two-dimensional domains given by discretizations of their boundaries. When constructing the grid, the algorithm preserves the given trace on the boundary. The proposed algorithm is applicable both to simple polygons and to arbitrary multicomponent multiply connected polygonal regions whose boundaries may not be one-dimensional manifolds. The question of constructing a discrete boundary for an arbitrary region with curvilinear boundaries will not be discussed here, but we will return to it in § 3.4.

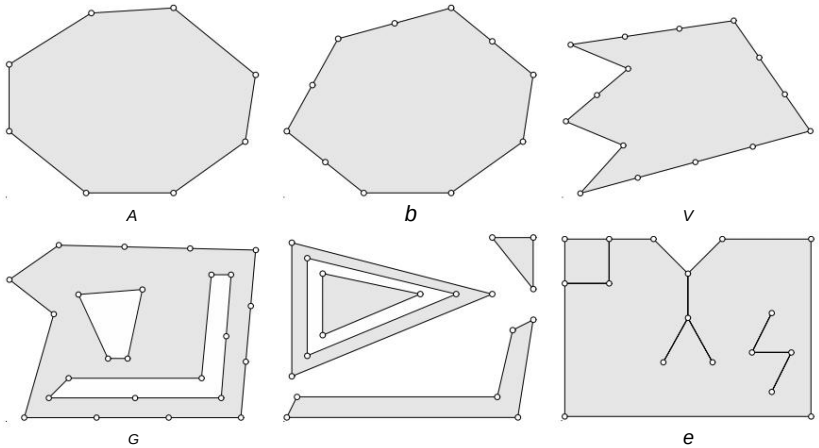
Below, we consider the classical advancing front algorithm and study the influence of computational errors on it, as well as the finiteness and running time of the algorithm. We need the following concepts and notation.

The positive half-plane with respect to the directed edge e12 = v2 ÿ v1 is the set of points v for which the algebraic area is positive [see. formula (3.3.1)]. Using the tensor product of vectors, it is convenient to write the algebraic area as a vector product:

$$S\ddot{y}(v1,v2,v) \ddot{y} \frac{1}{-} (v1 \ddot{y} v) \times (v2 \ddot{y} v) > 0. \quad 2$$

Similarly, *a negative half plane* is a set of points for which Sÿ(v1,v2,v) < 0. A *broken line* in the plane is a finite set of segments
connected in series by their ends. This does not exclude the location of two consecutive segments of the broken line on one straight line. If the first and last points of the polyline coincide, then such a polyline will be called *closed.* A broken line whose segments do not intersect each other will be called a broken line *without self-intersections.*

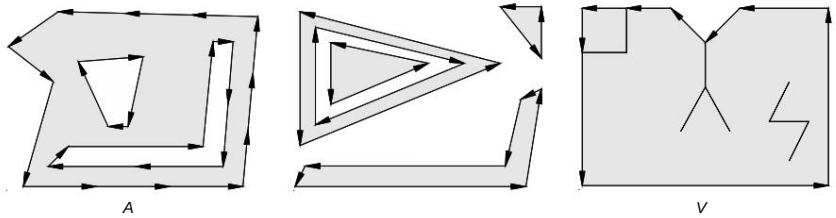
A polygon on a plane is a bounded open part of the plane whose boundary consists of one or more non-intersecting broken lines without self-intersections. *A simple polygon* is a polygon bounded by one closed broken line. The vertices of the polygon will be called the vertices of the broken lines, and its sides, the segments of the broken lines. On fig. 3.2 illustrates several types of polygons.



e Fig. 3.2. Types of polygons: *a* - strictly convex, *b* - non-strictly convex, *c* - non-convex, *d* - multiply connected, *e* - multicomponent, *f* - with internal cuts

The sides of a polygon can be divided into two types: external and internal. Let v1v2 be a side of P, v the midpoint of v1v2, and ÿÿ(v) an ÿ-neighborhood of v. Consider Pÿ = P ÿ ÿÿ(v). If for any ÿ > 0 the set Pÿ lies on both sides of the segment v1v2, then we will call v1v2 *the inner* side, or *cut.* If, for some ÿ > 0, a part of the polygon Pÿ lies only on one side of the segment v1v2, then v1v2 will be called the *outer* side of the polygon P. In the latter case, a direction can be introduced on the edge e12 so that the part of the polygon Pÿ is in the positive half-plane with respect to v1v2. We will call this direction the *counterclockwise* direction of the polygon.

arrows. On fig. 3.3 shows the directions of bypassing the outer sides of several polygons. Note that the totality of all external sides forms one or more closed broken lines.



b Fig. 3.3. Bypassing the polygon counterclockwise: *a* - multiply connected polygon, *b* - multicomponent polygon, *c* - polygon with internal cuts

A front on a plane is a set of directed edges without self-intersections. Each polygon P can be assigned a front F(P). To do this, on the outer sides of the polygon, we introduce a counterclockwise direction of traversal, and assign to each inner side v1v2 a pair of directed edges e12 and e21. The totality of all these directed edges forms the front F(P). We will call a front F *closed* if there exists a polygon P such that F = F(P).

3.3.1. Advance Front Algorithm

In this subsection, we consider the advanced front algorithm for constructing in a given polygonal domain P a conformal triangular mesh consistent with the boundary P.

By the current front Fk we mean some closed front, the boundary of the polygonal region Pk, in which the triangular grid is to be constructed. The symbol T k will denote the set of constructed triangles. We set the initial moment of time P0 = P, F0 = F(P0) and T 0 = ÿ. Next, at the kth step, we will build a new triangle ÿ123 ÿ Pk, add it to the triangular grid, and subtract it from the area where the grid has not yet been built:

$$T_{k+1} = T_k \cup \ddot{y}123, \quad P_{k+1} = P_k \setminus \ddot{y}123.$$

When Fk+1 = F(Pk+1) becomes the empty set, or equivalently, Pk+1 = ÿ, we can say that the set of triangles T = T k+1 completely covers P.

Recall that, according to our notation, ÿ123 denotes a triangle with vertices v1, v2, and v3, while e12 denotes an edge with vertices v1 and v2. The triangle ÿ123 will be constructed in such a way that one of its sides will be one of the segments of the current front Fk. Then after subtracting the new triangle from the polygonal area, the current front will change

insignificant: at least one segment from the front will be removed and no more than two new ones will be added.

Consider some directed edge e12 of the current front. For ÿ123 to lie inside Pk, it suffices to fulfill two conditions: 1) the algebraic area of ÿ123 is positive, i.e., S ÿ123 > 0; 2) the triangle ÿ123 does

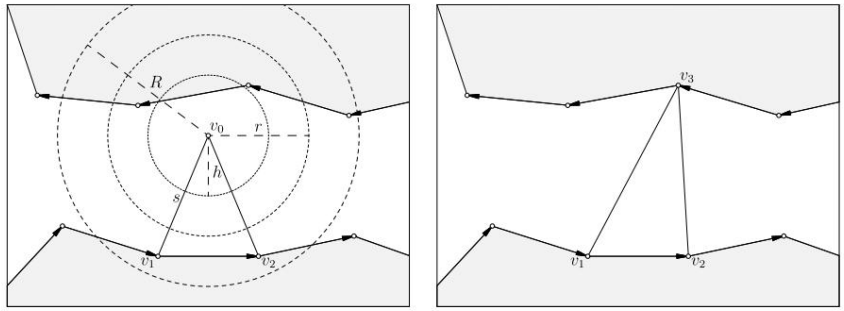
not intersect the front Fk. In this case, it is allowed that the vertex v3 is the vertex of the front, and the sides of the triangle coincide with the segments Fk. We will call the triangle ÿ123 that satisfies these conditions *suitable.* This choice of ÿ123 guarantees that after adding it to the grid, it will retain the

conformality property. In this case, the consistency of the final grid T with the initial domain P will be ensured. For definiteness, we will choose at each step the directed edge e12 with the smallest length and construct a new triangle ÿ123 on it. When constructing ÿ123, the choice of the third vertex v3 is not unique. We can be guided both by the desired size of the triangle in

this area, and by some other heuristic criteria. The vertex v3 will be chosen only from some finite set ÿ of candidate vertices. This set includes the point v0, equidistant from the points v1 and v2 at some distance s, and also add all the front vertices lying in the positive half-plane with respect to the directed edge e12. Since the set ÿ is finite, we can simply iterate over all triangles ÿ123, where v3 ÿ ÿ, until the next one is suitable (see Fig. 3.4b). Vp. 3.3.3 it will be proved that there is always at least one suitable triangle.

The complexity of such an algorithm is very large, and in practice it will not be effective. Therefore, we consider some convex open neighborhood ÿ containing the points v1, v2 and v0 (see Fig. 3.4). As the region ÿ, one can choose a circle of radius R>s centered at v0. We select from the current front only those segments that have at least one common point with ÿ. The resulting front may not be closed. We call it *a local front* and denote it by Fÿ. Any subdomain ÿ intersects with the current front if and only if it intersects with the local front. Let us construct the set ÿ = ÿ ÿ ÿ. Since ÿ is convex, any triangle ÿ123, where v3 ÿ ÿ ÿ, will lie in ÿ. To check the intersection of this triangle with the current front, it suffices to check its intersection with the local front. Local enumeration will be faster, but will no longer guarantee

the existence of a suitable triangle. Therefore, it is necessary to provide for the possibility to return to the full enumeration in the case when the local enumeration fails. To prove the finiteness of the number of operations during the operation of the algorithm, it is necessary to somehow estimate the maximum number of triangles,



A

b Fig. 3.4. Front advancement: *(a)* neighborhoods of the candidate vertex corresponding to different values of R; *b* - adding a suitable triangle to the grid

which we will build with it. To do this, you need to introduce some restrictions on the triangles themselves, on the edges and on the mesh vertices. From a practical point of view, it is most convenient to have a lower bound on the minimum pairwise distance between grid nodes. To maintain this constraint, we will exclude the point v0 from ÿ if it lies close to the front vertices or to the already constructed mesh. To do this, we will introduce two parameters: h is a restriction on the minimum distance to the current front; r is the limit on the distance to the tops of the front (see Fig. 3.4, a). If the current front intersects the h-neighbourhood of the point v0 or the vertices of the front lie in the r-neighborhood of the point v0, then we exclude the point v0 from the set of candidate vertices. Note that in Fig. 3.4 both conditions are violated, so the third vertex v3 is the front vertex. More details about working with these parameters will be

described in section 3.3.3. Taking into account all the previous remarks, we compose Algorithm 10. We will use the following additional notation. At the kth step of the algorithm, we denote the set of vertices of the front Fk by {pk }, and the set of nodes of the constructed grid T k by {vk }. For some R > 0 and a point v, we denote by ÿ R(v) the open convex R-dots v.

Next, we will study the stability of this algorithm to computational errors, its finiteness and complexity.

3.3.2. Impact of computational errors

When implementing Algorithm 10 on a computer, special attention should be paid to computational errors that arise when working with real numbers. Using specialized libraries for arbitrary precision calculations can solve the problem by increasing the program run time. In this section, we will analyze the impact of computational errors on the work

Algorithm 10. Advancing Front Method in 2D Space 1: Put T 0 = ÿ and F0 = F(P)

2: **for** k = 0, 1, ... **do** 3:
Choose e12 ÿ Fk with the minimum length Determine the
4: desired length s of the side of the triangle Construct the vertex v0: |e10| = |
5: e20| = s, S ÿ120 > 0 Choose some R>s, h > 0 and r > 0 Construct a
6: local front Fk R: Fk ÿ Fk ÿ ÿ R(v0) Determine the set of
7: candidate vertices ÿk = {pk } ÿ ÿ ÿ R(v0) If {pk } ÿ ÿ R(v0) = ÿ and Fk ÿ
8: ÿh(v0) = ÿ, then add v0 to ÿk **for all** v3 ÿ ÿk **if** ÿ123 does not intersect Fk
9:
10: **do**
eleven:
12: Add triangle ÿ123 to the grid: T k+1 = T k ÿ ÿ {ÿ123}
13: Update edge: Pk+1 = Pk \ ÿ123, Fk+1 = F(Pk+1) If Fk+1 = ÿ, then go to step
14: 20 Go to step 19 end if end for Put R = ÿ, go to **7** 18 :
15: 19: **end for** 20: Put T = T
16: k+1
17:

algorithm, and we will also propose several ideas that will make it possible to minimize this influence.

One of the main operations sensitive to computational errors is to check the intersection of a triangle with a segment. Incorrect definition of the intersection due to computational errors can lead to incorrect operation of the entire algorithm. Note that only two situations of obtaining an incorrect result are possible: 1) a non-intersecting triangle and a segment are incorrectly perceived as intersecting; 2) the intersecting triangle and the segment are incorrectly perceived as disjoint.

From the point of view of Algorithm 10, errors of the first type are not critical: a false intersection of a triangle and a segment means that the new triangle will not pass the test of intersection with the current front, and the corresponding candidate vertex will be rejected. On the other hand, errors of the second type can distort the check of the intersection of the triangle with the current front, which can lead to a self-intersecting front or to a nonconformal mesh. Thus, errors of the first type only increase the enumeration of candidate vertices, slightly slowing down the work

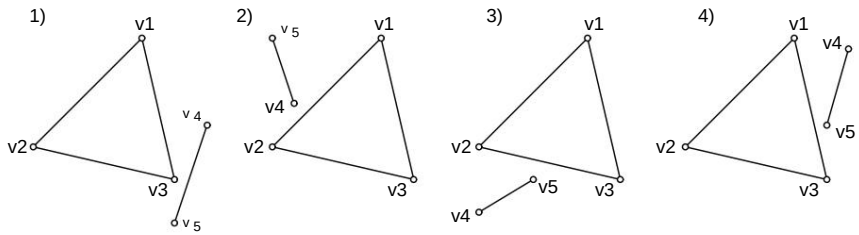
program, and errors of the second type can lead to an incorrect result of the program.

We will use an algorithm for checking the intersection of a triangle–segment pair, which excludes the occurrence of errors of the second type, but allows the occurrence of errors of the first type. In general, two planar convex objects do not intersect if and only if there is a line separating them. Under the conditions of Algorithm 10, common vertices and common edges are also allowed: for example, a segment and a triangle can have a common vertex, or a segment can be a side of a triangle. In these cases, the conformality of the grid will not be violated, so they must be considered as non-intersecting. Consider the triangle ÿ123 and the edge e45 (see Fig. 3.5). We will assume that

$$S\ ÿ123 > 0,$$

since only such triangles will be tested in Algorithm 10. First, we determine how many vertices a triangle and a segment have in common. Consider the case when there are no common vertices. The triangle and the sharp edges do not intersect if and only if either the triangle lies entirely in one of the half-planes with respect to the straight line e45, or e45 lies on the other side of the triangle with respect to one of its sides (see Fig. 3.5). Thus, if there are no intersections, then at least one of the following conditions is satisfied: 1) S ÿ451 > 0, S ÿ452 > 0, S ÿ453 > 0, or S ÿ451 < 0, S ÿ452 < 0, S ÿ453 < 0; 2) S ÿ124 < 0, S ÿ125 < 0; 3) S ÿ234 < 0, S ÿ235 < 0; 4) S ÿ314 < 0, S ÿ315 < 0.

The converse is also true: if one of these conditions is satisfied, then there are no intersections. These conditions can be written in a more convenient form: 1) sign(S ÿ451) + sign(S ÿ452) + sign(S ÿ453) = ±3; 2) sgn(S ÿ124) + sgn(S ÿ125) = ÿ2; 3) sgn(S ÿ234) + sgn(S ÿ235) = ÿ2; 4) sgn(S ÿ314) + sgn(S ÿ315) = ÿ2.



Rice. 3.5. Illustration of conditions 1), 2), 3) and 4) when a triangle and an edge do not intersect

Let's move on to the case when the triangle and the segment have one common vertex. In this case, the triangle and the segment do not intersect in our sense if and only if e45 lies in the negative half-plane with respect to one of the sides of the triangle, while their common vertex lies on the line, and the second falls into the negative half-plane. These conditions can be written as follows: 1) sgn(S ÿ124) + sgn(S ÿ125) = ÿ1; 2) sgn(S ÿ234) + sgn(S ÿ235) = ÿ1; 3) sgn(S ÿ314) + sgn(S ÿ315) = ÿ1. If at least one of these conditions is satisfied, then there is no intersection;

and vice versa, if there is no intersection,

then at least one of the conditions is satisfied.

In the latter case, when both vertices of the segment coincide with two vertices of the triangle, the edge e45 is a side of the triangle, and by our agreement, the triangle and the segment are considered to be non-peripheral.

split ends.

Suppose we have a function d(v1, v2, v3) that does not accurately calculate the sign of the expression S ÿ123 . But at the same time, if d(v1, v2, v3) = 0, then d(v1, v2, v3) = sign(S ÿ123), and in disputable situations d(v1, v2, v3) = 0. Then the intersection test can be performed using algorithm 11.

Algorithm 11. Checking the intersection of a triangle with a segment 1:

Find the common vertices of the triangle ÿ123 and the edge e45 2:
if there are no common vertices
3: **then** If d(v4, v5, v1) + d(v4, v5, v2) + d(v4, v5 , v3) = ±3, then the intersection no
4: If d(v1, v2, v4) + d(v1, v2, v5) = ÿ2, then there is no intersection If d(v2, v3, v4)
5: + d(v2, v3, v5) = ÿ2, then there is no intersection If d(v3, v1, v4) + d(v3, v1, v5) = ÿ2, then there is no intersection 6: 7: **end if** 8: **if** one common vertex **then**
9: If d(v1, v2, v4) + d(v1, v2, v5) = ÿ1, then there is no intersection If d(v2, v3, v4)
10: + d(v2, v3, v5) = ÿ1, then there is no intersection If d(v3, v1, v4) + d(v3, v1, v5) = ÿ1, then there is no intersection 11: 12: end if **13 : if** two common vertices **then** No intersection 15: **end if** 16: that the triangle
14: and the segment intersect

repent

We note that the admissible inaccuracy in the calculation of d(v1, v2, v3) can only lead to errors of the first type, while Algorithm 11 excludes errors of the second type.

The operation of determining the sign of the algebraic area S ÿ123 is the main operation used in both Algorithm 10 and Algorithm 11. Let us propose a method for inexact determination of the sign of d(v1, v2, v3) with the following property: if d(v1, v2, v3) = 0, then d(v1, v2, v3) = = sgn(S ÿ123). The calculation of S ÿ123 is reduced to finding the determinant of the 2 × 2 matrix:

$$2S\ \ddot{y}123 = e31 \times e32 = \begin{vmatrix} x1\ \ddot{y}\ x3 & y1\ \ddot{y}\ y3 \\ x2\ \ddot{y}\ x3 & y2\ \ddot{y}\ y3 \end{vmatrix} \ddot{y} \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad\ \ddot{y}\ bc = D. \quad (3.3.1)$$

When calculating the elements of the matrix, we already introduce some relative error. In addition to this, the relative error is again introduced in the expressions ad and bc, and at the end one more error is introduced into D. The total absolute error can be estimated from above as ÿ(|a| + |b| + |c| + |d| + 2| ad| + 2|bc|), where ÿ depends on machine precision. Taking this allowable error into account, we write an algorithm for determining sgn(S ÿ123). *Algorithm 12.* Calculation d(v1, v2, v3)

1: Calculate a = x1 ÿ x3, b = y1 ÿ y3, c = x2 ÿ x3, d = y2 ÿ y3 2: Estimate the error r = ÿ(|a| + |b| + |c| + |d| + 2|ad| + 2|bc|) 3: Calculate the determinant of the matrix D = ad ÿ bc
4: If D>r, then **return** 1 5: If D < ÿr, then **return** ÿ1, otherwise **return** 0

In Algorithm 12, the value D is calculated with an absolute error not exceeding r. Therefore, if d(v1, v2, v3) = 0, then we can be sure that the sign of D is calculated correctly and d(v1, v2, v3) = = sign(S ÿ123). In practice, if the type of real numbers with single precision is used to store coordinates in the

computer memory, and the calculations of the values a, b, c, d, D in Algorithm 12 are performed with double precision, then the machine precision is sufficient for the exact calculation of both all intermediate values , and the final value D. In this case, we can set ÿ = 0, and Algorithm 12 will calculate sgn(S ÿ123)

exactly.

3.3.3. Finiteness of Algorithm 10

Let us show that the number of operations in Algorithm 10 is finite. In the proposed algorithm, there are two explicit nested loops and one implicit one due to restarting the enumeration at step 18. The enumeration cycle at step 10 is always finite due to the finiteness of the set ÿk. will be found, and also show that the outer loop in step 2 will be final. R.

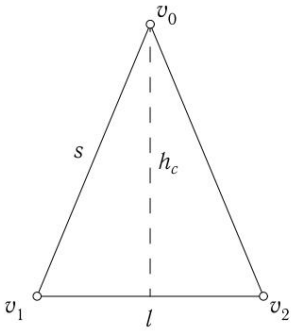
Let us prove that when searching for the candidate vertex v3 at step 10, there will always be at least one suitable triangle after restarting the search at step 18. It follows from Rado's theorem [80] that any polygon can be divided into triangles by diagonals. It follows from the existence of a triangulation without additional points that at least one triangle can be constructed for each side of the polygon. Let's formulate this statement.

Lemma 3.3.1. *Let* P *be a polygon with vertices* v1, v2, ... , vn *and* v1v2 *the side of P. Then there exists a third vertex of* v3 such *that the triangle* ÿ(v1, v2, v3) *lies entirely inside P.* If algorithm 10 reaches step 18, then the enumeration at step 10 becomes complete. From Lemma 3.3.1,

taking into account the previous remarks about the influence of computational errors and the possibility of exact calculation in Algorithm 12, it follows that after restarting the enumeration, a suitable triangle will always be found.

Let us show that the outer loop at step 2 will be finite, i.e., the number of constructed triangles is finite. To do this, we will estimate the number of triangles in terms of the number of grid nodes, and we will limit the number of nodes from above due to the boundedness of the polygon P and the lower limit on the distance between nodes vk i .

Let dk be the minimum of pairwise distances between points of the front pk i , ÿk be the minimum of pairwise distances between nodes from vk i , ÿ0 = ÿ. When adding a new triangle ÿ123 to T k, two situations are possible: v3 ÿ { pk } and v3 = v0. In the first case, the minimum distance between the grid vertices will not decrease more than to the minimum distance between the front vertices, then the minimum distance between the front vertices will not decrease: ÿk+1 min(ÿk,dk), dk+1 dk. Consider the case v3 = v0. Since ÿ120 does not intersect Fk, the distance from v0 to the set {vk } is not less than the distance from v0 to Fk, which, by construction, is not less than h chosen at Step 6. The distance from v0 to {pk } is not less than r. Accordingly, ÿk+1 min(ÿk, h), dk+1 min(dk, r). Let us propose some heuristic method for choosing the parameters s, r, and h. Denote by l = | v1v2| the length of the segment v1v2, and through hc the height ÿ120 lowered from the vertex v0 (see Fig. 3.6).



Rice. 3.6. Triangle with diameter s, base l and height hc

Suppose we have some scalar function s(x, y): P ÿ R + and we want the size of the elements in the grid to change according to this function. In this case, we will say that the size of the triangles

is given by *a user-defined function* s(x, y). We require that the function s(x, y) be separated from zero by P: s(x, y) > smin > 0. Take

$$s = \max \quad \overline{5}\, l,\ s(x, y) \,.\, 9$$

Such a choice guarantees the existence of the point v0 and limits you to ÿ19 the triangle cell from below: hc l, hc 18 for the rate $\frac{\ddot{y}19}{10}$ s.

increase in the size of triangles:

$$s = \ddot{y}l, \qquad \ddot{y}\ 1.$$

In this case, we will say that the grid step is chosen *automatically.* Since ÿ > then the point v0 always exists, and hc = ÿ 3 l.

$$= \ddot{y}2\ \ddot{y} \quad \frac{1}{l\,4} \quad \frac{}{2}$$

Let's take the parameter h = $\frac{1}{hc}$, and calculate the parameter r using the formula 2

$$r = \frac{}{\ddot{y}2}\ s + s0 + (s\ \ddot{y}\ s0)2 + \ddot{y}2 \,, \qquad (3.3.2)$$

where ÿ > 0, s0 0 and ÿ are some parameters. Note that for s>s0 the value r ÿs, and for s<s0 the value r ÿs0. The parameter ÿ can be chosen so that r = s0 for s = s0. The constraint r ÿs makes it possible in practice to avoid the appearance of sharp differences in the sizes of neighboring segments in a new front, and the constraint r ÿs0 allows us to limit the value of r from below. When implementing the algorithm on a computer, the parameters introduced above

can be selected in one of two ways, depending on whether the size of the triangles is specified by the user or automatic size selection is used. Let's consider both cases.

Let the size of the triangles be set by the user. Let's put

$$\ddot{y} = \frac{1}{2}, \ s0 = 0, \ \ddot{y} = 0.$$

Then r = $\frac{1}{}$ s. By our assumption, s>smin, hence 2 smin. Then dk min d0, r> $\frac{1}{}$ $\frac{1}{2}$ smin for any k. By construction

h $\frac{2}{\ddot{y}19}$ s and s>smin, so for any k

$$\ddot{y}k \ \min d0, \ smin \,.\, \frac{\ddot{y}19}{20}$$

Let now the size of the triangles is selected automatically. Let's take

$$\ddot{y} = \frac{1}{2},\ s0 = d0,\ \ddot{y} = 2s0(1 \ddot{y} \ddot{y})/\ddot{y}.$$

Then r $\frac{1}{2}$ d0, i.e., dk > ÿ $\frac{1}{2}$ d0 for any k. By construction h $\frac{\ddot{y}\sqrt{3}}{4}$l,

a l dk, so h $\frac{3}{4}$d0. Therefore, for any k 8

$$\ddot{y}k \quad \frac{\ddot{y}\sqrt{3}}{4}d0.\ 8$$

We have just shown that there exists ÿ > 0 such that ÿk > ÿ is true for any k. With this condition, we can estimate the maximum number of mesh nodes. The set of mesh nodes {vk } lies in the closure of P, and the pairwise distance between them is greater than ÿ. Let Nk be the number of nodes in the set {vk }. Let's build in each

node circle radius $\frac{\ddot{y}}{2}$. These circles do not intersect and cover the plane 2 ÿ2 .

spare equal to Nk $\frac{\sqrt{3}}{4}\ddot{y}2$. We extend the original polygon P by ÿ/2,

Pÿ = v3 : dist(v3,P) < circles. $\frac{\ddot{y}}{2}$ (See Figure 3.7). Then Pÿ will completely cover Let us

estimate the area Pÿ. Each side of a polygon of length l has a rectangle, which gives an increase in area no more than ÿ l. Thus, the increase in area due to rectangles 2 how

will be $\frac{\ddot{y}}{2}$p(P), where p(P) is the perimeter of the polygon P. Circular 2 sectors at the vertices with an internal angle ÿ give an increase in area

ÿ2 is not greater than (ÿ ÿ ÿ) , and this estimate is also true for interior angles greater than 8 ÿ; in this case, the gain becomes negative, but it is completely covered by the overlap of the two rectangles that we considered earlier. From the formula for the sum of the angles of a polygon, the contribution of all circular sectors can be estimated as

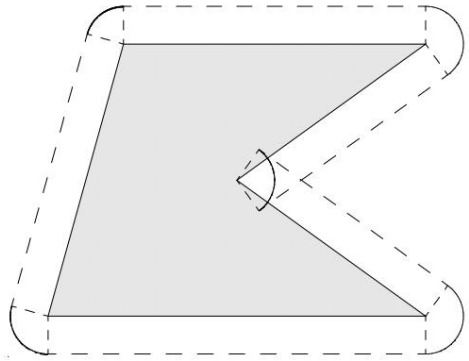
$\frac{\ddot{y}}{4}\ddot{y}2$ K(P), where K(P) is the number of connected components in P. We have

S(Pÿ) S(P) + $\frac{\ddot{y}}{2}$ ÿ2 p(P) + ÿ$\frac{\sqrt{3}}{4}$ K(P). From here, one can estimate the number of vertices:

$$Nk_v \quad \frac{4S(P)}{\ddot{y}\ddot{y}2} + \frac{2p(P)}{\ddot{y}\ddot{y}} + K(P).\ \ddot{y}\ddot{y}$$

Note that a similar estimate is also true for the number of vertices in the front Fk:

$$Nk_p \quad \frac{4S(P)}{\ddot{y}\ddot{y}2} \frac{2p(P)}{\ddot{y}\ddot{y}} + K(P).\ + \qquad (3.3.3)$$



Rice. 3.7. Polygon extension by $\frac{\ddot{y}}{2}$

From (2.2.5) it follows that the number of triangles in T k limited:

$$Nk_t \quad \frac{8S(P)}{\ddot{y}\ddot{y}2} \frac{4p(P)}{\ddot{y}\ddot{y}} + 2K(P)\ \ddot{y}\ 5.\ + \qquad (3.3.4)$$

This estimate limits the number of iterations in the outer loop of the algorithm to 10, which in turn proves that the number of operations of the algorithm is finite.

3.3.4. Speed of the Advanced Front Algorithm Let us briefly analyze the

speed of Algorithm 10. Formula (3.3.4) estimates from above the number of iterations of the outer loop at step 2. Denote by Ne(Fk) and Ne(Fk R) the numbers of segments in Fk and Fk, respectively. Let us write down all non-trivial operations that are used in Algorithm 10.

1. Selecting a segment with the minimum length from Fk (line 3). 2. Construction of the local front Fk (line 7). (line 8).
3. Constructing a list of candidates ÿk 4. Checking the intersection of a triangle with a local front (str ka 11).
5. Front update (line 13).

Operation 4 is performed in Ne(Fk R) operations of checking the intersection of a triangle with a segment (see Algorithm 11). Operation 3 can be performed in 2 Ne(Fk R) operations to check whether vertices from Fk R belong to the neighborhood ÿ R(v0). The complexity of operations 1,

2, and 5 depends on the data structures used. To store the front, we introduce an ordered list with the minimum length element at the root. To quickly find a local front, we will use a quaternary search tree. In this case, operation 1 becomes trivial, and the complexity of operation 2 will, on average, be proportional to the sum log Ne(Fk) + Ne(Fk R). Operation 5 consists of three operations of adding or removing a segment from the front, the complexity

each of them is on average proportional to log Ne(Fk). The structure of the search tree is discussed in more detail in § 2.3.

Let us summarize the available results on the computational complexity of Algorithm 10. Let the input be a polygon P completely covered by a square with side H. If the local grid step is given by a custom function of the desired triangle size s(x, y), then assume again that it is separated from zero: s(x, y) smin > 0. Vp. 3.3.3 it was shown that there is a parameter ÿ, which can be used to estimate the maximum number of triangles in an area using formula

(3.3.4). Let us estimate from above the computational complexity of one iteration of the outer loop in Algorithm 10. We will be interested in the dependence of the complexity of the algorithm on the grid step and, accordingly, on ÿ. Assume that a suitable triangle has been found in a local enumeration. In practice, R is chosen in the range from s to 2s, and the lengths of the front segments in this neighborhood are comparable to s. The

average estimate for the number of segments in the local front Ne(Fk R) turns out to be independent of ÿ. However, in the worst case, it can be proportional to Ne(Fk), which, in turn, worse

In our case, it reaches $4ÿÿ2\,\overline{S(P)} + 2ÿÿ\,p(P)\,\overline{+\,K(P)}$. Search local front is on average proportional to $Ne(Fk\ R) + \log 2 \frac{H}{ÿ}$.

Further, for each vertex of the local front, it is required to check the intersection of the candidate triangle with the local front. The complexity of this operation is on average proportional to (Ne(Fk R))2, which is a quantity independent of ÿ. But in the worst case, the complexity of the operation can reach a value proportional to (Ne(Fk))2.

The exhaustive search, when algorithm 10 goes to step 18, corresponds to the worst estimate for the local search.

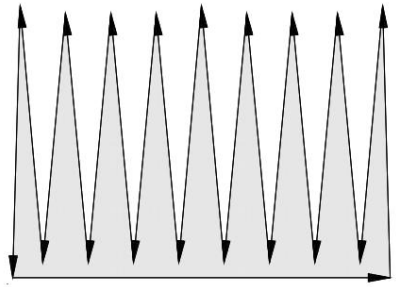
The overall complexity of the algorithm is on average proportional to

$$Wavg = \frac{4S(P)}{ÿÿ2} \quad \frac{2p\,(P)}{} + K(P)\log 2 + ÿÿ\,\frac{H}{ÿ}.$$

In the rare worst case, it is proportional to

$$wbad = \frac{4S(P)}{ÿÿ2} \quad \frac{2p\,(P)}{ÿÿ} + K(P) + \quad ^{3}.$$

In conclusion, we present several areas for which the complexity of the algorithm is much worse than the average estimate. Consider the area in the form of a comb, shown in fig. 3.8. We will increase the number of teeth N in it so that the area will remain approximately the same, and the perimeter will be proportional to N. The parameter ÿ will decrease inversely proportional to N. The number of segments in the initial front will be 2 N + 1. At each step local



Rice. 3.8. Worst edge configuration example

the neighborhood will almost completely cover the entire area, so the complexity of one iteration of the outer loop will be proportional to N2. In total, (2N ÿ 1) triangles will be built. The total complexity of the work will be proportional to N3, which is much worse than the average estimate of the order of N log N.

3.3.5. Experimental results

In this section, we will illustrate the behavior of Algorithm 10 with several examples. First, we experimentally estimate the average complexity of the algorithm, and then we present the results of using different methods for setting the desired grid spacing. To experimentally measure the speed of the advanced front algorithm,

we will take a unit square and build unstructured quasi uniform grids in it with step h. Decreasing the grid step h, we will monitor the number of triangles in the grid Nf and the grid construction time t. The results of the experiments are presented in table. 3.1.

T a b l e 3.1

Speed of the advanced edge algorithm

h	Nf	t, s Nf /t, sÿ1 t/(Nf log Nf), µs
0.01	19304	0.29 66566 1.52
0.005	128334	2.08 61699 1.38
0.0025	512956	8.96 57250 1.33
0.00125	2019486	37.67 53610 1.28

It can be seen from the calculation results that the operating time is proportional to the value of Nf log Nf . Note that the proportionality factor is close to 1. The table also includes a column for the meshing speed (triangles per second).

We now analyze how the choice of the local grid spacing, or parameter s, when constructing a new triangle in Algorithm 10 affects the grid and its quality.

Consider a unit square with a circle of radius 0.1 cut out in the center. The algorithms proposed in this subsection make it possible to specify the desired size of grid elements using the scalar function s(x, y). Using the same function, one can construct a discrete boundary of a region given analytically. This will be discussed in more detail in the next paragraph. When constructing the grids, two scalar functions were used: s1(x, y) and s2(x, y), which are responsible for the desired

size of the triangles. The first function decreased near two semicircles resembling in shape the graph of the function sin(2ÿx). The second function was identically equal to a constant on the entire region. The grids obtained with their help are shown in Figs. 3.9, *a* and *b,* respectively.

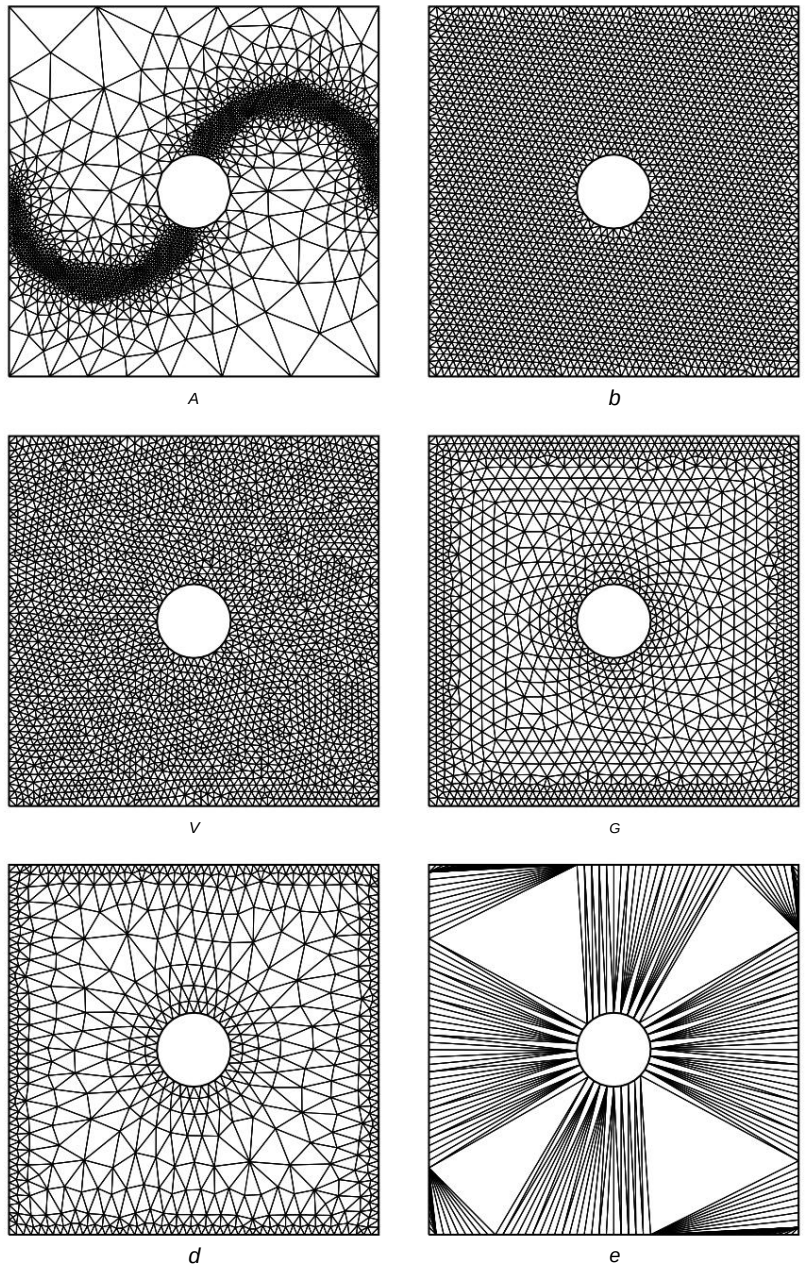
Next, we will test the automatic selection of the grid size for a given discrete area boundary. As a discrete boundary, we take the boundary obtained in the previous experiment with a constant function s2(x, y). Let us test the operation of the algorithm with automatic step increase for different values of ÿ = 1; 1.05; 1.25; 20. The resulting grids are shown in fig. 3.9, *c–e.* The choice of a larger value of ÿ leads to a rapid sparseness of the grid inside the domain. Therefore, for a very large value of ÿ, we obtained an irregular grid with a minimum number of nodes inside the region (see Fig. 3.9, f).

Vtab. 3.2 contains information about the number of triangles in the constructed grids. Recall that the quality of an isosceles right triangle [see formula (2.1.3)] is approximately 0.89, and the quality of an equilateral triangle is 1. The quality of the mesh Q(ÿh) is equal to the quality of the worst triangle in it. Note that in all the considered examples, except for the last one, Q(ÿh) > 0.6, which is typical for high quality regular grids.

T a b l e 3.2 The number of triangles and the worst quality of triangles for different methods of choosing the grid step

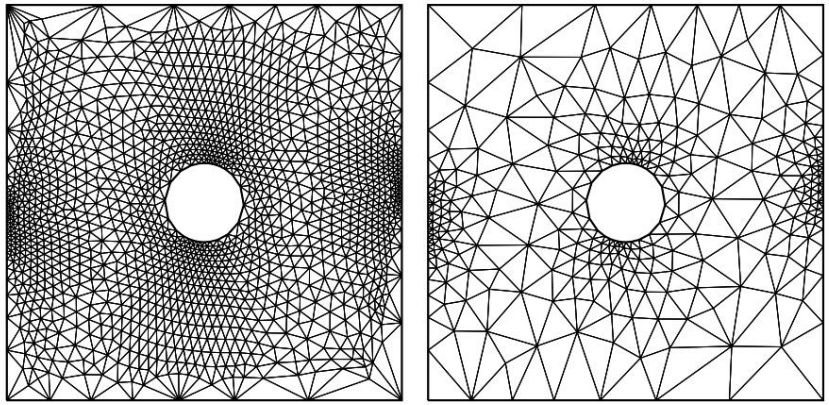
	s1(x, y)	s2(x, y)	ÿ = 1	ÿ = 1.05	ÿ = 1.25		ÿ = 20
Nf	3636	6212	5386	2596		1194	232
Q(ÿh)	0.732	0.829	0.633	0.877		0.778	0.641 10ÿ2

In addition to the considered grids with a uniform trace on the boundary, let us check Algorithm 10 for nonuniform boundary discretization. As such a discretization, we choose the trace of the triangulation obtained using the function s1(x, y) (see Fig. 3.9, a). Boundary discretization has a non-constant, but smoothly varying step. Results of automatic step selection with parameters ÿ = 1.05



Rice. 3.9. Different ways to choose the grid step: *a* — non-trivial function s1(x, y); *b* is a constant function s2(x, y); (c–f) automatic selection with ÿ = 1; 1.05; 1.25 and 20

and ÿ = 1.25 are shown in Figs. 3.10. The number of triangles in the resulting grids was 2904 and 550, respectively. The worst triangle quality was Q(ÿh) = 0.334 and Q(ÿh) = 0.696 respectively
actually.



A

b Fig. 3.10. Automatic selection of the grid step for uneven discretization of the boundary: ÿ = 1.05 (a), ÿ = 1.25 *(b)*

Using geometric criteria for choosing the function s(x, y), we can construct grids adapted to the features of the computational model. Decreasing the value of s in some subdomain, for example, in the prefracture zone, we will refine the mesh there. A more complicated approach to constructing the function s(x, y) is based on estimating the error of the finite element solution of the differential problem—see the methods described in the appendix. For example, s(x, y) can be adjusted to some error rate.

§ 3.4. Construction of a surface triangulation by the advancing front method

The technology for constructing triangular *surface* meshes, i.e., meshes at the boundary of three dimensional regions, is necessary for the further construction of unstructured tetrahedral meshes. Surface triangular meshes can also be used to solve quasi-2D problems on curved surfaces. Examples of such problems are the problems of deformation of thin-walled structures or the solution of shallow water equations on the surface of a sphere or geoid. In this section, we will show that the 2D advanced front algorithm can be extended to the case of curved surfaces, and also discuss methods for constructing discrete curvilinear boundaries and interacting with CAD to obtain the necessary

border information. We examine the finiteness and complexity of the presented algorithms and give several examples of their operation.

A surface is a two-dimensional manifold in a three-dimensional space. *A simple* surface is a surface that is homeomorphic to the unit square.

A curve on a surface or in space is a one-dimensional manifold on a surface or in space, respectively. *A simple* curve is a curve that is homeomorphic to a unit segment. Whenever possible, we will use the notation from § 3.3.

The area of a triangle ÿ123 on the surface is calculated as follows:

$$S_{ÿ123} = \frac{1}{2}(e31 \times e32)\, n3,$$

where n3 is the outward unit normal to the surface at the point v3, and a · b is the scalar product of two space vectors. We note that for the plane the new notation goes over into the definition from § 2.1. We also note that such a quantity corresponds to the value Sÿ(v are orthogonal

$_{1,\ ÿ}$ v3) in the plane tangent to v3, where v $_{1}$ and v2
projections onto the plane of the points v1 and v2, respectively.

3.4.1. Surface representation

Consider first a simple surface. According to its definition, there must exist a parametrizing function that takes a point from the unit square in the parametric space to a point on the surface. The construction of such a homeomorphism for an arbitrary surface is a difficult task. In practice, it is easier to construct a homeomorphism from some bounded domain ÿ ÿ R2 that is homeomorphic to the unit square.

The splitting of the boundary ÿ under a homeomorphism generates a splitting of the boundary of a simple surface, which can be regarded as a discrete boundary of the surface. Using this boundary as a front, one can apply the advanced front algorithm, treating the surface locally as a plane. A complex surface can usually be cut into several simple surfaces. Having fixed the
discretization on cuts, it is possible to construct a triangulation consistent with this discretization for each simple surface, then the general triangular mesh for the entire complex surface will be conformal.

We will compose an algorithm of actions based on the idea described above. The whole surface is divided into several simple surfaces ÿi, we will call them *curvilinear faces.* Each curvilinear face is parametrized by a smooth vector function ÿi = (x, y, z): (x, y, z) = Fi(p, s), (p, s) ÿ ÿi ÿ R2

.

Let us assume that the vector functions Fi have continuous first derivatives: Fi ÿ (C1(ÿi))3. We divide the

boundary of a curvilinear face into several simple curves ÿk, which we will call *curvilinear edges.* In we parametrize on curvilinear edges: ÿk = (x, y, z): (x, y, z) = Gk(t), t ÿ [tk,t We will also need mappings pik,sik

from the parametric ^{0} ^{1} k].

curve space into parametric surface space:

Gk(t) = Fi(pik(t),sik(t)), t ÿ [tk]. k,t Therefore, ^{0} , ^{1}

practice, this approach is applicable for fairly simple domains.

When implementing this method on a computer, you can use a different approach. Let us consider the parametrization of curvilinear edges in the parametric space of the surface separately for each curvilinear face. Let the face ÿi be parametrized by the function Fi(p, s), then the edge ÿk can be considered as a curve in the parametric space (p, s). For example, if we use the mapping p ÿ (p,sik(p)), then the curvilinear edge will be parametrized as follows:

ÿk = {(x, y, z): (x, y, z) = Fi(p,sik(p)), p ÿ [p0 i ,p1 i]}.

Such parametrizations of the curve ÿk may be different for different curvilinear faces containing ÿk, but must provide mathematically equivalent representations of the curve. In practice, it is possible to allow a slight discrepancy, within the permissible error, of representations of the same curve by different parametrizations, which simplifies the construction of parametrizations and expands the class of domains for which this approach is applicable. To construct surface meshes on curvilinear faces, a discretization of curvilinear edges is first

constructed. Consider a parametrized simple curve:

(x, y, z) = F(p,s(p)) = G(p), p ÿ [p0,p1].

We will build points on it:

vi = G(ti), i = 0, 1, ... , n.

The values of the parameters ti ÿ [p0,p1] are chosen using the bisection method in accordance with the desired distance between the points vi.

After constructing the discretization of the boundary of the curvilinear face, we can apply an analogue of the advanced front algorithm to construct a triangular mesh. Since neighboring faces have the same discretization of their common edge, and the triangular ones built on them

grids are consistent with this discretization, then the general grid composed of them will be conformal.

Let us write down the main stages of constructing a surface grid. Let us suppose that the surface is divided into several non-intersecting simple surfaces—curvilinear faces. The curvilinear boundary of each face is divided into several simple curves—curvilinear edges. We will consider only conformal partitions, i.e., those in which two neighboring faces have common curvilinear edges. At the first stage, discretization is constructed for all curvilinear edges. To do this, points are placed

on the curved line, and the curve is approximated by a broken line. The length of the segments is controlled by the desired size of the grid elements, while the position of the points is calculated using the bisection method.

At the second stage, for each curvilinear face, a discrete boundary is compiled from the available edges. For points on edges, the values of parameters (p, s) are restored in the parametric space of the face. Finally, using the advanced front algorithm, a surface triangulation is constructed that is consistent with the discrete boundary. We formalize these ideas in Algorithm 13.

Algorithm 13. Construction of a surface mesh 1: **for all**

curvilinear edges **do** Choose one of the
2: parametrizations of a simple curve Construct a discretization of the curve using the bisection method
3: 4: **end**
for 5: **for all** curvilinear faces **do** Compose
6: the discretization of the boundary from the discretizations of the edges
7: Compute the parameterization of the nodes of the discrete boundary in the parametric space of the face
8: Apply the advanced front method to build the mesh 9: **end for** 10: Merge all the meshes
built into one common mesh

Consistency of surface meshes of faces with discretizations curvilinear ribs, the conformality of the overall mesh is guaranteed.

3.4.2. Interaction with the geometric CAD core

Information about the region boundary can be obtained using the CAD geometric kernel [4]. Most of the existing CAD systems offer an interface for interacting with their internal geometric core, allowing you to obtain information about the topology and geometry of the area. Each CAD uses its own interface for this interaction, and there are no common standards in this area yet. Some open source CAD is good

documented. One example of such a CAD can be the open system Open CASCADE Technology [19]. However, in most cases, the closed nature of the source code and the lack of documentation on interfaces in the public domain complicate the development of interfaces with CAD. A promising direction is the use of intermediate libraries that provide a common

unified interface for the developer and support interaction with different CAD systems. During the development of the commercial package CUBIT Tool Suite, which includes, in particular, its own ACIS geometric kernel, developers began to add new interfaces to other CAD systems. For this, a special layer was created that provides a common interface for interacting with CAD. Later, this part of the code was removed from a commercial project as a separate open library Common Geometry Module (CGM) [20, 82]. At that time, interfaces to the geometric kernels of the ACIS and Pro/ENGINEER systems were developed. The openness of the source code made it possible to use and improve this library. The CGM project is currently being developed under the new name CGMA [21]. This version adds the ability to interact with the Open CASCADE geometric kernel.

CGMA offers a universal interface for communicating with geometric CAD kernels. In this case, all information is divided into two parts.

- Topological information about the model: component parts of the model and their topological relations with each other.
- Geometric information: coordinates, dimensions, parameters and parametrizing functions for curves and surfaces. Most CAD kernels use boundary

representation of models (B-Rep or BREP is short for boundary representation). This representation method is also used in the CGMA library. The model consists of parts that form a tree-like hierarchy. Let's look at these parts, moving from simple to more complex.

1. A point for which its coordinates in space are given

   (x, y, z).
2. An edge is a part of a smooth parametrized curve bounded by points. For the edge, a certain parametrization t ÿ (x, y, z) is defined, as well as the points limiting it and the values of the parameter t at these points.
3. A loop is a connected and closed set of edges. It does not carry any geometric information and is a purely topological object.
4. A face is a part of a smooth parametrized surface bounded by a loop. The edges of the loop must lie on the surface of the face. The face is given by the surface parametrization (p, s) ÿ (x, y, z).

5. A shell is a connected and closed set of faces. Just like a loop, it is a purely topological object. 6. The body is a part of space bounded by a shell. 7. A set of bodies representing the model as a whole. As noted earlier, in order to construct a three-dimensional triangular

surface mesh, each face has its own triangulation, and the triangulations of faces are conformally connected on common edges. To do this, we first construct an edge discretization. Edges are approximated by broken lines with a space step specified by the user. After that, a triangulation is constructed for each face using the advanced front algorithm [54]. The discretization of edges acts as the initial front.

Particular attention should be paid to curved surfaces. Geometric CAD kernels often create faces with periodic parametrization, as well as with parametrization having singular points. For example, Open CASCADE defines the lateral surface of a cylinder with one face with periodic parametrization, and the lateral surface of a cone with a face with a singular point at the vertex of the cone. Moreover, the surface of the ball is given by one face with a periodic parametrization and two singular points at the poles; the face itself is limited to only one edge connecting the two poles. The advancing front algorithm should be modified accordingly to allow for such peculiarities. Examples of how the algorithm works with geometric CAD models will be given in Section 3.4.4.

3.4.3. Advance Front Algorithm

The advancing front algorithm 10 is extended to the case of curved surfaces. In this subsection, we only note the main features of the advancing front algorithm for such surfaces. Vp. 3.3.2, algorithm 11 was proposed for checking the intersection of a triangle with a segment. This algorithm used only the function of determining the sign of the expression S ÿ123 . Let us show how to get the sign of the expression S ÿ123 on the surface. Let us construct the point v = v3 + n3 so that n3 = v ÿ v3 is the outward normal to the surface at the point v3. By our definition

$$2S_{\ddot{y}123} = (e31 \times e32)\, n3 = \begin{vmatrix} xv1 \ddot{y} xv3 & yv1 \ddot{y} yv3 & zv1 \ddot{y} zv3 \\ xv2 \ddot{y} xv3 & yv2 \ddot{y} yv3 & zv2 \ddot{y} zv3 \\ xv \ddot{y} xv3 & yv \ddot{y} yv3 & zv \ddot{y} zv3 \end{vmatrix}_{\ddot{y}} \begin{matrix} abc \\ def \\ hi \end{matrix}_{\ddot{y}} .$$

The sign of the determinant D = aei ÿ ce g + bf g ÿ afh + cdh ÿ bdi of a 3 × 3 matrix is calculated by an algorithm similar to Algorithm 12. The absolute error can be estimated as r = ÿ(|a| + |b| + |c| + + |d| + |e| + |f| + | g| + |h| + |i| + |aei| + |ce g| + |bf g| + |afh| + |cdh| + + | bdi|). Note that if single precision is used for the coordinates of the nodes of real numbers, and for calculating the determinant

and intermediate values \u200b\u200b- double precision, then there may not be enough margin of accuracy to obtain an exact answer. Accurate calculations require triple or quadruple precision. Some modern computer architectures allow quadruple precision calculations, and some compilers can use software implementation of quadruple precision on conventional processors.

Algorithm 11, taking into account the previous remark about calculating S ÿ123, can be used to check the intersection of a triangle and a segment on a surface. In fact, we will check the intersection of the projection of the triangle and the projection of the segment on the tangent plane, so this method is applicable only in local neighborhoods in which the surface differs little from the tangent plane.

To calculate the value of S ÿ123, you need to be able to build a normal to the surface. Let the surface be parametrized as follows: (p, s) ÿ (x, y, z), and in some neighborhood of the point v3 the first derivatives of the components x, y, z with respect to the parameters p and s exist and are continuous. Then the direction of the normal can be calculated using the following formula:

$$n3 = k \left(\frac{D(y,z)}{D(p,s)}, \frac{D(z,x)}{D(p,s)}, \frac{D(x,y)}{D(p,s)} \right)^T, \qquad (3.4.1)$$

Where

$$\frac{D(y,z)}{D(p,s)} = \begin{vmatrix} \frac{\ddot{y}y}{\ddot{y}p} & \frac{\ddot{y}y}{\ddot{y}s} \\ \frac{\ddot{y}z}{\ddot{y}p} & \frac{\ddot{y}z}{\ddot{y}s} \end{vmatrix},$$

$\frac{D(z,x)}{D(p,s)}, \frac{D(x,y)}{D(p,s)}$ are defined similarly, k is the normalizing set inhabitant.

In formula (3.4.1), all three components can simultaneously turn to 0. In this case, to determine the normal, one can choose derivatives along other directions in the parametric space. Let's conditionally write it like this:

$$n3 = k \left(\frac{D(y,z)}{D(p\,,s\,)}, \frac{D(z,x)}{D(p\,,s\,)}, \frac{D(x,y)}{D(p\,,s\,)} \right)^T,$$

where, for example, p = p ÿ s and s = p + s. In the case where the point v3 is a singular point of the parametrization, one can retreat a small distance from the point v3 and calculate the normal at the neighboring point.

Another important operation that is easily performed on the plane is the construction of an isosceles triangle ÿ(v1, v2, v3) on a given edge e12 with sides of length l. To find the position of the vertex v3, we will look for a point in the parametric space (pv3 , sv3) such that |v1v3| = |v2v3| = l in 3D

space. Let the vertex v1 be parametrized by the point (pv1 , sv1). We will look for the parametrization of the point v3 in the form

$$(pv3\,,\, sv3\,) = (\, pv1 + \ddot{y}\cos\ddot{y},\; sv1 + \ddot{y}\sin\ddot{y}), \qquad \ddot{y} > 0,\ \ddot{y}\ \ddot{y}\ [0,\ 2\ \ddot{y}].$$

For a fixed ÿ, one can choose ÿ so that |v1v3| = l with some accuracy. The bisection method will speed up the search for such a point. Depending on the result of the side comparison |v2v3| with the desired length l, we will increase or decrease ÿ until we achieve the equality |v2v3| = l with some accuracy. The bisection method will speed up the search for an acceptable direction ÿ. The use of two nested bisection methods results in a large amount of computation required to construct an isosceles triangle.

Therefore, in practice, the advancing front method on the surface is slower than the analogous method on the plane. Next, we will briefly analyze the advanced front algorithm for constructing surface meshes.

When analyzing the advancing front algorithm, attention must be paid to two points: the existence of a suitable triangle at each step and the finiteness of the number of steps. Unfortunately, the conclusion about the existence of a suitable triangle, obtained in Section 3.3.3, does not carry over to the case of a surface polygon. Moreover, for sufficiently large surface curvature and large lengths from cuts in the front, a suitable triangle may not exist on a given edge. In practice, provided that the curvature of the surface is limited and the lengths of the front segments are sufficiently small, a suitable

triangle is usually found. Moreover, it is among the candidates in some small neighborhood of the considered edge. If the curvature of the surface is bounded and the local neighborhood is sufficiently small, then the projection of the surface onto the tangent plane will be a one-to-one mapping. In this case, an analog

of Algorithm 11 can be used to find intersections of a triangle with a local front. Using Algorithm 10, we can maintain a lower bound on the minimum pairwise distance between vertices of a surface mesh. The arguments from Sec. 3.3.3 are applicable in this case up to the curvature of the surface and the Euler characteristic of the surface, which for simple surfaces

is equal to the Euler characteristic of the plane [13].

The analysis of the complexity of the advancing front algorithm from Sec. 3.3.4 is also applicable in the superficial case. Although the finiteness of the algorithm has not been proven in the general case, in practice, when working with surfaces of limited curvature and with a sufficiently fine mesh step, the complexity of the algorithm is on average proportional to Nf log Nf , where Nf is the total number of triangles.

3.4.4. Experimental results

In this section, we will carry out two numerical experiments. In the first experiment, we estimate the complexity of the advanced front algorithm for curved surfaces given analytically. In the second experiment, we will show the possibility of using the interface with the CAD geometric kernel when constructing a surface mesh. Let us measure experimentally the speed of the algorithm of the advancing front on the surface. Let us take the upper hemisphere of the unit sphere as the surface. Let us specify the parametrization of the hemisphere analytically using the map (p, s) ÿ (p, s, 1 ÿ p2 ÿ s2), where p2 + s2. We will construct a quasi uniform grid with step h. Decreasing the grid step, we will monitor the number of triangles in the grid Nf and the grid construction time t. The 1.
results of the experiments are presented in table. 3.3.

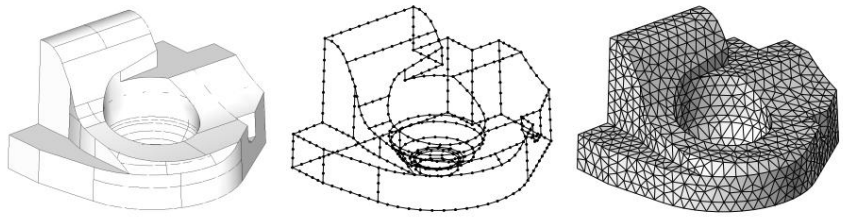
T a b l e 3.3 The speed of the advanced front algorithm on the surface

h	Nf	t, s	Nf /t, sÿ1	t/(Nf log Nf), µs
0.1	1298	0.49	2649	121
0.05	5460	2.18	2505	107
0.025	22152	9.36	2367	97
0.0125	89730	40.34	2224	91

It can be seen from the calculation results that the operating time is limited by Nf log Nf . The proportionality coefficient (the last column in the table) is quite large. The table also includes a column for meshing speed (triangles per second).

Let's demonstrate the joint operation of the surface triangular mesh generator and the geometric CAD kernel - OpenCASCADE. As an example, consider the 29_misc1 model from the Open CASCADE website [19]. The geometric model consists of 63 vertices, 96 curvilinear edges and 36 curvilinear faces (see Fig. 3.11, a).

First, a discretization of all curvilinear edges of the geometric model is constructed. The constructed discretization is shown in fig. 3.11b . It consists of 461 nodes and 494 segments. Further, for each of the 36 curvilinear faces, the algorithm of the advancing front on the surface is launched. Depending on the size of the curved face, the number of constructed triangles ranges from 3 to 641. The final surface triangulation contains 2594 triangles and 1299 nodes. The resulting surface triangulation is shown in fig. 3.11, c.



A *b* *V*

Rice. 3.11. Model specified in CAD: *a* - BREP model; *b* - discretization of curvilinear edges; *c* - surface quasi-uniform triangular mesh

Note that the speed of the algorithm that uses the interface with the CAD kernel strongly depends on the speed of calculating the parameterizing functions inside the CAD system. In the example above, the face with the most triangles (641) was part of a plane, and it took 2.11 seconds to triangulate, while the other curved face took 4.54 seconds to complete a triangular mesh consisting of only 25 triangles. . The total time for constructing the surface grid was 14 s.

§ 3.5. Method for improving a given surface mesh

Surface meshes obtained by export from many CAD systems have the property that the number of triangles in flat areas is minimal, and the triangles themselves can have a very elongated shape, which affects the regularity of triangulation of the initial front, the quality of the resulting tetrahedral mesh, and even the possibility of its build. *To construct high quality tetrahedral meshes, the initial front must be regular.* The proposed technology for improving the surface mesh is based on identifying almost planar connected subdomains in the original surface mesh and using the advanced front algorithm to cover them with

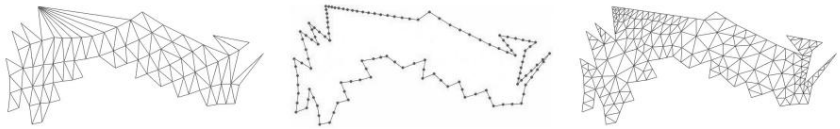
a regular mesh. The main idea of the proposed method is the selection of flat (or almost flat) pieces of the surface, which are called *polygons below,* and their repartition into new triangles of a regular shape. In this case, an insignificant deviation from the initial discrete surface is possible [7]. The input data

for this method are a conformal triangulation of the surface and a possible labeling of triangles, which ensures the division of the surface into polygons. Each polygon will be processed separately, and the geometric boundaries of the polygons will be preserved in their original form. Several options are available to the user to control the result of the repartition.

Among them is the rate of coarsening of triangles when constructing a new surface mesh using the advanced front algorithm. There is also a parameter responsible for the permissible degree of deviation in terms of
ligons from the plane.

The algorithm for constructing a regular triangulation of a given discrete surface is an iterative procedure (see Algorithm 14). First, the surface is divided into polygons, for which the problem is reduced to a two dimensional one, and then the triangulation of the entire surface is assembled from the resulting triangulations of the polygons. The main operations with the polygon are shown in fig. 3.12.

Algorithm 14. Rebuilding the surface mesh 1: **while** the original triangulation is not exhausted **do** Select a connected subdomain (polygon), whose triangles 2: swarm lie in the same plane within the specified accuracy Rotate the polygon plane to the Oxy plane and set the z-
3: coordinates to zero, map the polygon to plane Oxy Split the polygon boundaries and apply the advanced front method to construct a 2D regular triangulation of the area
4: with a given trace on the boundary Project (along the Oz axis) each node of the new triangulation onto the rotated polygon. Since the latter slightly deviates from the Oxy plane, the quality of the projected
5: triangles hardly deteriorates. Rotate the Oxy plane to the plane of the polygon to complete the inverse change of coordinates
6: Exclude selected polygon from initial triangulation 7: **end while**



A *V*

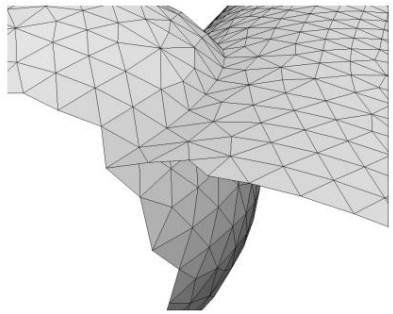
b Fig. 3.12. Selected polygon: *a* — original mesh in the polygon; *b* - new division of the polygon boundary; *c* - new mesh in the polygon

Let us consider the procedure for selecting a polygonal subdomain in more detail. A polygonal subdomain is a connected set of triangles that belong to the same plane with a given accuracy. The accuracy is determined by the maximum allowable distance h0 from the triangle to the plane and the maximum deviation ÿ between the normal to the triangle and the normal to the given plane.

The formation of a polygon begins with the selection of an initial triangle. For definiteness, we choose the not yet considered

the triangle with the largest area. Its plane is assumed to be the plane of the future polygon, its marker specifies the polygon marker, and its boundary is assigned to the current polygon boundary. We will call this plane *the reference plane.* Next, an iterative algorithm for constructing a polygon works. One of the boundary edges of the polygon is selected and all triangles

of the original mesh that border the polygon through this edge are checked. A triangle is added to a polygon if its marker coincides with the polygon marker, the distance from the reference plane to any of the triangle's vertices is less than h0, and the sine of the angle between the reference plane normal and the normal to the candidate triangle is less than ÿ. If the triangle satisfies all these conditions, then it is added to the polygon and a new boundary is built; otherwise, the boundary edge is marked as checked and the next one is considered. Only one triangle can be added through one edge. If there are several suitable candidate triangles, then none of them is added, and the edge is marked as checked. This situation is possible if the edge belongs simultaneously to several surface triangulations (see Fig. 3.13). After all boundary edges are marked, the polygon is considered to be built.

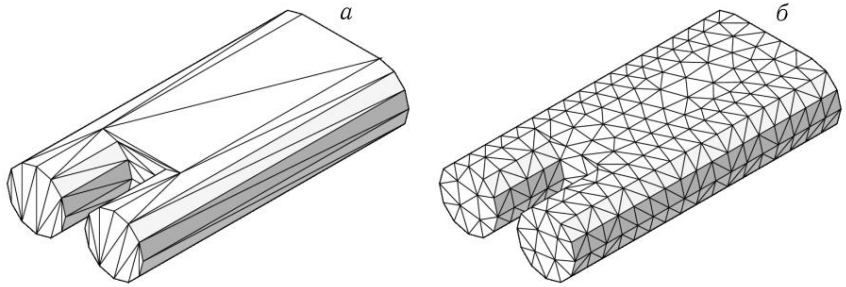


Rice. 3.13. Intersection of two surface triangulations

The discretization of the polygon boundary given by the initial triangulation may turn out to be highly nonuniform, since the initial triangulation may be irregular (in the sense of the shape of triangles). The inhomogeneity of the discretization of the boundary will inevitably lead to the appearance of triangles with very sharp corners near the boundary. Therefore, with a new division of the boundary, it is necessary to get rid of sharp drops in the division step. One solution may be to use a constant uniform step to discretize the polygon boundary. This approach

is convenient for constructing a quasi-uniform surface grid based on a given surface triangulation. However, in this case, the sampling step must be no less than the minimum length

boundary edge of each polygon. An example of improving the surface mesh is shown in fig. 3.14.



Rice. 3.14. Surface mesh exported from CAD (a) and improved surface mesh *(b)*

Another approach can be used, in which the discretization step of the polygon boundary is consistent with the local curvature of the surface. The calculation of the local curvature of a surface given by its triangulation is a separate difficult task. In this subsection, a method will be proposed that does not explicitly calculate the local curvature of a surface.

Assuming that the local size of triangles in the original mesh is consistent with the curvature of the surface, we will use the minimum sizes of existing triangles to construct a regular three angulation. For each node v of the original triangulation, we define the step parameter hv as the minimum height of all triangles converging at the node. The step parameter specifies the size of the regular grid at the vertices of the selected polygon. For each boundary edge of a polygon, we define a continuous function of the step parameter as a linear interpolation of the values hv1 and hv2 given above at the vertices of the edge. In this case, the location of the nodes of the new discretization along the edge e(v1, v2) is given by the formula

$$ri = v1 + \frac{v2 \ddot{y} v1}{|v1v2|} h_{v1} i + \sqrt{d2}\, i, \; i = 1, \ldots, [Q] \ddot{y} 1,$$

Where

$$d = \frac{h \ddot{y} h\, v2\, v1}{[Q]}, \quad h_{v1}\ Q = hv1\sqrt{\ }, \quad h_{v2} \qquad \frac{Q2|v1v2| = hv2}{[Q]\ hv1 + hv2}, \quad Q =$$

[Q] is the integer part of

Q. Note that after rotating the polygon and zeroing the z-coordinate of its vertices (step 3 of algorithm 14), it is possible to cross the boundary of the flat polygon that needs to be triangulated. This situation does not allow using the advanced front algorithm. In this case, the algorithm returns and the polygon is built

again, and when adding another triangle to the polygon, the possibility of self-crossing of the boundary is additionally checked.

When constructing a new grid, a uniform grid spacing can be used if the polygon boundary has been subdivided with a constant grid spacing. In the case of an uneven step on the boundary, a variant of the advanced front algorithm with automatic step selection and coarsening when moving inside the region is convenient. In practice, it is better to limit the grid spacing within a polygon to some maximum value. If it is possible to calculate the local curvature of a surface given by its triangulation, the local grid step can be matched to the local curvature of the surface.

Since the location of nodes on the edge of a polygon depends only on the values hv1 and hv2 at the vertices of the edge, and the mesh inside each polygon has a given trace on the boundary, the resulting surface mesh is conformal.

Improvement of the surface mesh is the most important stage in the technological chain of constructing high quality tetrahedral meshes. The quality of the volumetric mesh strongly depends on the quality of the surface mesh. Another example of the application of the method proposed here is the reduction in the number of triangles in too fine surface meshes. Such grids can be obtained, for example, as a result of three-dimensional scanning. They can have hundreds of thousands of triangles and are very inconvenient for constructing tetrahedral meshes. The surface enhancement method can be used to reduce the total number of triangles while maintaining the basic geometric features of the model.

§ 3.6. Construction of a tetrahedral mesh by the advanced front method

In this section, we will study an extension of the advancing front algorithm to three dimensions. The issues of its reliability from the point of view of inaccurate calculations and from the point of view of the possibility of advancing the front, the finiteness of the algorithm and the complexity of its work will be discussed. Let us introduce concepts and notation

similar to those used in § 3.3. Recall that f(v1, v2, v3), or f123, denotes a triangle with vertices v1, v2, and v3. Similarly, ÿ(v1, v2, v3, v4), or ÿ1234, denotes a tetrahedron with vertices v1, v2, v3, and v4. *By an oriented triangle* in space we mean a triangle with some fixed, up to an even permutation, order of

traversal of its vertices. *The positive half-space* with respect to the oriented triangle f123 is the set of points v for which

oriented volume Vÿ(v1, v2, v3, v) > 0. Similarly, *a negative half space* is the set of points for which Vÿ(v1, v2, v3, v) < 0. *A triangulation* in space is a finite set of triangles conformally connected through

common edges. Moreover, two neighboring triangles can lie in the same plane. If each edge of a triangulation belongs to exactly two triangles, then such a triangulation will be called *closed.* A triangulation whose triangles (as open sets) do not intersect each other will be called a triangulation *without self-intersections. By a polyhedron* in space we mean a bounded part of space whose boundary consists of one or more nonintersecting triangulations without self-intersections. *A simple polyhedron* is a polyhedron bounded by one closed triangulation. *The vertices* of the polyhedron will be

called the nodes of its triangulation, and *the faces* of the polyhedron will be called the triangles of the triangulation. Note that the faces of the polyhedral domains considered in § 2.2 can be any flat polygons. As in § 3.3, the faces of a polyhedron are divided into external and internal. In this case, on the outer faces, one can introduce an orientation so that the polyhedron lies in a positive half-space with respect to each oriented triangle. Such an orientation will be called *positive. A front* in space is a set of oriented triangles without self-intersections. Each polyhedron P can be assigned a front F(P). To do this, we introduce a positive orientation on the outer faces P, and assign to each inner

face f123 a pair of oppositely oriented triangles f123 and f321. The totality of all these oriented triangles forms the front F(P). We call the front F *closed* if there exists a polytope P such that F = F(P).

A tetrahedral mesh ÿh for a polyhedral domain P is a partition of this domain into disjoint tetrahedra. Recall that a mesh is *conformal* if any two of its elements either do not have common points, or have one common vertex, or have one common integer edge, or one common integral face. We say that a conformal mesh *is consistent with the boundary* P if each face of P is a face of some tetrahedron in the mesh. In particular, the inner faces of a polyhedron will have exactly two neighboring tetrahedra, and the outer faces will have exactly one. The set of mesh tetrahedra will be denoted by T

.

3.6.1. Advance Front Algorithm

Algorithm 10 of the advancing front is generalized to the case of a three-dimensional space. In this section, we note the main features of the 3D advanced front algorithm.

A distinctive feature of three-dimensional space is the existence of such fronts for which there are no suitable tetrahedra. For this reason, the algorithm adds the ability to mark faces for which no suitable tetrahedra were found and skip them during further enumeration. At the end of the algorithm, we obtain a set of tetrahedra T of the mesh ÿmesh for the polyhedral subdomain Pmesh ÿ P and a

front for the unsplit polyhedral subdomain Pout = P\Pmesh. Note that if the advance front algorithm fails, then the mesh will be built only in part of the area. In this case, Pout = ÿ, and the second algorithm must be applied to complete the meshing. The general sequence of actions is presented in Algorithm 15. Let us analyze in detail the main actions that are performed when the front advances.

Algorithm 15. Advanced Front Method in 3D Space 1: Set T 0 = ÿ, F0 = F(P) 2: **for** k = 0, 1, ... 3: If

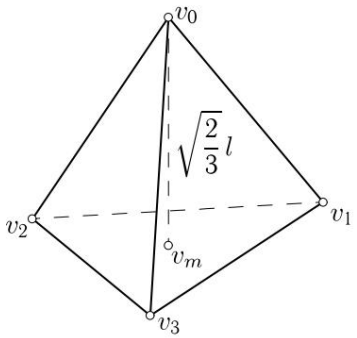
there are no unlabeled faces, then go to 22 **do**

4: Choose an unlabeled face f123 ÿ Fk with minimal plane spare
5: Determine desired edge length s of the tetrahedron Construct
6: a vertex v0 given s and Vÿ1230 > 0 Choose some R0 > s,
7: h > 0 and r > 0 **for** R ÿ { R0,ÿ} **do** Construct a local
8: front Fk R: Fk ÿ Fk ÿ ÿ R
9: (v0) Determine the set of candidate vertices ÿk R = {pk } ÿ ÿ ÿ R(v0)
10: If {pk } ÿ ÿr(v0) = ÿ and Fk ÿ ÿh(v0) = ÿ, then add v0 to ÿk R i

eleven: i R
12: **for all** v4 ÿ ÿk **if** R **do**
13: ÿ1234 does not intersect Fk R **then**
14: Add tetrahedron ÿ1234 to mesh, T k+1 = T k ÿ ÿ { ÿ1234}
15: Update edge: Pk+1 = Pk \ ÿ1234 and Fk+1 = = F(Pk+1) Go
16: to 21 **end if end**
17: **for** 18:
19: **end**
for Mark f123 and
T go to 4 20: 21: **end for** 22: Put T = k, Pout = Pk
and Fout = Fk

At step 4, the unlabeled face f123 with the smallest area is selected from the front and the vertex v0 is constructed in accordance with the chosen parameter l depending on s (see step 5). When implementing the algorithm on a computer, it is proposed to use the following heuristic method for choosing l and the position of the vertex v0. Let a, b, c be the lengths of the sides of the triangle f123. First, the center of mass vm of this triangle is constructed.

ka, then from vm along the normal to f123 at a distance l the 3rd vertex v0 is fixed (see Fig. 3.15). The l parameter is chosen based on a user-supplied function of the desired item size, or it is calculated automatically by the formula $l = \frac{a2 + b^2 2 + c}{3}$, where parameter responsible for the speed of automatic mesh coarsening.



Rice. 3.15. Tetrahedron with base f123 and vertex v0

Let us estimate the lengths of the edges e10, e20 and e30. For ÿ 1 the height of the $\frac{1}{3}$ tetrahedron is ÿ 2a2 + 2b2 + 2c2. Consider the edge e10: $|e10|^2 = |e1m|^2 + |em0|^2$. l0 The length e1m is expressed from the formula for the median of a triangle: |e1m| = ÿ 2b2 + 2c2 ÿ $= \frac{1}{3}$ a2. Thus, we get

$$|e10|^2 \quad \frac{22b + 2c^2 \text{ ÿ } a2}{9} \; 2a2 + \frac{2b + \quad ^2 + 2c^2}{9} =$$

$$= \frac{a2 + 4b^{2\,2 + 4c}}{9} \qquad \min\{a2 , b2 , c2\}.$$

If the sides of the triangle ÿ123 were at least d, then |e10| will be at least d. Similar estimates are true for e20 and e30.

For the candidate tetrahedron with vertex v4, we check its intersection with the local front. If all candidates turned out to be unsuitable, then a complete enumeration of the entire front is carried out. If no suitable tetrahedron was found in this case either, then the face f123 is labeled

as already considered, and the enumeration starts from the beginning for the next face. Unlike

Algorithm 10, the section of Algorithm 15 with step numbers 4–20 can be executed several times to construct one tetrahedron. Each time this section is executed, the corresponding face is either removed from the front or marked. At the end of the algorithm, all such faces become faces of the finite mesh ÿh. Therefore, the number of executions of this section of the algorithm is limited from above not by the number of tetrahedra in the final mesh, but by the number of faces.

Let us consider in more detail the rest of the differences between the advancing front algorithm in three dimensions and its analogue in two dimensions from § 3.3.

Let us construct an algorithm for the intersection of a tetrahedron with a triangle similar to Algorithm 11. As in Section 3.3.2, we will perform checks based on the sign of the determinant of the 3 × 3 matrix.

In general, two convex objects do not intersect if and only if there is a plane separating them. Under the conditions of Algorithm 15, a tetrahedron and a triangle can also have one common vertex or one common edge, or the triangle can be one of the faces of the tetrahedron. Let's single out from the problem of checking the intersection a separate subtask of checking the intersection of a

triangle with a segment. Vp. 3.4.3 we have already considered the analog of this problem for a surface. Now consider the general case: we will look for a plane that separates the triangle and the segment, taking into account the previous remark about common vertices. The enumeration of all possible planes is presented in Algorithm 16. Checking the intersection of a triangle with a segment is used in Algorithm 17 for checking the intersection of a tetrahedron with a triangle. We first check to see if there are intersections between the

tetrahedron's boundary and the triangle's boundary, and then we check for special cases, such as a triangle lying entirely inside the tetrahedron. This order of checks is the most efficient in practice, since in most cases at least one of the faces of the tetrahedron intersects the edge of the triangle, less often the triangle intersects at least one of the edges of the tetrahedron, and even more rarely the triangle lies inside the tetrahedron. The proposed combination of Algorithms 16 and 17 uses the function d(v1, v2, v3, v4) to check the intersection, which calculates the sign of the expression Vÿ1234 , which is similar to the function d(v1, v2, v3) defined in Algorithm 12. Let the function d(v1 , v2, v3, v4) it is known that for d(v1, v2, v3, v4) = 0 we have d(v1, v2,

v3, v4) = sgn(Vÿ1234). Then the proposed algorithms exclude the occurrence of errors of the second type (see § 3.3), i.e., the actually intersecting tetrahedron and triangle will not be erroneously perceived as non-intersecting. This guarantees closedness and the absence of self-intersections as the front advances.

Algorithm 16. Checking the intersection of a triangle with a segment in three-dimensional space

1: Find the common vertices of the triangle ÿ123 and the segment e45 2: **if** there are no common vertices **then** Determine
3: the position of the vertices v4 and v5 relative to the plane ÿ123 **if** v4 and v5 lie in the same
4: half space **then** There are no intersections **else if** v4 and v5 lie in
5: different half spaces
6: **then**
7: Calculate k1 = d(v4, v5, v2, v3), k2 = d(v4, v5, v3, v1), k3 = d(v4, v5, v1, v2) If among k1, k2, k3 there
8: are also 1 and ÿ1, then there is no intersection **else if** v4 and v5 lie in the plane
9: ÿ123 **then** Check that one of the lines given by the segments
10: v4v5, v1v2, v2v3 or v3v1 separates the triangle and the segment into different half planes If such a line is found, then there is no intersection **end if** 12: 13: **end if** 14: **if** one common
eleven: vertex **then** If the second vertex of the segment does not lie in the
plane of the
triangle, then there is no intersection
15: Otherwise, check that one of the lines defined by the segments v1v2, v2v3, v3v1 separates the triangle and the
16: segment different half-planes If such a line is found, then there is no intersection 17: 18: **end if** 19: **if** two common vertices **then**
No intersection 20:
21: **end if**
22: In other cases, we assume that the triangle and the segment intersect
repent

Let us analyze Algorithm 15 and show that the number of operations is finite. The proposed algorithm contains three explicit nested loops and one implicit one due to restarting the enumeration at step 20. The enumeration loop at step 12 is always finite due to the finiteness of the set ÿkR. The loop at step 8 does at most two iterations. In the three-dimensional case, such front configurations are possible for which there is no suitable

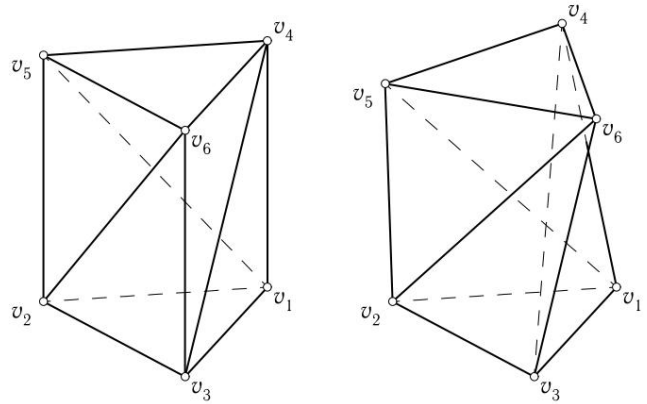
tetrahedron with a fourth vertex from the set of front candidate vertices. Examples of such a front configuration, for which further advancement is impossible, are shown in Figs. 3.16: any tetrahedron with vertices from the set {v1, v2, v3, v4, v5, v6} intersects one of the edges {e26, e34, e15}. Precisely for this reason, at step 20 of algorithm 15, the faces that do not have

Algorithm 17. Checking the intersection of a tetrahedron with a triangle

1: For each face of the tetrahedron ÿ1234 and triangle edge f567
check their intersection 2: For f567
and each edge ÿ1234 check their intersection 3: Find the common vertices of the
tetrahedron ÿ1234 and the triangle f567 4: **if** there are no common vertices **then** 5: If
the three points v5, v6, v7 lie inside
ÿ1234, that is, the intersection
nie
6: **end if** 7:
if one common vertex **then** 8: If the two
remaining vertices f567 lie inside ÿ1234, i.e. intersection 9: **end if** 10: **if** two common vertices
then 11: If the
remaining
vertex f567 lies inside ÿ1234, i.e.
intersection 12:
end if 13:
Otherwise, we assume that there is no intersection

suitable tetrahedra are marked and skipped during further meshing. Each face is labeled at most once, and the number of faces in Fk is finite, so the implicit loop at step 20

finite.



Rice. 3.16. Examples of front configurations for which no further advance is possible: Schonhardt prisms

Let us show that, with a certain choice of parameters in Algorithm 15, it is possible to limit the maximum possible number of constructed tetrahedra. The arguments in § 3.3.3 carry over completely to the three-dimensional case. We can limit the distance between grid nodes from below: ÿk ÿ > 0 for any k. A similar estimate (3.3.3) is also true

the maximum number of nodes that can be placed in a domain P that has volume V (P), surface area S(P), sum of edge lengths p(P), and consists of K(P) connectivity components: 6V (P)

$$N_k \leq \frac{6V(P)}{\sqrt{2}} + \frac{3S(P)}{\sqrt{3}} + \frac{3P(P)}{2\pi} + K(P). \quad (3.6.1)$$

From (2.2.9) it follows that the number of tetrahedra and, consequently, the number of iterations of the outer loop in algorithm 15 is limited.

The analysis of the complexity of the advancing front algorithm from Section 3.3.4 is also applicable in the three-dimensional case. The upper estimate of the number of operations in the worst case will be proportional to N_f^3 where N_f is the number of faces in the final mesh, which is limited according to (2.2.9) and the estimate of the number of nodes (3.6.1). The average estimate of the number of operations is proportional to $N_f \log N_f$. This estimate of the average speed of work will be experimentally confirmed in Section 3.7.6.

We have shown that Algorithm 15 performs a finite number of operations. However, it does not guarantee the construction of a conformal tetrahedral mesh for the entire domain P. In practice, the advanced front algorithm splits more than 90% of the volume of the domain, and Pout, the unbroken part of the domain, consists of a certain number of isolated gaps—polyhedra with a small number of faces. The second method, which will be discussed in the next section, is used to tetrahedrize the Pout region.

§ 3.7. Reliable algorithm for constructing a tetrahedral mesh

In this section, we present a method for constructing a tetrahedral mesh consistent with a given front based on the Delaunay tetrahedralization. The general idea of this method was proposed by George et al. in [55]. We will consider a simplified version of this method applicable to splitting the gaps left by the advancing front algorithm [12]. This method is suitable for arbitrary closed fronts, but it does not allow one to control either the size or the quality of the constructed tetrahedra. A third method is required to improve the mesh quality. The grid construction is divided into three stages. At the first stage, a grid is constructed for the convex hull of the set of points of a given front. At the second stage,

the mesh is refined and the area geometry is restored. At the third stage, the consistency of the grid with the given front is restored.

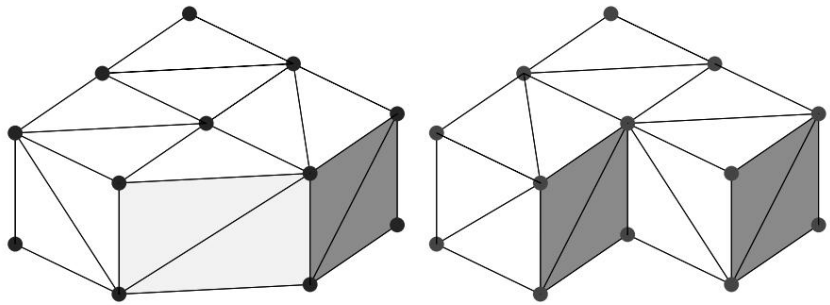
3.7.1. Delaunay tetrahedrization

Recall that *the Delaunay tetrahedra* for a finite set of points V = {v1, ..., vn} is such a conformal partition

the convex hull of a set of points into tetrahedra such that for any tetrahedron inside the sphere circumscribed around it there are no other points from V. There are many methods for constructing the Delaunay tetrahedronization; we will not dwell on the details, referring the reader to [18]. We only note that the simplest iterative method in complexity is proportional to n2 in the worst case, and proportional to n3/2 on average.

3.7.2. Region Geometry Restoration

At this stage, we have a tetrahedral mesh $\ddot{y}_h$ with a set of tetrahedra T and a set of nodes V, and a given front F with vertices V. An example of a mesh and a front is shown in Fig. 3a. 3.17. Note that they represent different areas.



A

b Fig. 3.17. Two meshes on one set of vertices: *a* — tetrahedral mesh for the convex hull of points V, *b* — front F for a nonconvex gap

We refine the mesh $\ddot{y}_h$ by adding new nodes on the edges and faces so that the edges and faces of the front F are represented in the new mesh either as a whole or as their partitions. Details are given in Algorithm 18 using recursive procedures.

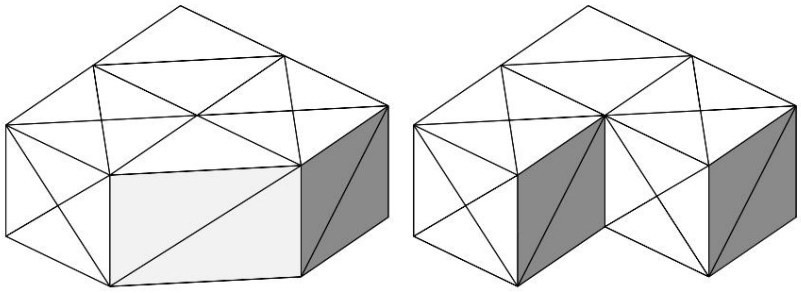
The first pass checks the intersections of the edges of the front F with the faces T . For each edge from F, the tetrahedra from T intersecting it are partitioned in such a way that the edge is represented in the grid, either as a whole or as a partition. The second pass is similar; intersections of edges of tetrahedra from T with faces F are checked. For each face from F, the tetrahedra from T intersecting it are partitioned in such a way that the face is represented in the mesh either as a whole or as a partition. Each time we split tetrahedra from the set T, we create an intersection point lying on F. In this case, only elements from T

are split. At the end, F will remain unchanged, and the new set of tetrahedra T will completely cover the faces of F with their faces lying outside the boundaries of the polyhedron P, we will restore the geometry of the area. On fig. 3.18 shows a refined mesh with a set of tetrahedra T before and after the removal of the tetrahedra.

Removing tetrahedra from T ,

Algorithm 18. Restoring the region geometry 1: **procedure**

```
PROCESS_EDGE(v1, v2) 2: Find a tetrahedron with
vertex v1 intersecting the edge e(v1, v2) if tetrahedron is found then
3:
4:          Construct an intersection point v on a face of a tetrahedron Split the
5:          corresponding tetrahedra into T PROCESS_EDGE(v, v2)
            6: 7: end if 8: end procedure 9:
procedure
PROCESS_EDGE(v1,
v2) for all faces f ÿ F do if face f intersects edge e(v1,
10:     v2) then Construct intersection
eleven:     point v Split corresponding tetrahedra into T PROCESS _
12:             FACES(v1, v) PROCESS _ FACES(v, v2)
13:             Exit procedure end if end for 19: end procedure 20: for
14:             all edges e (v1, v2) from F do 21:
15:             PROCESS _ EDGE(v1, v2) 22: end
16:             for 23: for all edges e(v1,
17:         v2) from
18:     T do 24:
PROCESS _ FACES(v1,
v2) 25: end for 26: Remove tetrahedra outside
of F
```



A

b Fig. 3.18. Restoring the geometry of the area: *a* - refined mesh, *b* - mesh with the correct geometry of the area

The resulting mesh ÿh will be a conformal mesh for the polyhedral domain P, but it will not be compatible with its boundary F. The boundary of ÿh will contain new intersection points, which we have just

built (cf. Fig. 3.17, *b* and Fig. 3.18, b). The algorithm for removing these points from the boundary will be presented in the next paragraph.

3.7.3. Restoring a trace of a grid on a boundary

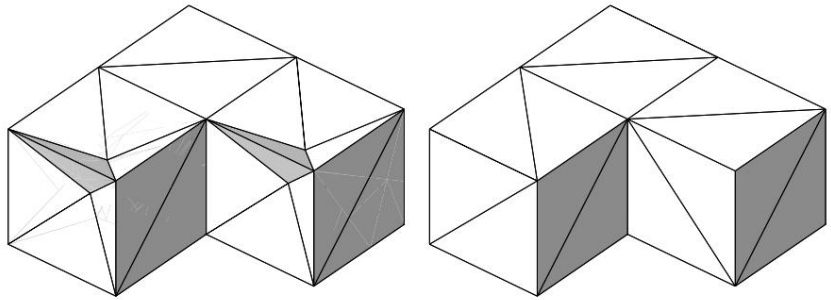
At the final stage, extra nodes are removed from the grid boundary. This is achieved by shifting them into the region and filling the resulting “dents” with conformal tetrahedral networks.

All extra points are divided into points lying on the edges of F, and into points lying on the faces of F. The elimination of points in both cases occurs in the same way. Consider, for definiteness, the second case. Let v be a node on a face in F that must be removed from the surface. Assume for simplicity that the

front F is simple. With minor additions, the proposed algorithm is also applicable in the general case. Consider the set ÿ(v) of tetrahedra from T with common vertex v. The boundary ÿv = ÿÿ(v) of this set consists of triangles of two types: those lying on the boundary of P and those lying inside P. When the vertex v is shifted inside P,

“dents” are formed on the boundary of the domain. Each of the "dents" is a pyramid with a vertex v and a polygonal base. It can be conformally divided into tetrahedra; for this, polygonal bases are divided by diagonals into triangles. It can be shown that inside ÿ(v) there is always a non-empty open set of points where the node v can be shifted while maintaining the nondegeneracy of the tetrahedra of the mesh T. 3.19 shows the mesh ÿh with four visible "dents" and with closed "dents".

.



A

b Fig. 3.19. Restoration of the trace of the grid on the boundary: *a* - grid with formed "dents", *b* - grid with the correct trace on the boundary

The first case, when the node lies on the edge F, is treated similarly; the only difference is that instead of a pyramid, a “dent” will be the union of two cones with a vertex v and a polygonal base

niami.

The complete algorithm of actions at this stage is presented in Algorithm 19. First, nodes on the edges of F are considered, and then on the faces of F.

Algorithm 19. Restoring the trace of the mesh on the boundary 1: **for all**

nodes v on the edges F and faces F **do**
2: Construct the set of neighboring tetrahedra ÿ(v)
3: From ÿv = ÿÿ(v) select the faces lying on F; let ÿ = ÿv ÿ F Repartition ÿ Add to the set = T the
4: tetrahedra formed v into triangles, excluding the node v
5: by the triangle
nicknames from ÿ v and node v
6: Move v inside ÿ(v), restoring the non-degeneracy of these tetrahedra 7: **end for**

The proposed method guarantees the construction of a grid that is consistent with a given closed front, provided that all calculations are carried out accurately. The quality of the constructed tetrahedra is in no way limited from below. In practice, due to the accumulation of computational errors, the method can generate degenerate or inverted tetrahedra. For this reason, as a final step, it is necessary to use algorithms that can significantly improve the quality of the mesh and get rid of degenerate elements (see, for example, Table 3.5).

3.7.4. Finiteness of the algorithm

Let us briefly analyze the finiteness of the number of operations in the proposed method without going into computational complexity estimates.

At the first stage, we construct the Delaunay tetrahedrization ÿh. The number of nodes in it coincides with the number of vertices in the front F. Given an estimate for the number of nodes in the mesh ÿh, using (2.2.8)–(2.2.9), we obtain estimates for the number of edges, faces, and tetrahedra.

Algorithm 18 uses three basic operations at the second stage to reconstruct the geometry. In line 21, for each edge from F of new nodes in ÿh, at most the number of tetrahedra in T at the time before line 21 is executed is added. The number of edges in F is finite, and therefore the number of new nodes is limited.

In line 24, for each edge of tetrahedra from T new nodes in ÿh, no more than the number of faces in F are added. Note that new edges, when added to ÿh, will lie on F, and therefore will not themselves generate new vertices.

The remaining operations: removing outer tetrahedra in line 24 of Algorithm 18 and shifting extra points from the boundary of the region in Algorithm 19, do not add new vertices to ÿh and, therefore, will work in finite time.

In practice, the proposed algorithm based on Delaunay tetrahedrization is used only for localized gaps with a small number of vertices and faces. The number of tetrahedra constructed with its help is small, and its operation time makes an insignificant contribution to the total operation time of the entire tetrahedral mesh construction chain. Vp. 3.7.6 experiments will be carried out to estimate the distribution of running time between the advancing edge algorithm and the proposed algorithm.

3.7.5. Improving the quality of the resulting mesh

A combination of three algorithms is used to construct a tetrahedral mesh. The first advanced front algorithm is used to build the majority of the mesh with the ability to control the mesh spacing and the quality of the tetrahedra. The second algorithm, based on the Delaunay tetrahedralization, is needed to partition the rest of the region. The third and final algorithm is used to improve the quality

grid properties.

Note that in order to achieve a good mesh quality using only the first two algorithms, as a rule, a good surface mesh, i.e., the initial front, is required. A regular initial front, a suitably chosen function responsible for the desired size of the tetrahedra, or a correctly chosen automatic coarsening parameter are the key factors for obtaining a high quality mesh. Nevertheless, within the framework of a fully automatic and reliable technology for constructing unstructured tetrahedral meshes, an additional improvement in the quality of the mesh using a third algorithm is required. In this section, we will not discuss any particular algorithm for improving the quality of tetrahedral meshes. We only mention the main ideas that can be used. The simplest methods use the shift of grid nodes without changing its topological structure. More sophisticated algorithms use local

modifications of the mesh topology, such as splitting an edge with a new node and shrinking the edge into a single node, in addition to shifting the nodes. These methods, as a rule, make it possible to significantly improve the mesh quality. They

will be described in Chap. 5. Their use is recommended as the final step in the chain of constructing a tetrahedral mesh. Vp. 3.7.6 experiments will be given to measure the quality of the mesh at different stages of its construction.

3.7.6. Experimental results

In this paragraph, several examples of the operation of the above algorithms will be presented. We will be primarily interested in the quality of the mesh Q(ÿh) and the distribution of the quality of tetrahedra over the entire mesh. The mesh quality is calculated by formulas (2.1.7) and (2.1.8).

To better display the results, we will use a logarithmic scale. The experiments were carried out on different

computers under different conditions. However, test runs within the same experiment were carried out under the same conditions.

First, we will experimentally measure the speed of the advanced edge algorithm. Let's take a unit cube as an area. On the surface of the cube, using the advanced front algorithm described in Section 3.4.3, we construct a quasi-uniform grid with step h. The resulting surface triangulation is used as the initial front for the 3D advanced front algorithm in constructing a quasi uniform mesh with step h. Decreasing the grid step h, we will monitor the number of tetrahedra in the grid Nt and the construction time ee t. The results of the experiments are presented in

table. 3.4.

T a b l e 3.4

The speed of the advancing edge algorithm in three dimensions

h	Nt	t, s Nt/t, sÿ1 t/(Nt log Nt), µs	
0.1	3544 0.48 7383		16.6
0.05	27988 4.81	5819	16.8
0.025	204734 43.35 4723		17.3
0.0125	1613980 406.75 3968		17.6

It can be seen from the calculation results that the operating time is proportional to the value of Nt log Nt, with a relatively small coefficient. The table also includes a column for meshing speed (tetrahedra per second).

We now consider an example of successive application of the advancing front algorithm and the stable algorithm based on the Delaunay tetrahedralization. The first algorithm is used to build most of the grid, the second one is used to split the remaining gaps.

Consider the initial surface triangulation of the 3D scanned dragon model shown in Fig. 3.20 a. It consists of 54,296 nodes and 108,582 triangles. All grid nodes lie in plane slices parallel to the xy plane. In each cut, the vertices are connected by segments, forming the contour of the section of the model by a plane. The contours of adjacent slices are connected by triangles and form a conformal surface triangular grid. Note that this initial mesh contains a sufficiently large number of triangles

with bad quality.

A

b Fig. 3.20. Operation of the advanced front algorithm on the dragon model: *a* - initial front, *b* - final front

At the first stage, we will apply the advancing front method with an automatic increase in the grid step when moving inside the area. The parameter ÿ, which is responsible for the rate of increase in the grid step, is chosen to be 1.3. The advanced front algorithm built a grid in a part of the area that occupies 99.67% of the volume of the entire area. In total, 250,461 tetrahedra were built, and 1718 triangles remained in the final front, which are shown in Fig. 3.20 b. The distribution of the quality of the tetrahedra of the resulting mesh is presented in the first row of Table 1. 3.5. The columns of the table show the number of tetrahedra whose quality exceeds the corresponding value and which were not included in the previous

columns. The worst mesh quality is Q(ÿh) = 3.698 10ÿ7, which is due to the poor quality of the triangles in the initial front. The resulting front is automatically divided into

disconnected gaps, and a stable method is run for each of them. Gaps are very badly shaped, and so the stable method produces very bad elements. The quality distribution of tetrahedra in the final mesh is given in the second row of Table 1. 3.5. The worst mesh quality is now Q(ÿh) = 1.232 10ÿ14, i.e. the worst tetrahedron is

T a b l e 3.5

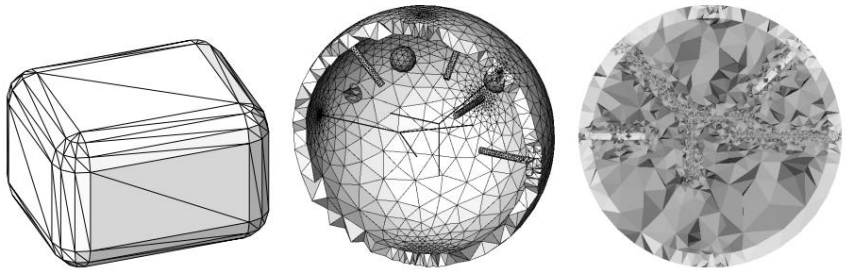
Distribution of the quality of tetrahedra at different stages of meshing for the dragon model: ÿ is the advancing front algorithm; D is a stable algorithm based on Delaunay tetrahedrization; PS - improving the grid by rebuilding it while maintaining a given trace on the boundary

	Q(ÿh)	Nt	10ÿ1	10ÿ2	10ÿ3	10ÿ4	10ÿ5	10ÿ6	10ÿ7
F	3.7 10ÿ7	250 461 248	359	1923	166	11	1.2	1	0 1
1.9	10ÿ14	254 764 249	674	3428	1020	340	138	71	93 F+D F+D+PS
10ÿ7	267 475	258 449 8	269	718	30			6	0 3

this is a sliver. The resulting tetrahedral mesh contains 71,241 nodes, 108,582 boundary faces that completely coincide with the given initial front, and 254,764 tetrahedra.

To improve the quality of the resulting mesh, we used the mesh rebuilding (GR) methods described in Chap. 5, while the boundary faces of the grid were fixed. The quality distribution of tetrahedra in the new mesh is shown in the third row of Table. 3.5. Tetrahedra with poor quality added to the mesh at the second stage were removed. The quality of the tetrahedral mesh in this example is limited by the poor quality of the initial surface triangulation. The new mesh has 75,665 nodes, 108,582 boundary faces, and 267,475 tetrahedra. Note that with a preliminary improvement in the quality of the initial surface mesh, a tetrahedral mesh with better quality can be obtained [40]. Similar experiments were carried out for a large number of different areas, we will give here only a few illustrative examples (see

Fig. 3.21). The first example is the rbox1 model, a CAD-exported triangulation of a surface for a cuboid with rounded edges. The surface mesh consists of 104 nodes and 204 triangles. The quality of the triangles in it is very poor, there are sharp differences in the size of the triangles. built



A *b* *V*

Rice. 3.21. Examples of areas used: *a* - rbox1 model obtained by export from CAD; *b* - section of the initial front of the Lymph model; *c* - section of the constructed mesh for the Lymph model

for the rbox1 model, the tetrahedral mesh contains 138 nodes and 436 tetrahedra, the minimum quality of tetrahedra is
Q(ÿh) = 4.430 10 ÿ13. The second example is the drag3 model, the dragon model discussed in this section. The third example is the Lymph model, a schematic model of a lymph node created in the Open CASCADE CAD system. The lymph node model consists of several components. The outer part consists of a marginal sinus, a thick spherical shell, six cylindrical trabecular sinuses, and two Y-shaped conduits. The diameter of the conduits is 1000 times smaller than the diameter of the lymph node. The model includes 4 spherical follicles. The rest of the inside of the spherical membrane of the lymph node represents the cortical and paracortical regions. The geometric model of Lymph consists of 50 nodes, 62 curved edges and 30 curved faces. The constructed triangular surface mesh consisted of 24,720 vertices and 95,732 triangles, while the volumetric tetrahedral mesh consisted of 103,891 vertices and 619,691 tetrahedra. The worst quality of the tetrahedra was Q(ÿh) = 3.45 · 10 ÿ12. Vtab. 3.6 for each model shows the distribution of the volume of the area for which the grids are constructed using the advancing front method and the stable method. For each method, the number of

constructed tetrahedra and the time required for its operation are also given.

T a b l e 3.6

Distribution of region volume, number of tetrahedra, and running time between the advancing front algorithm and the stable algorithm based on Delaunay tetrahedration

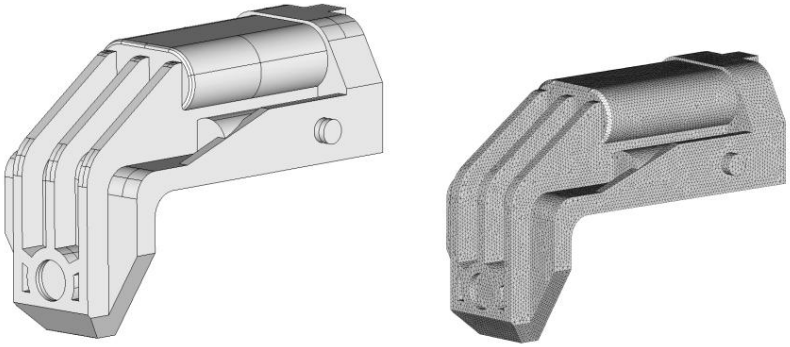
Model	Promoted Front Steady Method						
	%Nt		t, s	%	Nt	t, s	
rbox1	97.25	265	0.18	2.75	171	0.03	
drag3	99.67	250 461	101.70	0.33	4303	0.76	
Lymph	99.96	617 691	168.93	0.04	2000	0.34	

We note that the stable method is used only in a small part of the entire region and therefore takes a small part of the time compared to the advancing front algorithm.

The tetrahedral meshes constructed for the rbox1 and Lymph models are of poor quality; this is an expected result, since the algorithms presented in this chapter are primarily aimed at constructing topologically correct meshes. The mesh quality can be improved using the meshing methods described in Chap. 5: for example, the quality of the new mesh for the rbox1 model increases to Q(ÿh) = 7.818 10 ÿ8, and for the Lymph model -

up to Q(ÿh) = 7.78 10ÿ2. With preliminary improvement of the surface mesh (see § 3.6) and final smoothing of the spatial mesh (see Chap. 5), the quality of the final mesh for the rbox1 model increases to Q(ÿh) = 2.18 · 10ÿ1.

In conclusion, we will demonstrate the joint work of methods for constructing surface triangular and volumetric tetrahedral meshes. As an example, consider model 62_shaver1 from the Open CASCADE website [19]. The geometric model consists of 266 nodes, 403 curvilinear edges and 153 curvilinear faces (see Fig. 3.22, a). Using the method of advancing front on the surface, described in § 3.4, a quasi-uniform triangular mesh with 33,176 nodes and 66,348 triangles was constructed (see Fig. 3.22, b) .



A

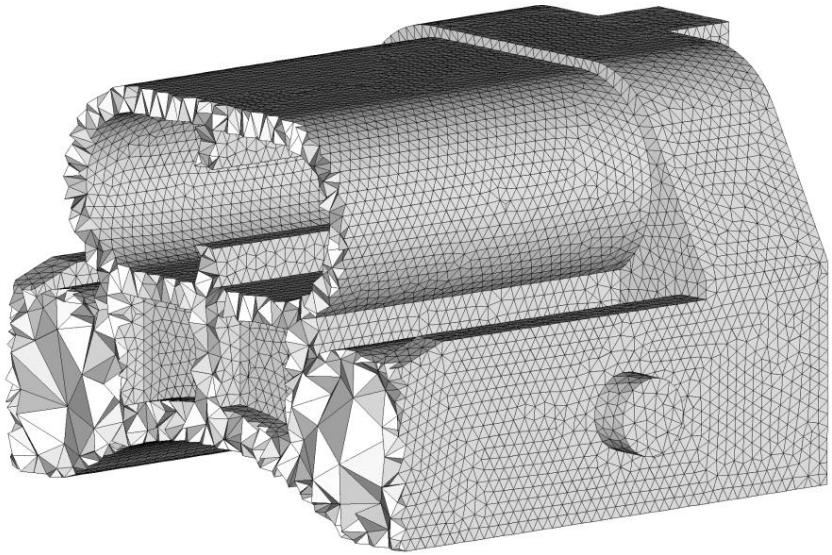
b Fig. 3.22. Model specified in CAD: *a* - BREP model, *b* - surface quasi-uniform triangular mesh

A combination of three methods was used to construct the tetrahedral mesh: the advancing front algorithm, the stable algorithm based on Delaunay tetrahedralization, and the mesh improvement method described in Chap. 5. The first method was used to construct a mesh with 158,548 tetrahedra for a part of the region that occupies 99.81% of the total volume of the model. The distribution of the quality of the tetrahedra of the resulting mesh is presented in the first row of Table 1. 3.7. There are 838 triangular faces left in the front; they were passed to the input of the second method, which constructed another 1280 tetrahedra. The detailed distribution of the quality of tetrahedra in the final mesh is given in the second row of Table 1. 3.7. The minimum quality of the resulting grid was Q(ÿh) = 1.117 10ÿ6. After the mesh was corrected using the third method, the quality of the mesh improved significantly, and the minimum quality of one tetrahedron amounted to Q(ÿh) = 1.102 10ÿ2. The resulting mesh contains 48,999 nodes, 66,348 boundary faces that completely match the faces generated by the surface mesh generator, and 176,764 tetrahedra.

T a b l e 3.7

Distribution of the quality of tetrahedra at different stages of meshing for the 62_shaver1 model: ÿ is the advancing front algorithm; D is a stable algorithm based on Delaunay tetrahedrization; PS - improving the grid by rebuilding it while maintaining a given trace on the boundary

	Q(ÿh)	Nt	10ÿ1	10ÿ2	10ÿ3	10ÿ4	10ÿ5	10ÿ6	
F	7.03 10ÿ4	158 548	157 091	1430	1.12	26	1	0	0
P+D	10ÿ6	159 828	157 693	1821	244	36	28		6
P+D+PS	1.10	10ÿ2	176 764	177 690		74	0	0	0	0



Rice. 3.23. Section of a tetrahedral mesh for a region defined in CAD

The quality distribution of tetrahedra in the new mesh is presented in the third row of Table 1. 3.7. On fig. 3.23 shows a cut of a tetrahedral grids.

The total running time was 5 min 53 s. The time distribution by stages is as follows: surface mesh construction — 3 min 23 s, volume mesh building — 1 min 37 s, mesh improvement — 8 s. The remaining time was spent on checking the correctness of the resulting grid and various auxiliary operations.

Chapter 4

MULTILEVEL HIERARCHICAL GRINDING AND ROUGHING THE MESH

This chapter will describe methods for constructing dynamic 2D and 3D meshes based on multilevel hierarchical refinement and coarsening of triangles and tetrahedra. The presented set of algorithms makes it possible to effectively multiple local remeshing when simulating various dynamic processes.

§ 4.1. Principles of multilevel grid construction

In practice, there are often problems related to the study of non-stationary processes with moving singularities. Such processes can be the movement of the front of the contaminated zone, the propagation of a crack, and the formation of a prefracture zone. Successful solutions to such problems require grids that can be quickly rebuilt to reflect ongoing changes.

One of the ways is to construct in the domain under consideration a sequence of simplicial (triangular or tetrahedral) grids ÿn n = 0, 1, ... satisfying the following properties: — any grid ÿn is conformal; — the initial grid ÿ0 can be arbitrary; — each grid for n 1 is obtained from the previous one by means of operations of multilevel refinement or coarsening of some grid elements. These operations generate a sequence of hierarchical grids. The set of triangles or tetrahedra subjected to refinement or coarsening is determined by the specifics of the problem, the wishes of the user, and the requirement to maintain the conformity of the resulting meshes.

Such grids are called *hierarchical.* Sections 4.2, 4.3, and 4.4 will be devoted to the description and detailed analysis of algorithms for modifying hierarchical triangulations and tetrahedrizations. In § 4.5 we consider an algorithm for constructing grids that trace the features of dynamic processes. It is important to note that the operations of refinement and coarsening of the hierarchy cal grids obey the following set of principles.

1) The mesh refinement operation is applicable to *an arbitrary* initial Noah conformal triangulation or tetrahedralization of the region.

2) The mesh coarsening operation is applicable only to those mesh elements that were previously refined. Thus, the grid element ÿn cannot be n > 0 coinciding with some element ÿ0 is coarsened.

3) The described algorithms make it possible to perform local refinements and coarsenings, which makes it possible to construct grids that are condensed in a subdomain.

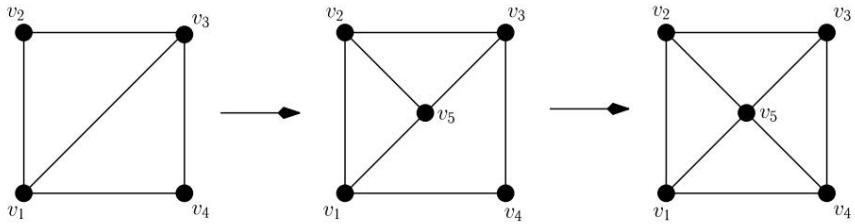
4) The resulting grids have the property that the minimum angle of any triangle cannot be less than half of the minimum angle of the triangles in the initial grid. Similarly, the minimum dihedral angle of any tetrahedron cannot be less than half the minimum of the dihedral angles of the tetrahedra of the initial mesh. The same is true for the flat corners of tetrahedra.

5) Grinding and coarsening operations are performed quickly enough; the number of actions performed grows linearly with the number of triangles or tetrahedra in the mesh to which the operations in question are applied.

Principle 4) is especially important for the numerical solution of partial differential equations by the finite element method.

§ 4.2. Bisection Method for Refining Triangulations

Triangulation refinement is the process of successively repeatedly applying the operation of splitting a certain triangle into two triangles. We will regard such a partitioning operation as a basic one and call it *a bisection* of a triangle. A bisection of one triangle often causes a violation of the conformity of the mesh on an adjacent element. Because of this, the number of mesh elements increases, which must be refined to build a mesh. As can be seen from fig. 4.1, under the bisection of the triangle ÿ123, the triangle ÿ134 breaks the conformality on the edge e13. Splitting the triangle ÿ134 helps to keep the grid conformity.



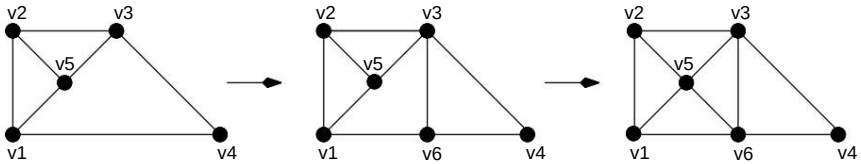
Rice. 4.1. Mesh Conformity Restoration

Before describing the triangular mesh refinement algorithm itself, let us dwell in detail on the bisection of a particular mesh

element. There are various methods for splitting a triangle. We will focus on two of them.

The first method was proposed by Rivara in [78, 79] and is known as *the largest edge partitioning method.* The method consists in the fact that if a triangle ÿ is included in the set of grid elements that need to be refined, or if one of its edges does not satisfy the mesh conformity property after the bisection of other triangles, then, first, this grid element is split by a segment connecting the middle of the largest edge with opposite top. If, after making this bisection, the grid conformality property is still violated at least on one edge of the original triangle, then the midpoint of this edge is connected to the midpoint of the largest edge of the triangle ÿ. The principle of operation of the partitioning method along the largest edge is illustrated in fig. 4.2. As a result of the bisection of the triangle ÿ123, the conformality property of the triangle ÿ134 on the edge e13 is violated. Triangle ÿ134 must be refined to maintain conformity. First, its partition is carried out by the edge e36. However, this does not restore the mesh's conformity

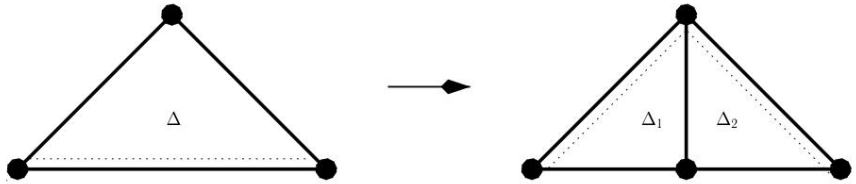
property. Only the subsequent splitting of the triangle ÿ136 by the edge e56 leads to the desired result.



Rice. 4.2. Rivara's triangle bisection algorithm

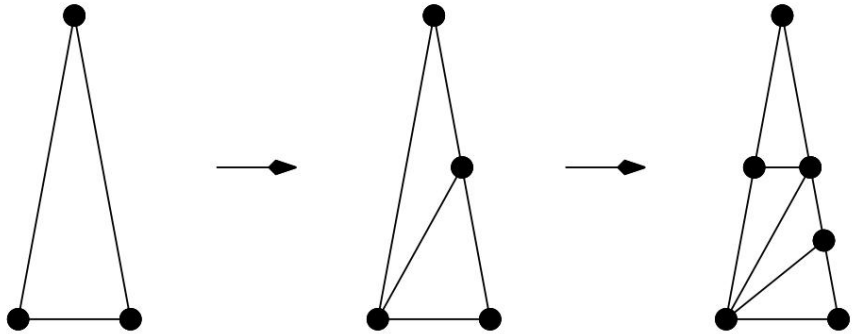
Another method, which was described in Bansh's paper [31], is called *the labeled edge partitioning method.* In our opinion, it is more convenient and easy to implement when developing a complex algorithm for constructing dynamic grids. Let us pay special attention to this method. Each triangle of the grid is associated with one of its edges, which we will call *labeled.* The choice of such an edge for the triangles of the initial grid is carried out arbitrarily, regardless of which edge of the neighboring triangle turned out to be labeled. Often take the longest edge in the triangle.

The Bansch splitting method not only splits a triangle into two, but also marks the edges of two new triangles. According to this method, each triangle ÿ can be divided only by an edge connecting the middle of the labeled edge with the opposite vertex. For two new triangles, the labeled edges are the edges that coincide with the edges of the original triangle ÿ that have not changed during its bisection. The principle of operation of the Bansh algorithm is shown in Fig. 4.3, where the dotted lines in the triangles mark the marked edges.



Rice. 4.3. Bansch's Triangle Bisection Algorithm

At first glance, it may seem that the two proposed algorithms perform the same partition. To refute this assumption, we give an example of an initial grid consisting of a single triangle and the results of the two described methods. In this case, we will bisection each of the available triangles twice: first, by splitting along the largest edge, then along the labeled edge. Since the initial mesh consists of only a single triangle, the conformality of the mesh will not be violated in the course of bisections. Rice. 4.4 demonstrates the operation of the Rivara algorithm, and fig. 4.5 - Bansch's algorithm. In this case, in the original triangle, the edge with the maximum length is selected as marked. As we can see, the resulting grids turned out to be different.



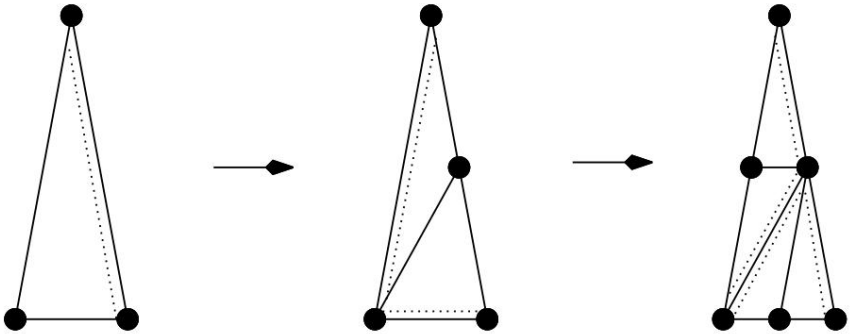
Rice. 4.4. Two steps of bisections of triangles by Rivara's method

In what follows, speaking of the bisection of a certain triangle ÿ, we will assume that it is produced by the Bansch method. We introduce the following convenient notation for the triangle bisection operation:

$$B(ÿ) = (ÿ1, ÿ2). \tag{4.2.1}$$

Let ÿh(V, T) denote a triangular grid, where V is a collection of grid nodes, each of which is determined by two coordinates, and T is a set of triangles. Let V(ÿ) be the set of global indices of the vertices of the triangle ÿ(v1, v2, v3), and r(ÿ) be the local number of the labeled edge in the triangle ÿ: 1 r(ÿ) 3.

V = {v1, v2, v3},



Rice. 4.5. Two steps of triangle bisections by the Bansch method

For simplicity, we assume that the local number of the vertex opposite the labeled edge is also equal to r(ÿ). Let us define a four of numbers

$$D(ÿ) = (V(ÿ), r(ÿ)). \qquad (4.2.2)$$

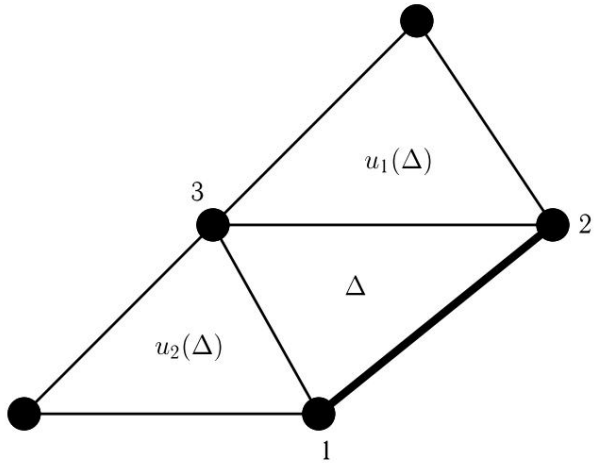
An efficient implementation of the triangulation bisection algorithm requires information about neighboring triangles for all grid elements. For each triangle ÿ we define the set

$$U(ÿ) = (u1(ÿ), u2(ÿ), u3(ÿ)),$$

where the subset ui(ÿ) contains the numbers of triangles that border ÿ along the ith edge (see Fig. 4.6). For a conformal grid, ui(ÿ) contains only one triangle. If the ith edge lies on the boundary of the domain, then we set ui(ÿ) equal to the label of the corresponding part of the boundary with a minus sign. Let U(ÿh) denote the collection U(ÿ) for all ÿ belonging to the grid ÿh. The set U(ÿh) for a conformal grid is best represented in the form described in Chap. 2 structured list TT(3,Nf), where Nf is the number of triangles in the current grid. When one triangle is split, only the data about its nearest neighbors change. Therefore, the cost of maintaining a structured list U(ÿh) does not actually affect the complexity of the entire mesh refinement process.

However, before performing the bi section triangulation procedure, U(ÿ) must be determined for each triangle. The complexity of the optimal algorithm for computing U(ÿh) is linear with respect to the number of triangles in ÿh. When constructing U(ÿh), for each node of the grid v, we will use the superelement ÿ(v) as a set of triangles with node v as one of their vertices. The calculation of U(ÿh) is presented in Algorithm 20.

To check the conformity of the grid or its violation on some grid element, it is enough for each triangle to determine the binary array C(ÿ) of three numbers (c1(ÿ), c2(ÿ), c3(ÿ)). Let



Rice. 4.6. The set U(ÿ) for the boundary triangle

Algorithm 20. Building a structured list U(ÿh) for three angulations

```
 1: For each node v put ÿ(v) = ÿ 2: loop over all triangles
ÿ ÿ ÿh 3: For all v ÿ V(ÿ) add ÿ to the set ÿ(v) 4: end
loop 5: loop over all triangles ÿ ÿ ÿh loop over all vertices vj ÿ V(ÿ) Find
vk,vm ÿ V(ÿ)
distinct from vj , vk = vm Find the set of triangles L
6:      = ÿ(vk) ÿ(vm) if L contains triangle ÿ1 = ÿ
7:          then Set uj (ÿ) = ÿ1
8:
9:
10:
eleven:     else
12:             Determine uj (ÿ) from edge label information end if 14: end loop 15: end
                loop
13:
```

the value ci(ÿ) = 1 indicates problems with conformality on the edge opposite vertex i. Let M be some given subset of T

, a expression

$$T) = Bisection\_method(ÿh(V, T),M) ÿ h(V , \qquad (4.2.3)$$

will mean that the mesh ÿ h is obtained from the triangulation ÿh by the obligatory bisection of all triangles from the set M.

When the mesh is refined, the sets of nodes and triangles are constantly changing. The grinding procedure can be written in the form of Algorithm 21.

Algorithm 21. Bisection method of triangulation ÿh(V, T)

```
1: Run Algorithm 20 to initially determine U(ÿh) 2: Put V = V, T = T 3:
while set M is not empty do

4:      loop over all triangles ÿ ÿ M
5:          Bisection B(ÿ) = (ÿ1, ÿ2) Add ÿ1 and ÿ2 to T
6:          Remove ÿ from M and T
7:
8: end loop loop
        over all triangles ÿ ÿ T 9:
10:         if array C(ÿ) contains 1 then Add ÿ to
eleven:         set M end if end loop 13: 14:
12:         end
        while
```

In this algorithm, the bisection operation B(ÿ) = (ÿ1, ÿ2) is a subroutine that, given a quadruple of numbers D(ÿ) (4.2.2), builds D(ÿ1) and D(ÿ2) and simultaneously updates auxiliary information about the grid (see Algorithm 22). Note that during the operation of this algorithm, the conformality of the grid may be violated and some triangles may border (through an edge) with two triangles, denoted as ÿ3 and ÿ4. Let us illustrate the operation of this algorithm using the example of Fig. 4.1, omitting the

stage of determining the neighbors for each grid element. Let the original mesh ÿh consist of two triangles ÿ123 and ÿ134 that have a common labeled edge e13. The set M contains only the triangle ÿ123. The presented algorithm first determines for ÿ123 the neighbors ÿ3 and ÿ4 through the edge e13. In this case, ÿ3 and ÿ4 coincide - this is the triangle ÿ134. Next, a bisection of the grid element ÿ123 is performed, which leads to the appearance of two new triangles ÿ125 and ÿ235. Since ÿ3 and ÿ4 coincide, the triangle ÿ134 on the edge e13 did not violate the grid conformality. Therefore, after splitting ÿ123, we add the point v5 to the list of grid nodes and impose on the edge e13 of the triangle ÿ134 the mesh conformality violation condition. For triangles ÿ125 and ÿ235, we determine the neighbors, and for ÿ134 we change the information about the neighbors through the edge e13. It is important to note that at the moment this triangle has two neighbors at once. Next, we add ÿ125 and ÿ235 to the set of grid triangles, and remove the grid element ÿ123 from the grid and from the set M. As a result, M becomes empty. After that, passing through all the elements of the grid

Algorithm *22*

```
1: Given U(ÿ), find ÿ3 and ÿ4 adjacent to ÿ along the edge r(ÿ) 2: if ÿ3 =
ÿ4 then 3: 4: else
        Set v as the midpoint of the side r(ÿ)

5:      Determine v as a common point of ÿ3 and ÿ4 lying on the edge r(ÿ)

6: end if
7: Determine D(ÿ1) and D(ÿ2) from D(ÿ) 8:
Determine U(ÿ1) and U(ÿ2) and adjust U(ÿh) of neighbors

ÿ9: if ÿ3 = ÿ4 then
10:     Assign 1 to the array element C(ÿ3) corresponding to the edge shared
        with ÿ. Add v to V 11: end if 12: Put
cr(ÿ1)(ÿ1)
= 1 and cr(ÿ2)(ÿ2) = 1 if cj (ÿ) = 1 on the corresponding edge ÿ
```

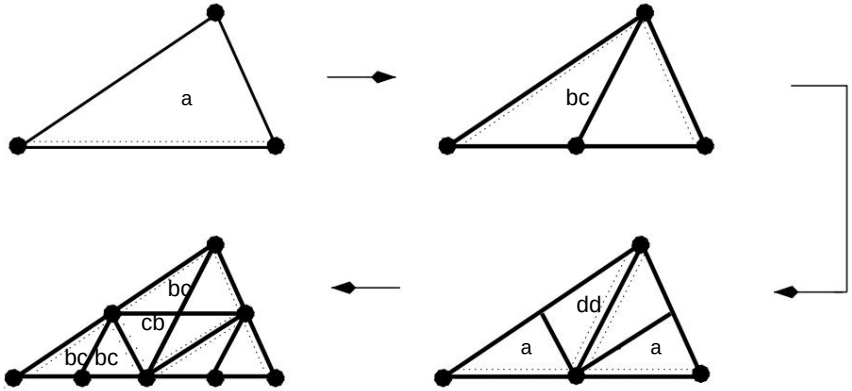
in order to detect triangles on whose edges the conformality property is violated, in M we write the triangle ÿ134.

Since the set of labeled triangles is again non-empty, the algorithm again starts bisecting triangles from M, namely ÿ134. Now the role of ÿ3 and ÿ4 is played by grid elements ÿ125 and ÿ235. Since they are different, after the bisection ÿ134 we do not need to add the point v5 to the list of grid nodes (it is already there), and there is no need to impose a conformality violation condition through the edge e13. After the bisection ÿ134, we remove it from the set M, which becomes empty. We obtain the required conformal grid.

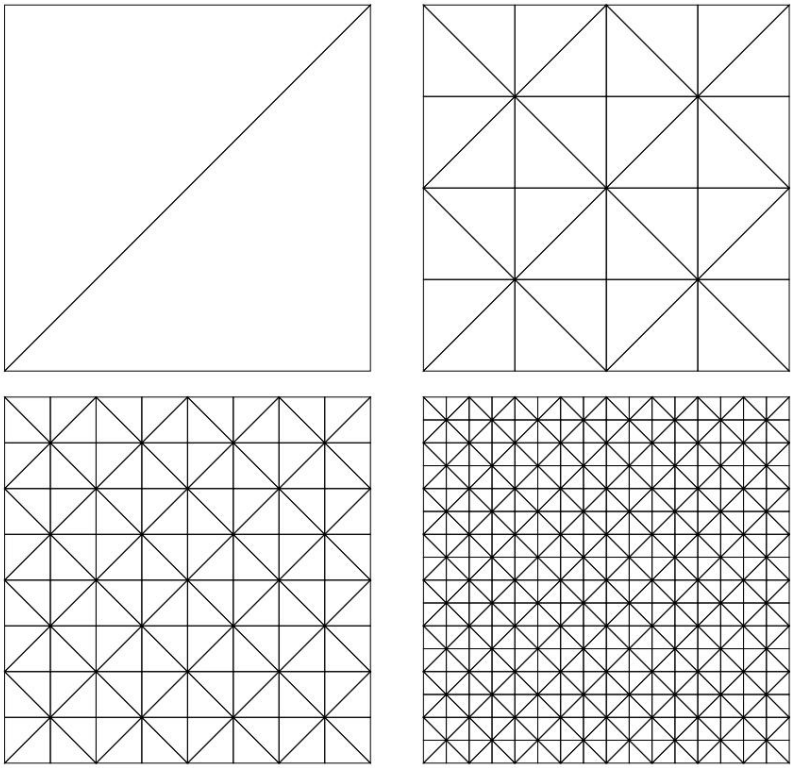
Let's make some important remarks. As can be seen from Algorithm 22, the bisection of one triangle can lead to violation of the mesh conformality only on one mesh element. Therefore, if the set M contains a small number of elements compared to the number of all triangles in the mesh to which the bisection procedure is applied, then the mesh will be refined locally. The operation of finding v, a common point of ÿ3 and ÿ4 lying on the labeled edge ÿ, can be carried out taking into account information about neighbors ÿ3 and ÿ4. If the edge being split belonged to the boundary, then during the bisection, new triangles on the edges located on the boundary must retain the value ui(ÿ) previously assigned to the split edge. This will save information about the various boundary

labels.

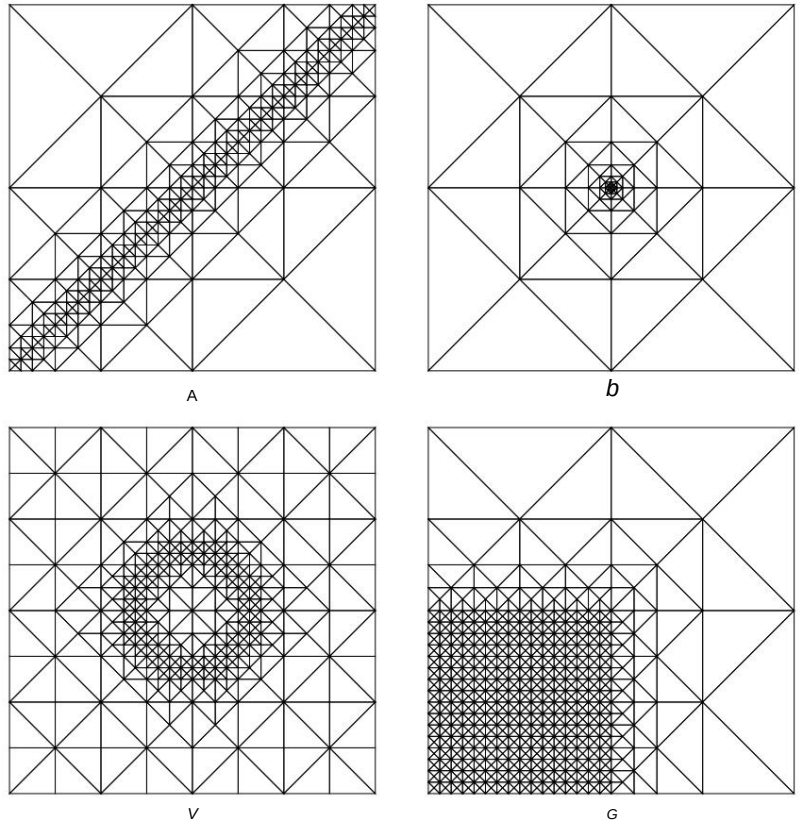
The process of *multilevel refinement of triangulation* consists in the sequential application of the bisection algorithm. This process



Rice. 4.7. Similar triangles arising from one, two and three levels of uniform triangle refinement



Rice. 4.8. Several levels of uniform meshing



Rice. 4.9. Various adaptive mesh refinements

implies the existence of a procedure for determining the set of labeled triangles. Bansh's choice

of labeled sides ensures the regularity of the triangles in the refined mesh. This follows from the following theorem [31].

Theorem 4.2.1. *With multilevel refinement of triangulation, the minimum angle of any grid element for any number of grid partitions is not less than half of the minimum angle of the triangles of the initial grid.*

Proof. According to the description of the method of splitting a triangle along a marked edge, the bisection line each time leaves the vertex that appeared during the previous refinement and connects it to the midpoint of the opposite side. Therefore, if we consider several levels of refinement of an arbitrary triangle, then we obtain a set of triangles, each of which is similar to either

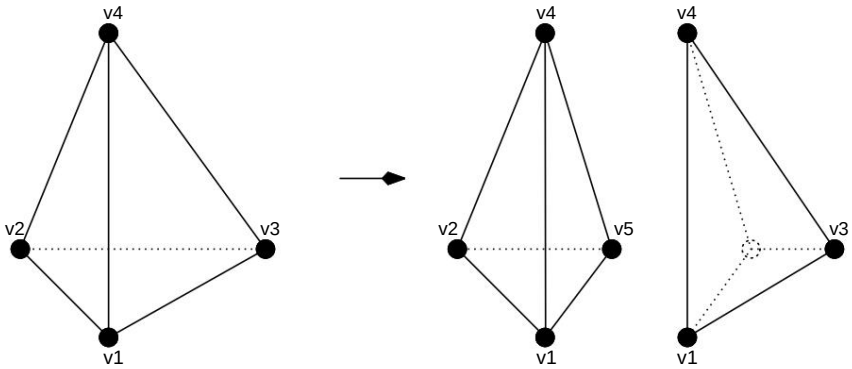
the original triangle itself, or triangles obtained from the one under consideration after one or two levels of refinement (see Fig. 4.7, similar triangles are marked with the same letters). The minimum angle during one or two refinements cannot be reduced by more than two times, and the maximum reduction is achieved when the equilateral triangle is split. Consequently, even with a greater number of refinements, a decrease in the angle of the triangle by more than two times cannot occur.

It follows from Theorem 4.2.1 that if the minimum of the angles of triangles in the initial grid is ÿ, then any angle in the resulting grid is bounded above by ÿ ÿ ÿ.

In conclusion, we present two examples of how the algorithm works. On fig. 4.8 shows the initial grid and the resulting triangulations after 4, 6 and 8 levels of uniform partitioning; rice. 4.9 illustrates different types of adaptive refinements: to a diagonal (a), to a point (b), to a circle *(c) ,* and to a subdomain (d).

§ 4.3. Bisection Method for Refining Tetrahedrals

The bisection method for refining tetrahedral structures is based on the same approaches that were described in the study of the procedure for refining a triangular mesh. The main basic operation is *the bisection* of the tetrahedron, which splits it into two tetrahedra, as shown in Fig. 4.10. However, the third dimension brings its own characteristics to the process of meshing.



Rice. 4.10. Bisection of a tetrahedron

As in the two dimensional case, we first describe in detail the method of bisecting an individual tetrahedron. In the three dimensional case, there are much more possibilities for specifying the rules for the bisection of the tetrahedron ÿ and for determining methods for further refinement of grid elements, which are

result of the bisection ÿ. Any admissible set of rules must guarantee that the minimum dihedral and planar angles in the resulting mesh, when it is repeatedly refined, decrease by no more than a factor of two compared to the corresponding minimum dihedral and planar angles of the initial mesh. We will focus on the approach proposed by Arnold et al. in [28]. The presented method is a generalization of the Bansh method of bisection of tetrahedra [31]. Let us introduce the following notation. To each tetrahedron ÿ we assign one of its edges, which we will call *the split edge* r(ÿ), and the two faces containing this edge, the *split faces.* For each of the two non-decomposable faces, we single out

one of the edges, calling it *labeled,* and denote these edges by l1(ÿ) and l2(ÿ). The split and labeled edges will be called *singular.* In contrast to the two-dimensional case, when the labeled edge of the triangle was split, here the split and labeled edges are different. The tetrahedron ÿ will be characterized *by the binary flag* s(ÿ) ÿ {0, 1} associated with the mutual arrangement of singular edges. After a bisection, this flag determines the type of further splitting. The values s(ÿ) must be determined for all elements of the initial grid. In this case, s(ÿ) can be set arbitrarily, taking into account only one restriction: if the singular edges do not lie in the same plane, then s(ÿ) = 0. Below, when formulating the bisection method, we indicate how, based on the singular edges of the tetrahedron ra ÿ determine the edges to be partitioned and labeled and the binary flags of the cells into which the tetrahedron ÿ is partitioned. The method of bisection of the tetrahedron ÿ is as follows. First, ÿ is split into two tetrahedra ÿ1 and ÿ2 by a plane that passes through the middle of the edge being split and two vertices of the tetrahedron ÿ that do not belong to this edge (see Fig. 4.10). Note that each of the tetrahedra ÿ1 and ÿ2 entirely contains one of the unbreakable faces ÿ. The labeled edge of this face is defined as the edge to be split for the corresponding new tetrahedron. As a result, for ÿ1 and ÿ2 it becomes known which of the faces are breakable. It remains to determine the labeled edges for the indecomposable faces of the tetrahedra ÿ1 and ÿ2. Let v be the midpoint of the splitting edge of the tetrahedron ÿ, which has become common for ÿ1 and ÿ2. Consider any of the unbreakable faces of the

grid element ÿ1 or ÿ2. If this face is part of a face of the tetrahedron ÿ, then we mark in it an edge opposite to v. In the case when a common face for ÿ1 or ÿ2 is considered, it is endowed with the same labeled edge according to the following rule. If s(ÿ) = 1, then the labeled edge is the one that connects the vertex of the already defined splitting edge ÿ1 or ÿ2 and the point v; otherwise, the edge opposite v will be labeled. For two new tetrahedra, s(ÿ1) = s(ÿ2) = 1 only if the subdivided and labeled edges of the grid element ÿ

lie in the same plane and the condition s(ÿ) = 0 is satisfied. Thus, the method of bisection of a particular tetrahedron is completely described. This operation, by analogy with the two dimensional case, will also be denoted by the formula B(ÿ) = (ÿ1, ÿ2).

For simplicity of working with a tetrahedral mesh, it is convenient to introduce a correspondence between the vertices, edges, and faces of a particular mesh element. Given a tetrahedron defined by four mesh nodes v1, v2, v3, and v4, we will order the six edges as follows:

e(v1, v2), e(v1, v3), e(v1, v4), e(v2, v3), e(v2, v4), e(v3, v4),

and four edges like this:

f(v1, v2, v3), f(v1, v2, v4), f(v1, v3, v4), f(v2, v3, v4). Now,

speaking about some local number of an edge or face of a tetrahedron, one can unambiguously determine their vertices.

Let ÿh(V, T) denote the tetrahedral mesh ÿh, where V is the set of mesh nodes, each of which is defined by three coordinates, and T is the set of tetrahedra. Each tetrahedron ÿ ÿ T is assigned a data set of eight numbers

D(ÿ) = (V(ÿ), r(ÿ), l1(ÿ), l2(ÿ), s(ÿ)), (4.3.1) where V(ÿ) = (v1, v2, v3,

v4) is a vector consisting of the global numbers of four grid nodes that are vertices ÿ, r(ÿ) is the local number of the split edge, l1(ÿ), l2(ÿ) are the local numbers of labeled edges, and s(ÿ) is the binary flag mutual arrangement of special edges. Note that

1 r(ÿ), l1(ÿ), l2(ÿ) 6.

The bisection process can be accelerated if information about neighboring tetrahedra is available for any grid element. To each tetrahedron ÿ we associate the set U(ÿ) = = (u1(ÿ), u2(ÿ), u3(ÿ), u4(ÿ)), where the subset ui(ÿ) contains the numbers of tetrahedra that border on ÿ along face opposite to vertex i. For a conformal mesh, ui(ÿ) contains the number of only one tetrahedron. For boundary faces, as ui(ÿ) we can take the label of the boundary with a minus sign, by analogy with how it was proposed in the two-dimensional case. Let U(ÿh) denote the collection U(ÿ) for all ÿ belonging to the grid ÿh. When constructing U(ÿh), for each node v of the grid ÿh, we will use the superelement ÿ(v) as a set of tetrahedra having the point v as one of their vertices. The procedure for determining U(ÿh) can be implemented with complexity linearly dependent on the number of tetrahedra in ÿh (see Algorithm 23).

As in the bisection of a triangular mesh, in the three-dimensional case, the mesh conformality check operation is necessary. Conformity violation is easier to track on the ribs, since the refinement of one

Algorithm 23. Construction of a structured list U(ÿh) for tetrahedralization

```
1: For each node v put ÿ(v) = ÿ 2: loop over all
tetrahedra ÿ ÿ ÿh 3: For each v ÿ V(ÿ) add ÿ
to ÿ(v) 4: end loop 5: loop over all tetrahedra ÿ ÿ ÿh 6:


      loop over all vertices vj ÿ V(ÿ) Find distinct
7:        vertices vk, vm, vn ÿ V(ÿ) other than vj Find the intersection L = ÿ(vk)
          ÿ(vm) ÿ(vn)
8:        if ÿ1 ÿ L and ÿ1 = ÿ then Put uj (ÿ) = ÿ1
9:
10:
eleven:   end if
12:       if L = {ÿ} then
13:           Determine uj (ÿ) from boundary information
14:       end if
15: end loop 16:
end loop
```

tetrahedron often entails violation of the grid conformity on several grid elements at once. For each tetrahedron, we introduce an integer array C(ÿ) of six numbers (ci(ÿ), i = 1, ..., 6) corresponding to the edges of the given grid element. If ci(ÿ) = 0, then this means a violation of the conformality on the edge i, i.e., the appearance of a new grid node v0 in the middle of the edge of the tetrahedron ÿ. This edge may belong to a whole set of grid elements W at once. To preserve conformality, all tetrahedra in W must be reduced by bisections that split the edge under consideration. Therefore, all grid elements from W need information about the node v0. The condition that the bisection of some tetrahedron and the appearance of a new node v0 lead to violation of conformality on the edge j of the tetrahedron ÿ will be given in the form cj (ÿ) = v0. Initial moment for each grid element, all ci(ÿ) are zero. For each element ÿ, we determine the depth, or level of refinement z(ÿ). As noted in § 4.1, there is a sequence of grids, each of which is obtained from the previous one by refinement or coarsening. We will assume that any element of the initial grid has z(ÿ) = 0. When a tetrahedron with level z1 is partitioned, we obtain two grid elements with level z1 + 1. Denote the procedure for refining the tetrahedralization ÿh as follows:

ÿh(V, T) = Bisection_method(ÿh(V, T), M), (4.3.2)

where M is some given subset T assumes that the · Equality (4.3.2)
mesh ÿh is obtained from the tetrahedralization of ÿh by the obligatory bisection of all mesh elements from the set M. Algorithm 24 for refining a tetrahedral mesh is similar to its two-dimensional counterpart (algorithm 21). As in the bisection of triangles, the set of mesh tetrahedra is constantly changing, and the set of nodes is increasing.

Algorithm 24. Bisection method of tetrahedralization ÿh(V, T)

1: Run Algorithm 23 of the initial determination of U(ÿh) 2: Put V =
V and T = T 3: **while** the set M
is not empty **do** 4: Find the smallest value
of zmin from z(ÿ) over all ÿ ÿ M Form M = {ÿ ÿM| z(ÿ) = zmin} **loop** over
5: all tetrahedra ÿ ÿ M Bisection B(ÿ) = (ÿ1, ÿ2) Add
6: ÿ1 and ÿ2 to set T Remove ÿ from
7: sets M and T **end loop loop** over all
8: tetrahedra ÿ ÿ T **if** array C(ÿ) contains
9: nonzero value **then** Add ÿ to set M
10: **end if**
eleven: **end loop** 15: 16: **end while**
12:
13:
14:

In this algorithm, the bisection operation B(ÿ) = (ÿ1, ÿ2) is a subroutine that is much more complicated in the three dimensional case due to the fact that one edge can belong to a whole set of grid elements at once (see Algorithm 25).

The refinement of one tetrahedron can lead to violation of the grid conformity only on a small number of neighboring grid elements. If the set M is small compared to the number of all elements in the grid, then the grid will be refined locally.

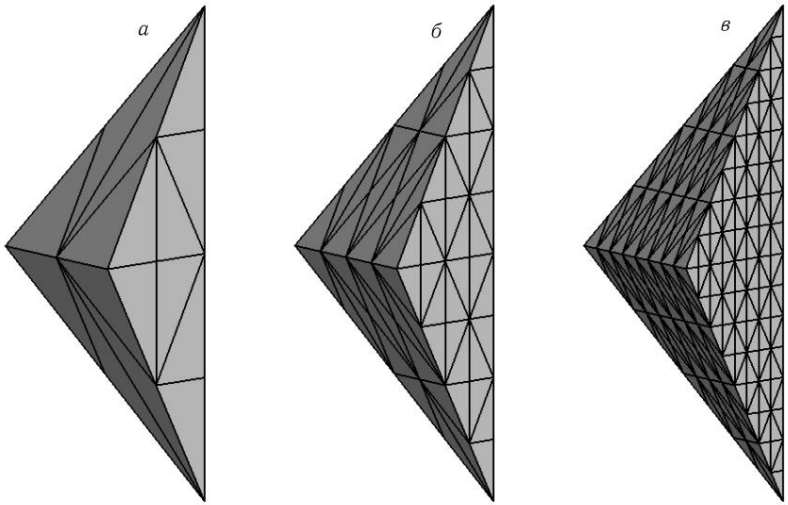
The bisection algorithm for a tetrahedral mesh is constructed taking into account that the conformity preservation process does not lead to an infinite cycle, in which the restoration of conformity on one tetrahedron leads to its violation on other mesh elements. The correctness of this assertion is substantiated in [28]. The bisection process will be successful only if the split and labeled edges are correctly determined for all elements of the initial mesh. In contrast to the initial conditions for refinement of triangulation, the consistency of singular edges is required here. Therefore, we will dwell in detail on the method of specifying split and labeled edges for an arbitrary initial tetrahedralization.

Algorithm *25*

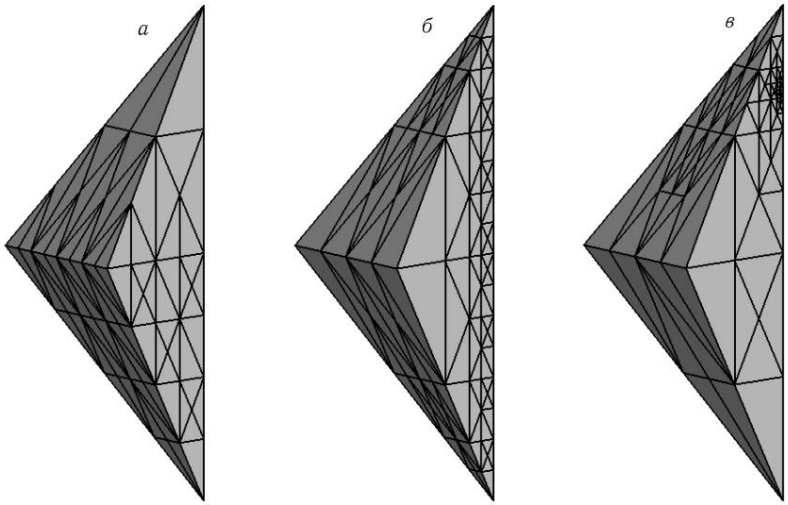
1: Determine edge r(ÿ) from D(ÿ) 2: **if**
cr(ÿ)(ÿ) = 0 **then** Place
3: new node v at the middle of edge r(ÿ) of tetrahedron ÿ
4: **else**
v = cr(ÿ)(ÿ)
5: 6: **end if**
7: Determine D(ÿ1) and D(ÿ2) from D(ÿ) 8:
Determine U(ÿ1) and U(ÿ2) and change U(ÿh) of neighbors ÿ 9: if cr(ÿ)
(ÿ) **=** 0 **then** 10: Find
vertices vk1 and vk2 of the edge r(ÿ) **loop** over
eleven: all tetrahedra ÿ ÿ ÿ(vk1) ÿ(vk2) except ÿ
12: Set cj (ÿ) = v0 for edge j with vertices vk1 and vk2 13: **end loop**
Add v0 to set V 14: 15: **end if**
16: Put cj (ÿ1) = ci(ÿ) if ci(ÿ) = 0 on the corresponding
unbreakable edge i of the tetrahedron ÿ
17: Set cj (ÿ2) = ci(ÿ) if ci(ÿ) = 0 on the corresponding
unbreakable edge i of the tetrahedron ÿ

For the initial mesh, we will define a list of its edges, each of which is given by the global numbers of its vertices. We arrange the edges in ascending order of length and assign to each edge a serial number in the new list, which will be called its *weight in what follows.* Next, for each tetrahedron, we choose an edge with the largest weight and mark it as splittable. Then, for each unsplitted face, we find the edge of this face that has the largest weight and consider it labeled. Thanks to this algorithm, any face belonging to two tetrahedra will have the same special edge for each of them, since each face has three edges with different weights. In the initial grid, the values of the binary flags s(ÿ) can be chosen, for example, to be zero. Similarly to the algorithm of multilevel bisection of triangulation, we can also speak of *multilevel bisection of tetrahedralization.* With multiple

mesh refinement, the bisection algorithm with the introduction of special edges ensures that the dihedral and flat angles of the grid elements are bounded from below by half of the minimum dihedral and flat angles of the tetrahedra of the initial mesh. This is achieved by dividing the faces into breakable and non-breakable ones and by a special choice of labeled edges that determine the further course of the bisection, as justified in [28]. Parallelization of multilevel bisection algorithms is discussed in [37, 38].



Rice. 4.11. Several levels of uniform mesh refinement



Rice. 4.12. Various local mesh refinements

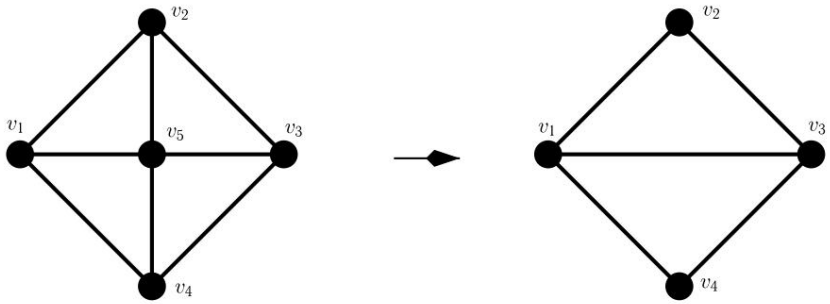
In conclusion, we will demonstrate two examples of the operation of the bisection algorithm in the three-dimensional case. On fig. Figure 4.11 shows the resulting tetrahedrizations after (a) three , *(b)* six , and *(c)* nine levels of uniform partitioning of the initial mesh consisting of two canonical tetrahedra. Rice. 4.12 illustrates various types of local condensations: to a subdomain (a), to a line *(b)* , and to a point (c).

§ 4.4. Multilevel Coarseness Algorithm

In the study of dynamic processes, an important factor is the fast rebuilding of the grid. Having only algorithms for hierarchical refinement, one would have to build many times very little different grids that take into account small changes in the dynamic process and are connected to each other only through the coarsest grid. Connecting grid coarsening algorithms provides a close connection between slightly different grids. In this section, coarsening algorithms for both 2D and 3D meshes will be described, however, all the problems encountered and methods for solving them will be illustrated using triangular meshes as an example.

An algorithm for coarsening grids obtained by bisecting triangles by the Rivara method can be found in [79]. Here we present a description of the coarsenings for triangulations and tetrahedrizations obtained by the refinements presented in § 4.2 and 4.3. The purpose of coarsening

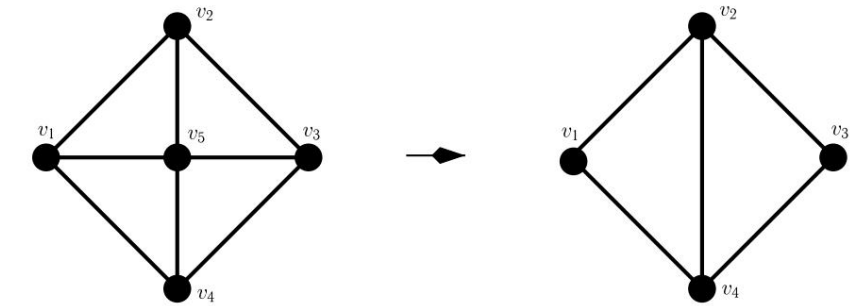
is to enlarge some of the selected grid elements by combining two adjacent ones. Note that the union can be done in different ways. On fig. 4.13 shows the union of two pairs of triangles on the same horizontal line. Rice. 4.14 illustrates another kind of coarsening. As a result, we get two different grids.



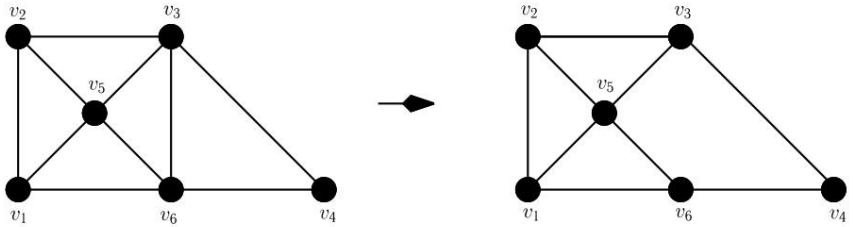
Rice. 4.13. Merging Triangles Horizontally

The coarsening process will be applied only to meshes obtained as a result of the bisection method. Therefore, the main goal of merging grid elements is to restore grids to which refinement has been applied. This can be done by keeping the history of mesh transformations. During hierarchical mesh refinement, its elements are subjected to a

different number of bisection operations. Therefore, simply joining an arbitrary triangle with any of its neighbors can result in mesh elements other than triangles, as shown in Fig. 4.15.



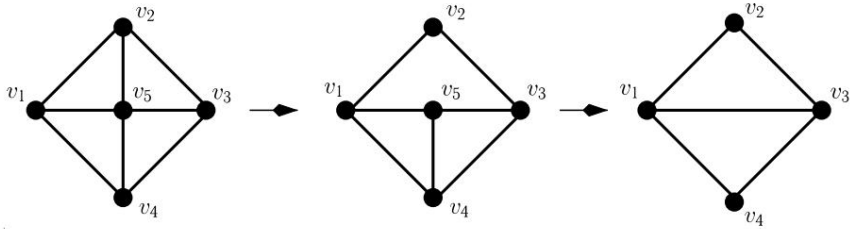
Rice. 4.14. Merging Triangles Vertically



Rice. 4.15. Union of a triangle with an arbitrary neighbor, leading to the appearance of a quadrilateral

The grid coarsening algorithm described below will help to solve this and other problems. Its main goal is that, starting with a conformal mesh resulting from the bisection procedure, the coarsening algorithm coarsens the given mesh elements and some others to maintain conformality and produces a mesh in such a form that either refinement or refinement can be applied to it again. coarsening. It was assumed in the bisection algorithm that the conformity is broken on the edge of the grid

element and can be restored by additional partitioning of adjacent elements. Similarly, when the mesh is coarsened, conformality can also be violated on the edge due to the presence of a node v in its middle, as shown in Fig. 4.16 but recover



Rice. 4.16. Restoring conformity when coarsening the mesh

it will be due to the coarsening of the grid elements whose vertex is v, and the subsequent removal of the node v itself. Therefore, we will say that the conformity is violated at the knot. Let us proceed to the

formalization of the coarsening algorithm. Let ÿh denote a triangular or tetrahedral mesh. Recall that each element ÿ ÿ ÿh has a refinement level z(ÿ). Two elements can be combined into one only if they have the same level, say, z1, then the resulting coarsened grid element will have the level z1 ÿ 1. According to the basic principles of meshing, it is impossible to enlarge the elements of the initial grid. This can be formulated as a prohibition on enlargement of elements with level zero: the level of any triangle or tetrahedron must be nonnegative. In addition to the refinement level, each element ÿ will be assigned an array h ÿ(z), which will contain the history of modification of this grid element. In the two dimensional case,

for each level z0, the only element h ÿ(z0) is defined as follows. If the bisection of triangle ÿ produces grid elements ÿ1 and ÿ2, then h ÿ1 (z0) contains the local number of the vertex of triangle ÿ1 opposite the edge along which ÿ1 borders ÿ2. As can be seen from the description of the bisection of an arbitrary triangle, h ÿ1 (z0) can never coincide with the vertex opposite to the side being split.

In the three dimensional case, h ÿ(z0)=(h(1) (z0), h(2) (z0)) is a pair of local vertices of the grid element ÿ. When the tetrahedron ÿ is split into ÿ1 and ÿ2, the common face of two new grid elements passes through the midpoint v of the split edge ÿ. Let us agree that h(1) (z0) is the local number of the vertex ÿ1 corresponding to the point v, and h(2) (z0) is the local number of the vertex ÿ1 opposite to the face along which ÿ1 borders ÿ2.

Let us first consider the process of combining two neighboring triangles ÿ1 and ÿ2 into a single grid element ÿ. In doing so, we again need the structured list U(ÿh) of neighbors of triangles defined in § 4.2.

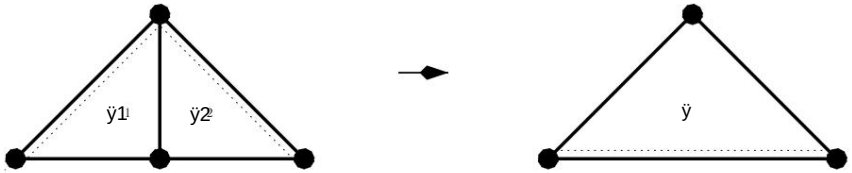
Two triangles ÿ1 and ÿ2 can be combined into one under three conditions: 1) z(ÿ1) = z(ÿ2) = z; 2) if k = h
ÿ1 (z), then uk(ÿ1)=ÿ2;
3) if m = h ÿ2 (z), then um(ÿ2)=ÿ1.

In this case, the global number of the grid vertex corresponding to the local number of the triangle vertex ÿ1 opposite to the labeled edge coincides with the equivalently defined number for the same triangle vertex ÿ2. When coarsening, this common vertex v will be removed. Since ÿ1 and ÿ2 had a common edge, they had two common mesh vertices. After deleting node v, another

the common vertex ÿ1 and ÿ2 and their other two vertices form a new grid element ÿ. To correctly define the coarsening operation, it remains to show which edge in ÿ will be marked. As r(ÿ) one should take the edge opposite to the non-removed common vertex of ÿ1 and ÿ2. Rice. 4.17 illustrates the coarsening process. Comparison with fig. 4.3 shows that this process is the reverse of the Bansch method. For simplicity, the coarsening operation will be denoted

chat like this:

$$ÿ = G(ÿ1, ÿ2). \tag{4.4.1}$$



Rice. 4.17. Triangle coarsening method

By analogy with the process of bisection during coarsening, we need to check the mesh conformality. We again assign this role to the binary array C(ÿ) of three numbers (c1(ÿ), c2(ÿ), c3(ÿ)). The value ci(ÿ) = 1 indicates problems with conformality at vertex i, i.e., the presence of a triangle ÿ whose vertex corresponding to the i-th vertex of triangle ÿ lies in the middle of the edge. Initial moment for each grid element,all ci(ÿ) are zero. We denote the procedure for coarsening the triangulation ÿh as follows:

$$ÿh(V, T) = Coarse\_method(ÿh(V, T), M), \tag{4.4.2}$$

where M is some given subset of T Equality (4.4.2) means that the grid ÿh is obtained from the grid ÿh by the obligatory coarsening of all elements from the set M. The coarsening algorithm is applicable only to the grid in which there are elements that differ from the triangles of the initial meshes, i.e., obtained as a result of the bisection method. Thus, at the start of the algorithm, we can assume that for each grid element ÿ, the set of its neighbors U(ÿ)

already defined.

The mesh coarsening process can be written in the form of Algorithm 26. In this algorithm, the operation of removing a vertex from the set of mesh nodes is best done as follows. Let's introduce some auxiliary binary array of grid node labels. In this array, the number of elements is equal to the number of nodes in the initial triangulation ÿh specified for coarsening. Initially, all elements of this array are equal to 1. Deleting a node from the grid means zeroing the corresponding component of the labels array. This approach makes it possible to use data about the remote node v with further coarsening

grid elements for which v is still a vertex. At the end of Algorithm 26, the array of labels gives information about the vertices of the resulting mesh. As a basic operation, Algorithm 26 uses the coarsening procedure ÿ = G(ÿ1, ÿ2) of triangles ÿ1 and ÿ2, which is described in detail in Algorithm 27.

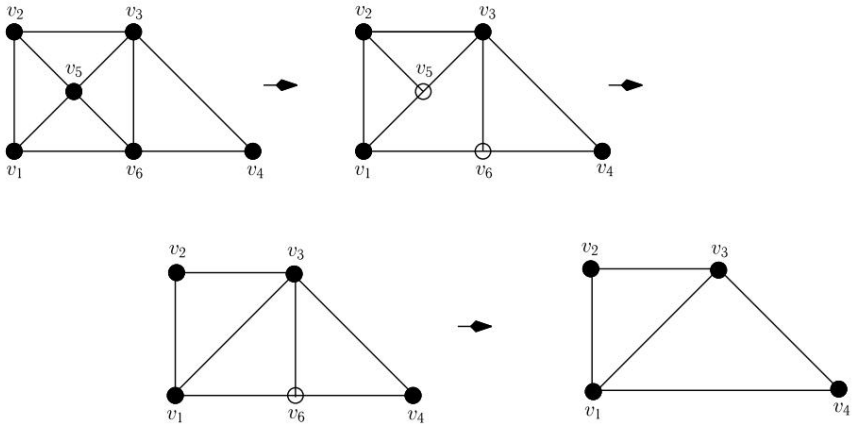
Algorithm 26. Method for coarsening the triangulation ÿh(V, T)

```
 1: Put V = V and T = T 2: loop until
set M is empty
3:      Remove ÿ from the set M if z(ÿ) = 0 4: Find zm — the
largest value from z(ÿ) over all ÿ ÿ M Form a subset M = {ÿ ÿM| z(ÿ) = zm} loop over all
5:      triangles ÿ1 ÿ M Find a ÿ2 that can be combined with ÿ1 Determine the grid
6:      node v corresponding to the vertex k of the triangle
7:          ÿ1 opposite to the labeled side if ck(ÿ1) = 0 then Using the
8:          set U(ÿh ), impose a conformity violation condition on all ÿ ÿ ÿ(v) Remove v from
            the set V end if ÿ2 is found then Coarse ÿ = G(ÿ1, ÿ2) Add ÿ to the set T
9:          and remove ÿ1 and ÿ2
10:             from M and T end if 17: end loop loop over all triangles ÿ ÿ T : 18: if array
                c(ÿ) contains 1 then
eleven:
12:
13:
14:
15:
16:
19:
20:             Add ÿ to set M end if 22: end loop 23:
21:         end
loop
```

Let us demonstrate the operation of Algorithm 26 using the example of Fig. 4.18. Let a grid ÿh consisting of five triangles be given, and the set M contains only the triangle ÿ346. In this case, we assume that the level of refinement for all grid elements is 2, except for ÿ346, for which it is equal to 1. The constructed set M coincides with M. We start with the grid element ÿ346. First, we look for a triangle with which it could be combined. Triangle ÿ356 is not suitable for this role, as it has a different level of refinement: z(ÿ356) = z(ÿ346). Therefore, we can only impose the conformality violation condition on all grid elements containing the point v6.

Algorithm 27. Coarseness of triangles ÿ = G(ÿ1, ÿ2) and modification of auxiliary information about the mesh

1: Using D(ÿ1), determine the vertex v corresponding to the reb
ru r(ÿ1)
2: Find a common vertex v1 for ÿ1 and ÿ2 other than v 3: Determine
V(ÿ) from V(ÿ1) and V(ÿ2) 4: Set r(ÿ) equal to
the local number of vertex ÿ, respectively
current v1
5: Determine U(ÿ) and correct U(ÿh) for neighbors ÿ1 and ÿ2



Rice. 4.18. Coarse algorithm illustration

There are two such triangles: ÿ156 and ÿ356. The v6 point itself is removed from the list of grid vertices. Triangles ÿ156 and ÿ356 have the same grinding level. They can be united into one triangle ÿ136 and remove ÿ156 and ÿ356 from the set M, simultaneously declaring the vertex v5 deleted. Under this coarsening operation, the conformity of the grid begins to be violated at the point v5; therefore, triangles ÿ125 and ÿ235 are added to the set M. Combining them into a grid element ÿ123 leads to the preservation of grid conformity. All triangles now have the same level of refinement, but the set M still contains the triangle ÿ346. Now it can be combined with the grid element ÿ136 and get a triangle ÿ134. As a result, we have a coarsened conformal grid.

Now we describe the process of combining two neighboring tetrahedra ÿ1 and ÿ2 into a single grid element ÿ. We again use the structured list U(ÿh) of neighbors described in § 4.3 and the vector C(ÿ) of four numbers c1(ÿ), c2(ÿ), c3(ÿ) and c4(ÿ), showing the violation of conformality at the vertex i.

Tetrahedra ÿ1 and ÿ2 can be combined into one if four conditions are met: 1) z(ÿ1) = z(ÿ2) = z;
2) if k = h(2) (z), then
uk(ÿ1)=ÿ2; 3) if m = h(2) (z), then
um(ÿ2)=ÿ1; h(1) 4) the vertex ÿ1 with index
coincides with the vertex ÿ2 with index h(1) (z).
rom

In contrast to the two-dimensional case, when the tetrahedron ÿ1 is coarsened, there is not enough information about which grid element ÿ2 it can be combined with. It is also necessary to know the number of the vertex v, which will be removed when a new tetrahedron is formed, see condition 4. It cannot be determined from D(ÿ1) and D(ÿ2). Therefore, during the bisection, this number is stored as the value of the elements h(1) (z) and h(1) (z). When coarsening, the common vertex v is removed from the grid. The segment connecting h(2) (z) with h(2) h(1) (z) and splitting through the tetrahedron ÿ. The labeled edges of ÿ will be those that were decomposable for ÿ1 and ÿ2. If for ÿ1 and for ÿ2 the split and labeled edges lie in different planes, while the singular edges of ÿ lie in the same plane, then we define s(ÿ) = 1; otherwise, s(ÿ) = 0. The operation of combining two tetrahedra will again be denoted by (4.4.1), and the
procedure for coarsening the tetrahedralization of ÿh by (4.4.2). The coarsening algorithm is applied only to tetrahedralization, in which there are elements that differ from the tetrahedra of the initial mesh, i.e., obtained as a result of bisection. Initially, we can assume that for each grid element ÿ the list of its nearest neighbors U(ÿ) is already defined and the set ÿ(v) is known for each node.

The mesh coarsening process can be represented by a modified algorithm 26. It uses a different basic procedure ÿ = G(ÿ1, ÿ2) for combining tetrahedra ÿ1 and ÿ2 and modifying auxiliary information (see Algorithm 28). Another difference of the modified algorithm 26 is that in the three-dimensional case, for each vertex v, it is necessary to constantly maintain and correct information about the set ÿ(v).

The described coarsening algorithms have one common feature. If some grid element ÿ needs to be coarsened, and there is no candidate for combining with ÿ due to the fact that the neighbors have a higher level of refinement, then the algorithm imposes a requirement on the neighbors to be coarsened and, after waiting for the required candidate for ÿ, performs its coarsening. Since the coarsening of one grid element
often leads to pairing of a small number of grid elements, the presented algorithms lead to a local change in the grid.

Algorithm 28. Coarseness of tetrahedra ÿ = G(ÿ1, ÿ2) and modification of auxiliary information about the mesh

1: Find the mesh vertex v corresponding to 2: Determine $h^{(1)}_{y1}(z)$ V(ÿ) from V(ÿ1) and V(ÿ2) from v 3: Assign number r(ÿ) to the edge connecting the nodes corresponding to h (2) (z) and h(2) (z) 4: Determine l1(ÿ), ing $l2(ÿ)_{y1}$ and s(ÿ) from $ÿ2$ D(ÿ1) and D(ÿ2) 5: Determine U(ÿ) and adjust U(ÿ h) at neighbors ÿ1 and ÿ2 6: Remove ÿ1 and ÿ2 from sets ÿ(v), where v ÿ V(ÿ1) V(ÿ2) 7: Add ÿ to sets ÿ(v), where v ÿ V(ÿ1) V(ÿ2) and v = v

§ 4.5. Algorithms for building dynamic grids

The described algorithms for multilevel hierarchical refinement and coarsening of triangular and tetrahedral meshes are convenient tools for constructing dynamic meshes. They make it possible to quickly rebuild the grid when modeling nonstationary processes, adapting it to the features of a changing solution.

Algorithm 29. Building a dynamic mesh

1: Given a quasi-uniform mesh ÿ0h(V0,T 0) with step d0 2: Determine labeled edges in triangles or binary flags, split and labeled edges in tetrahedra

3: **loop** over n = 1, ..., K 4: Calculate the function s(xÿ, tn) for each element 5: Set V = Vnÿ1 and T = T nÿ1

6: Form a set M ÿ T from elements ÿ satisfying the condition diam(ÿ) > s(xÿ, tn) **if** set M is not empty **then** ÿnh(Vn, T n) =

7: Bisection_method ÿnh(V, T), M Set V =

8: Vn and T = Tn. Go to step 6 **end if** Output V and T Generate a

9: set M ÿ T from elements ÿ satisfying diam(ÿ) < s(xÿ, tn) **if**

10: set M

eleven: is not empty **then**

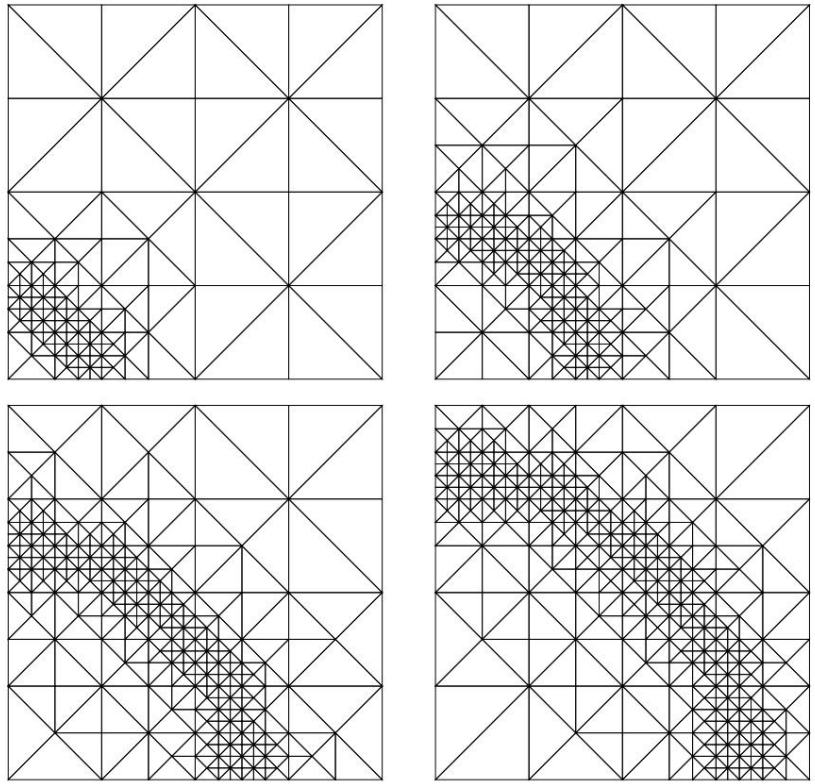
12: ÿnh(Vn, T n) = Coarse_method ÿnh(V, T), M Put V = Vn and T = T n. Go to step 12 **end if** 17: **end loop**

13:

14:

15:

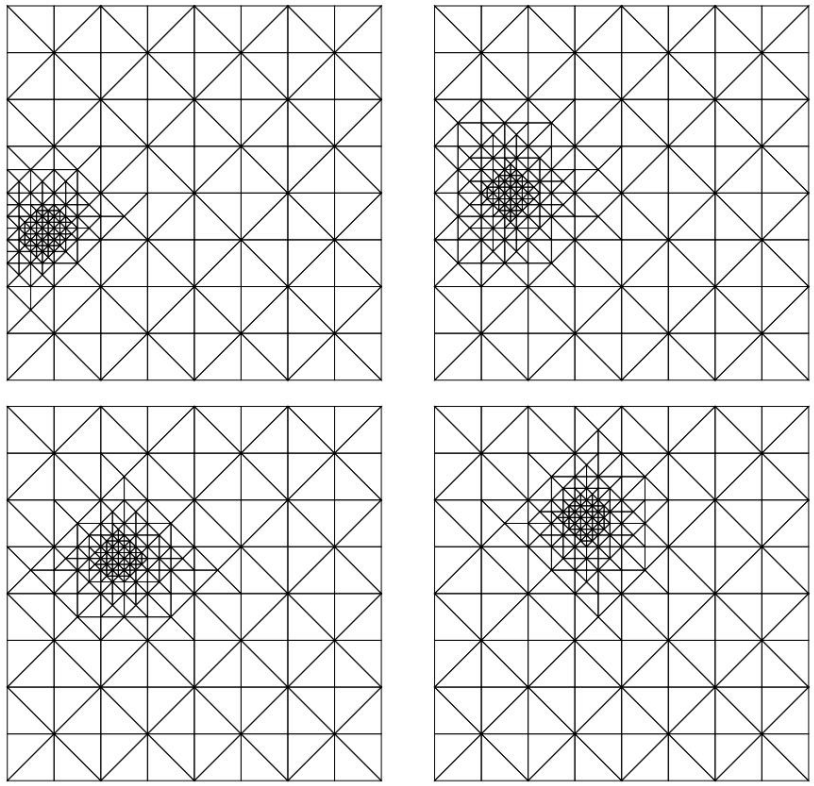
16:



Rice. 4.19. Circular front movement centered at the bottom left

Let us consider the construction in the domain ÿ of a set of grids ÿnh, in which the local size of the grid element is determined by the grid step function s(x, tn). The function s(x, t) can be piecewise constant, for example, to track the motion of a shock wave in a liquid or the propagation of a crack in a solid. In general, s(x, t) is determined in terms of a posteriori error estimates. Let ÿ0h be a given initial quasi-uniform triangulation or tetrahedralization of the domain ÿ with step d0. The process of

creating dynamic grids can be schematically represented in the form of Algorithm 29. The membership of a grid element ÿ in the set M depends on the value s(xÿ, tn) at its geometric center xÿ. The process of constructing a dynamic mesh is a repeated application of refinement and coarsening procedures. The essence of the process is that, having first constructed a grid that condenses to a certain subdomain, then we can



Rice. 4.20. Motion of a point feature

coarsen it in the region of condensation, and then refine it again in another subregion without rebuilding the entire mesh. Acting in this way, one

can model both the movement of an entire front (Figure 4.19) and the movement of some local feature (Figure 4.20). The use of hierarchically nested grids provides efficient reinterpolation of grid functions in the finite element solution of nonstationary partial differential equations on dynamic grids.

Chapter 5

REBUILDING SIMPLICIAL GRIDGES VIA LOCAL MODIFICATIONS

This chapter will describe methods for rebuilding triangular and tetrahedral meshes. The algorithms described in the previous chapters are mainly focused on the construction of regular grids. In some applied problems, it becomes necessary to construct grids with special properties, for example, anisotropic grids, in which the simplices are strongly elongated along the boundary of the region, the shock wave front, or grids that thicken towards the destruction zone. The presented set of algorithms makes it possible to build such grids by locally modifying the elements of the original grid. In addition to this, the technology of local modifications can be used to unravel the mesh, build a mesh with desired properties, and adapt the mesh to a given mesh solution.

§ 5.1. Principles of organizing algorithms

Let ÿh(V, T) denote the computational grid with the set of nodes V and the set of elements (simplices) T . Let N(ÿh) be the number of simplices ÿ in this grid. Recall that we use the term "element" or "simplex" as a generic term for triangle and tetrahedron. The first important principle of organizing algorithms for rebuilding the grid ÿh is *the locality* of basic

operations. Each mesh modification affects only a few simplices, opening up the possibility of using efficient parallelization techniques. The grid can be modified independently and simultaneously in several

places.

Since the computational complexity of re-meshing is proportional to the number of local modifications, the distribution of work among several processors is greatly simplified. For example, synchronization of processors is possible after each local modification, i.e., almost immediately after the processor receives this command.

Locality is also the key to developing robust and efficient algorithms. First, you can analyze several options for changing the mesh topology and choose the best one. Secondly, our experience shows that consistent change

meshing in spatially unrelated places ultimately requires fewer local modifications than sequential meshing around neighboring elements.

The second important principle of organizing mesh rebuilding algorithms is *the monotonic increase in mesh quality.* Recall that the concepts of the quality of a grid Q(ÿh) and the quality of a simplex Q(ÿ) were introduced in § 2.1 for regular grids. In § 6.2 these concepts will be generalized to the case of anisotropic meshes. Since we do not use the exact formula for Q(ÿ) in this chapter, we will simply assume that the procedure for calculating the quality of a simplex is known, and that

$$0 < Q(ÿ) \le 1.$$

The quality of a simplex is close to 1 if its shape and size in the vicinity of a particular point x ÿ ÿ in the computational domain are close to the parameters specified by the user. For example, for a regular grid, it is natural to require that Q(ÿ) = 1 for an equilateral (regular) simplex of diameter h(x ÿ), where the function h(x) is given by users and x ÿ denotes the geometric center of the simplex. The mesh quality is calculated in the same way as before:

$$Q(ÿh) = \min_{ÿ ÿ ÿh} Q(ÿ).$$

Thus, the quality of the mesh is the same as the quality of the worst case simplex in the grid.

Similarly, the quality of an arbitrary set of simplex cos coincides with the quality of the worst simplex in this set. For example, for a superelement ÿ(v) formed by simplices with a common vertex v, we have

$$Q(ÿ(v)) = \min_{ÿ ÿ ÿ(v)} Q(ÿ).$$

Recall that, according to the notation introduced in § 2.1, normal font (v) is used to designate a vertex as a mesh object, and bold font (v) is used for the vertex spatial coordinate vector v.

Denote by Q(i)(ÿh) the mesh quality after i local modi fications. The principle of monotonic increase in mesh quality says:

$$Q(i)(ÿh) \ge Q(j)(ÿh), \quad \text{If} \quad i < j.$$

Note that the last formula does not guarantee that the mesh quality will tend to 1 with an increase in the number of local modifications, since the mesh quality can only be increased by increasing the quality of the worst mesh element. Unfortunately, there are cases when local algorithms for modifying the mesh around the worst simplex cannot improve its quality. In these cases, local algorithms are applied to the second worst mesh simplex,

then to the third, and so on (see § 5.2). This approach requires ordering the simplices in ascending order of quality using the methods described in § 2.3.

§ 5.2. Rebuilding triangulations

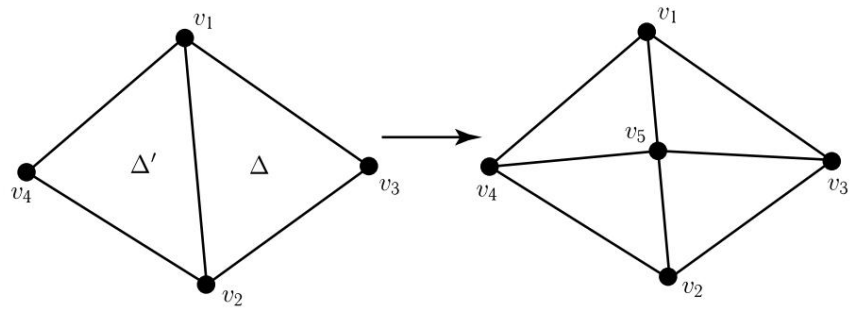
The minimum set of data structures that define a triangular mesh includes three structured lists for triangles, boundary edges, and node coordinates. Rebuilding the grid will require several auxiliary data structures, which will be described below.

Reliable reconstruction of a triangular grid requires the use of both algorithms that change the topology of the grid and algorithms that change the shape of triangles. Consider a triangle ÿ(v1, v2, v3) (or ÿ123 for simplicity of notation) with vertices v1, v2 and v3 and five basic local algorithms to improve its quality. We describe the basic algorithms for a triangle ÿ located strictly inside the region. The generalization of these algorithms to the boundary triangle is discussed in the comments to the algorithms. Let ÿ(ÿ123) denote the set of triangles including ÿ123 and those of its neighbors that have at least one point in common with ÿ123:

$$ÿ(ÿ123) = \bigcup_{i=1}^{3} ÿ(vi).$$

Basic operations change the grid inside the superelement ÿ(ÿ123). The grid at the boundary ÿ(ÿ123) and in the remaining computational domain does not change. Based on this property, one can construct a reliable parallel algorithm for rebuilding a triangular mesh.

The first basic algorithm puts a point in the middle of an edge of the triangle ÿ, splitting ÿ and the adjacent triangle in half. Rice. 5.1 illustrates the operation of this algorithm for the internal mesh edge e12, common to triangles ÿ and ÿ. Algorithm 30 describes the first basic algorithm for triangle ÿ. Algorithm 30 is easily generalized to the case of a triangle ÿ with boundary edges. If e is a boundary edge, then the neighboring triangle ÿ does not exist and Q0 = Q(ÿ). Splitting a boundary edge may also require updating the list of boundary edges. It is possible that algorithm 30 will never perform step 5. In this case, we move on to the next basic operation to improve the quality of ÿ. The computational domain model can be specified using curvilinear boundaries. In this case, an additional grid validation check must be included in the basic algorithm 30 . Assume that the edge e12 with vertices v1 and v2 lies on some curvilinear boundary (internal or external), and suppose



Rice. 5.1. Local modification of mesh topology after v5 node addition

Algorithm 30. Putting a node on an edge of a triangle

1: **loop** over all edges e of triangle ÿ 2: Find neighboring triangle ÿ with vertices v1, v2 and v4 as shown in fig. 5.1.
3: Determine Q0 = min{Q(ÿ), Q(ÿ)} Put a test node v5 in the middle of the edge e and calculate the qualities of triangles ÿ315, ÿ235, ÿ145 and ÿ425. Define Q1 = min{Q(ÿ315), Q(ÿ235), Q(ÿ145), Q(ÿ425)} **if** Q1 > Q0 **then** Add node v5 to the grid and replace triangles
4: ÿ and ÿ with
5: triangles ÿ315, ÿ235, ÿ145 and ÿ425. Finish algorithm 6: **end if** 7: **end loop**

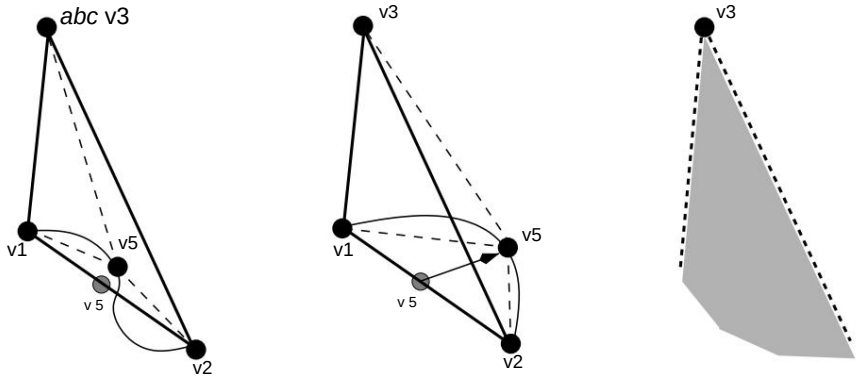
that each point of this edge is mapped to a single point of the curvilinear boundary. In practice, for each curvilinear boundary, it is convenient to use a one-parameter function F(t) such that

$$v1 = F(t1), \qquad v2 = F(t2),$$

and intermediate values of t define a point on a curvilinear edge. The parameters t1 and t2 can be stored in an additional structured array of boundary edges. The coordinates of the midpoint of the edge e12 and the point projected onto the curvilinear boundary are calculated as follows:

$$v_5 = (v1 + v2)/2, \qquad v5 = F((t1 + t2)/2).$$

For a linear function F(t), the points v5 and v5 coincide. For a smooth function F(t), the second formula gives a node located close to v 5. Otherwise, various methods of projecting the midpoint v5 onto a given curve can be used. Once the v5 node is defined, the correctness of the mesh needs to be verified. As shown in fig. 5.2, the projection of node v5 on the outside



Rice. 5.2. Two cases of v5 node projection onto an external curvilinear boundary *(a,* b) and an admissible set of v5 node positions *(c,* shaded area)

A curvilinear boundary can lead to both a regular grid (a) and an inverted grid (b).

Let Sÿ315 denote the algebraic area of the triangle ÿ315:

$$S\ddot{y}315 = \overline{1}2 \det\{v1 \ddot{y} v3, v5 \ddot{y} v3\}, \tag{5.2.1}$$

where det( ) denotes the determinant of the 2 × 2 matrix composed of the vectors v1 ÿ v3 and v5 ÿ v3. Geometric interpretation of this determinant through the vector product

$$\det\{v1 \ddot{y} v3, v5 \ddot{y} v3\} = (v1 \ddot{y} v3) \times (v5 \ddot{y} v3)$$

shows that the algebraic area differs from the ordinary area of a triangle only in sign. Both definitions of area are the same when the vertices of the triangle are ordered counterclockwise (when looking at the triangle from above). Consider auxiliary triangles constructed using v5 instead of v5. The grid remains correct if the algebraic areas of the constructed triangles (ÿ315 and ÿ235) have the same sign as the algebraic areas of the corresponding auxiliary triangles. A more detailed analysis shows why the algebraic area method leads to correct mesh verification. Consider first the case when the neighboring triangle ÿ does not exist. The straight line given by the edge e23 splits the plane into two half-planes. Let ÿ1 be the half-plane containing the vertex v1. This half-plane

divided as a set of points

$$\ddot{y}1 = x \ddot{y} R2 : \det\{x \ddot{y} v2, x \ddot{y} v3\} > 0 .$$

Now consider the partition of the plane into two half-planes of the straight line defined by the edge e31. Let ÿ2 be the half plane containing the vertex v2:

$$ÿ2 = x ÿ R2 : \det\{x ÿ v3, x ÿ v1\} > 0 .$$

We construct the set D as the intersection of these half-planes:

$$D = ÿ1 ÿ ÿ2.$$

The set D is shown in fig. 5.2, c. Note that this set is not limited. **Lemma 5.2.1.** *Under the conditions of exact*

arithmetic, the projection v5 =
= F((t1 + t2)/2) *leads to a regular grid if* v5 ÿ *D.*

Proof. Consider Fig. 5.2 and the line defined by the edge e23. This line divides the plane into two half-planes. If the nodes v5 and v1 are in different half planes, then the triangle ÿ153 will intersect the edge e32. Thus, the first sufficient condition is that the node v5 must be in the half plane ÿ1. Consider the line given by the edge e31 and the triangle ÿ325. Similar reasoning leads to the conclusion that the node v5 must be in the half-plane ÿ2. Thus, if

the node v5 belongs to the set D, then the interiors of no two triangles intersect. Therefore, the grid is correct.

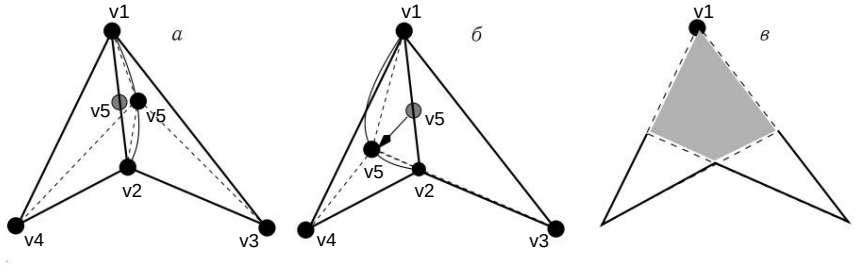
To determine the correct sign of the algebraic area, we apply the algorithms described in § 3.3. These algorithms use the function d(v1, v2, v3) which returns either the correct sign of the algebraic area S ÿ123 or zero if rounding errors could lead to an incorrect result. For example, a triangle and a line segment that do not intersect may be misinterpreted as intersecting. Errors of this type are acceptable in re-meshing, since they only narrow the set D. Therefore, in the worst case, this basic operation will not be performed. The use of the d(v1, v2, v3) function is also recommended for other basic operations involving the algebraic areas of triangles.

Lemma 5.2.1 gives a sufficient condition for the location of v5. The same condition becomes necessary when both neighboring triangles exist. Otherwise, the analysis of the necessary condition becomes nontrivial. **Lemma 5.2.2.** *Let* ÿ315 *and* ÿ235 *be auxiliary triangles constructed using* v5

instead of v5. *Then the grid remains correct if the algebraic areas of the triangles* ÿ315 *and* ÿ235 *have the same sign as the algebraic areas of the corresponding auxiliary triangles.*

Proof. According to the definition of the half-plane ÿ1, the algebraic area S ÿ235 does not change sign when the node v5 lies in this half-plane. Similarly, the algebraic area S ÿ315 does not change

sign when v5 lies in the half-plane ÿ2. Thus, the preservation of the sign of the algebraic areas is equivalent to the fulfillment of both conditions, i.e., the node v5 belongs to the set D. According to Lemma 5.2.1, we conclude that the grid is correct.



Rice. 5.3. Two cases of v5 node projection onto an internal curvilinear boundary *(a,* b) and an admissible set of v5 node positions *(c)*

Consider the case where the edge e12 is part of an internal curvilinear boundary (see Fig. 5.3). We define two additional half-planes associated with the edges e14 and e42: ÿ3 = x ÿ R2 : det{x ÿ v1, x ÿ v4} > 0

And

$$ÿ4 = x ÿ R2 : \det\{x ÿ v4, x ÿ v2\} > 0 .$$

We construct the set D as the intersection of four half-planes:

$$D = ÿ1 ÿ ÿ2 ÿ ÿ3 ÿ ÿ4.$$

The set D is shown in fig. 5.3, c. Note that this set is bounded, in contrast to the previous case. As shown in the figure, if the union of triangles ÿ and ÿ is not convex, then the set D includes only a part of these triangles. Otherwise, the set D coincides with the interior of the convex quadrilateral ÿ ÿ ÿ

.

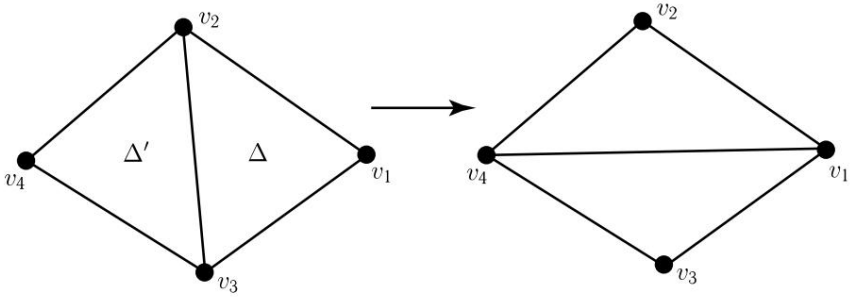
Lemma 5.2.1 remains valid, but only with a new definition of the set D. Lemma 5.2.2 will also be true if we compare the signs of the algebraic areas of four auxiliary triangles with a common vertex v5 and four triangles with a common vertex v5. *The second basic algorithm* is applied to a pair of triangles that share an edge. If the union of these

triangles forms a strictly convex quadrilateral, then there exists a second partition of this quadrilateral into two triangles. Algorithm 31 determines which of the partitions improves the quality of the mesh.

Algorithm 31 can be easily generalized to the case of a triangle ÿ with edges on the interior boundaries. Since such edges cannot be removed from the mesh, they are not considered in the algorithm.

Algorithm 31. Edge replacement

1: **loop** over all edges e of triangle ÿ 2: Find neighboring
triangle ÿ with vertices v2, v3, and v4 as shown in fig. 5.4. If triangles ÿ and ÿ do not form a
strictly convex quadrangle, then go to the next edge Determine Q0 = min{Q(ÿ), Q(ÿ)}
Calculate the qualities of triangles ÿ124 and ÿ134. Determine Q1 = min{Q(ÿ124),
3: Q(ÿ134)} **if** Q1 > Q0 **then** Replace triangles ÿ and
4: ÿ with triangles ÿ124 and ÿ134. End algorithm 7: **end if** 8: **end loop**
5:
6:

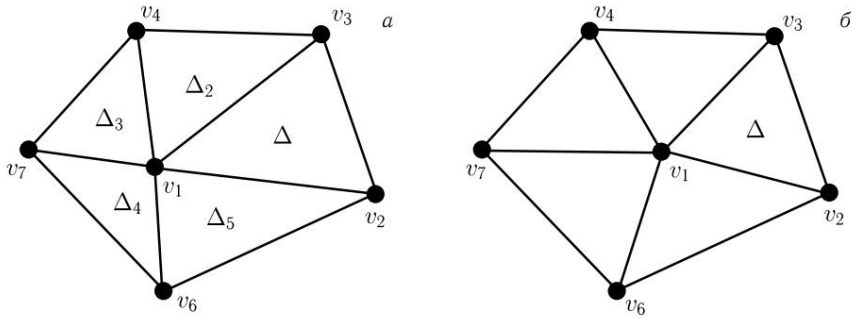


Rice. 5.4. Local modification of the mesh topology after replacing edge e23 with edge e14

The edge change operation is sometimes called *a flip* and has another interesting application. Flip can be used to build a Delaunay triangulation based on an existing mesh. To do this, we check the Delaunay condition for a pair of neighboring triangles ÿ and ÿ, and if the condition is not met, then we replace them with triangles ÿ124 and ÿ134 (Fig. 5.4). The theoretical substantiation of this approach is based on the following theorem [18]. **Theorem 5.2.1.** *A Delaunay triangulation can be obtained from any conformal triangulation by successively applying the edge replacement algorithm to a pair of*

triangles that do not satisfy the Delaunay condition. The third basic algorithm changes the position of the mesh vertex. Let v1 be one of the vertices of the triangle ÿ. Consider a superelement ÿ(v1) formed by triangles with a common vertex v1, as shown in Fig. 5.5, a. On fig. 5.5b ,

the node v1 is placed at the center of mass of the superelement ÿ(v1). This approach is used in methods for improving the shape of triangles. In general, the optimal position of node v1



Rice. 5.5. Local modification of the mesh by moving node v1

should increase the quality of the worst triangle in superelement ÿ(v1). As a rule, the quality Q(ÿ(v1)) is a non-linear function of the coordinates of the vertex v1. In this case, the search for the optimal position v1 will require the use of fairly complex computational methods.

Since the position of neighboring nodes can be changed in the process of constructing the optimal mesh, there is no need to use exact optimization methods. Algorithm 32 solves the problem of the optimal vertex position approximately. Let v1 = (x1, y1) and, for simplicity, Qÿ = Q(ÿ(v1)). Denote the approximate gradient of the quality function Qÿ by ÿhQÿ. The components ÿhQÿ are calculated based on finite differences:

$$\frac{ÿQÿ}{ÿx} \approx \frac{Qÿ(x1 + ÿx, y1)}{ÿx}, \qquad \frac{ÿQÿ}{ÿy} \approx \frac{Qÿ(x1, y1 + ÿy)\ ÿy}{ÿy}. \tag{5.2.2}$$

A reasonable value for increments ÿx and ÿy is the square root of machine precision: ÿx = ÿy = $\sqrt{ÿ}$. For the reliability of the algorithm, it is necessary to check that the calculated increment is significantly less than the diameter of the superelement, for example, less than 1% of the length of the minimum edge:

$$ÿx = ÿy = \min\{ ÿ, \min_{vi,vjÿÿ(v1)} |vi ÿ vj|/100\}.$$

The node v1 is shifted in the direction of the approximate gradient until the maximum Qÿ is reached:

$$v1 := v1 + ÿ\ ÿhQÿ, ÿ \geq 0. \tag{5.2.3}$$

To search for this maximum, various methods can be used, for example, the bisection method or various modifications of the Levenberg–Marquardt method [65]. Note that the node v1 cannot go beyond the superelement, since the quality Qÿ tends to zero as v1 approaches the boundary.

However, moving a node even within a superelement can confuse the mesh.

Consider the case where v1 is an internal mesh node with the initial position v1 and the final position v1. Let ÿi be triangles with a common vertex v1, ÿi be triangles with a common vertex v1, and ei be a boundary edge of the superelement ÿ(v1) belonging to the triangle ÿi. This edge defines a line that divides the plane into two half-planes. Let ÿi be the half-plane containing the point v1. We define the set D as follows:

$$D = ÿ1 \ ÿ \ ÿ2 \ ÿ \ ... \ ÿ \ ÿn,$$

where n is the number of triangles in the superelement. If the point v1 ÿ D, then the interior of one of the triangles ÿi will intersect the interior of the triangle located on the other side of the edge ei. Note that the admissible set D coincides with the superelement ÿ(v1) when it is convex. Otherwise, D is a subset of ÿ(v1), similar to the configuration shown in Fig. 5.3.

Algorithm 32. Node shift

1: **loop** over all nodes v1 of the triangle ÿ 2: Find the triangles ÿ1, ÿ2, ..., ÿn that form the superelement ÿ(v1). Let ÿ=ÿ1, as shown in fig. 5.5, where n = 5 Calculate the approximate gradient ÿhQÿ using formulas
3: (5.2.2) Calculate the maximum possible displacement of the node v1 in the
4: direction ÿhQÿ that does not violate the topology of the superelement. Let ÿ = ÿmax for this extreme position 5: Find the value of ÿ in the half-open interval [0, ÿmax) that

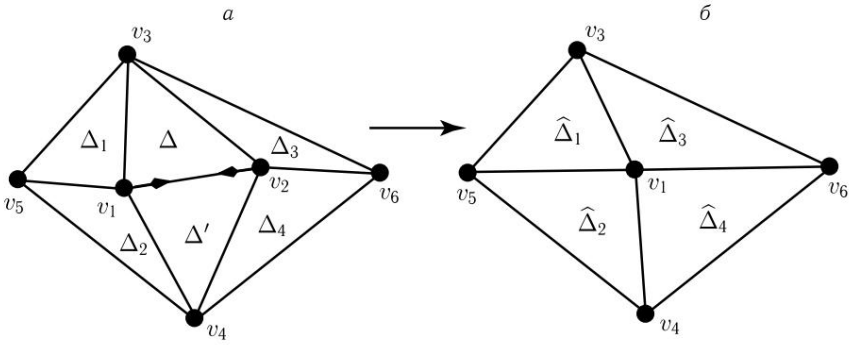
which maximizes the quality Qÿ If ÿ >
6: 0, then store the new position of the node v1 in the grid and finish Algorithm 7: **end loop**

Note that such configurations of the superelement ÿ(v1) are possible, when the motion along the approximate gradient does not increase the quality of Qÿ. In this case, ÿ = 0, and the algorithm proceeds to the next vertex of the triangle ÿ. If ÿ = 0 for all vertices of ÿ, then we apply the following basic operation to this triangle.

The maximum possible displacement of the vertex v1 in the direction ÿhQÿ that does not violate the topology of the superelement ÿ(v1) (see step 4 of Algorithm 32) requires finding the intersection of the line defined by this direction with the set D. Another method is based on comparing the signs of the algebraic areas of triangles ÿi and ÿi. They have the same sign if v1 is inside D.

Algorithm 32 can be easily generalized to the case of a triangle with vertices lying on the inner and outer boundaries. Such mesh nodes can only move along boundary edges. Movement along curvilinear edges should be accompanied by checking the correctness of the grid. As before, for this it suffices to control the conservation of the sign of the algebraic area of the triangles ÿ1, ..., ÿn.

The fourth basic algorithm removes an edge from the mesh. Algorythm 33 can be interpreted as follows. Let us start moving the vertices of the edge towards each other, possibly with different speeds. When vertices merge into one, the topology of the mesh changes. As shown in fig. 5.6, two pairs of edges merge, each into one edge, and the two triangles disappear from the grid.



Rice. 5.6. Local mesh modification by removing edge e12

The edge removal algorithm is the most complex of the basic algorithms discussed, since it modifies a larger number of triangles. Nevertheless, there is a simple method for checking the correctness of the modified grid. Not surprisingly, it is again based on checking the algebraic areas of modified triangles. The modified grid remains correct if the algebraic areas of the triangles retain their sign. Consider, for example, the triangle ÿ263 (Fig. 5.6, a), which goes into the triangle ÿ163 (Fig. 5.6, b). Let the algebraic areas of these triangles be

telny.

The generalization of Algorithm 33 to the case of a triangle with boundary edge e12 requires additional analysis. Removing this edge from the mesh can lead to a significant change in the boundary. Even a slight local change in the boundary can accumulate from operation to operation. This should be kept in mind when developing numerical methods, in which the preservation of the area of the computational domain or its subdomain is especially important.

Removing the boundary edge of the mesh is also wise to avoid when the only edge that represents an important part of the mesh is removed.

Algorithm 33. Removing an edge

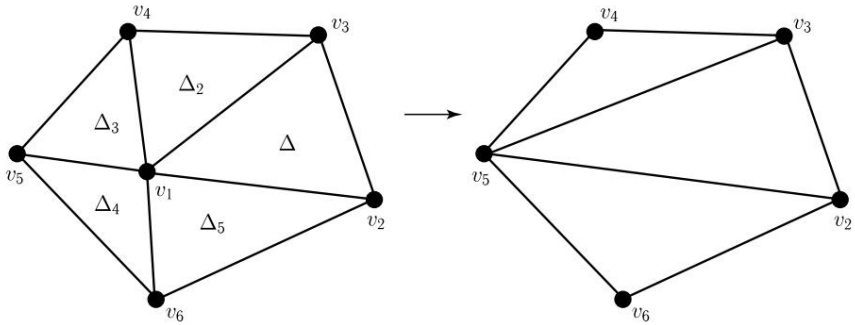
1: **loop** over all edges e of triangle ÿ 2: Find
neighboring triangle ÿ with vertices v1, v2 and v4 as shown in fig. 5.6 3: Find triangles
ÿ1, ÿ2, ..., ÿn that are: (a)
different from triangles ÿ, ÿ; (b) are included in the superelement ÿ(v1) or ÿ(v2) Determine
a new common (virtual) position v1 of the vertices (v1 + v2)
4:
v1 and v2: $v1 = \frac{1}{2}$
5: Determine (virtual) triangles ÿ1, ÿ2, ..., ÿ by shifting the vertices v1 and v2 of the
original triangles in v1 If the algebraic areas of the triangles ÿi and ÿi have
different signs for at
6: least one i, then remove all virtual objects and go to the next edge Determine
the qualities Q0 = min{Q(ÿ1), Q(ÿ2), ..., Q(ÿn)} and Q1 = min{Q(ÿ1), Q(ÿ2), ..., Q
(ÿ n)} **if** Q1 > Q0 **then** Delete vertex v2 and triangles ÿ, ÿ from the
7: grid; change vertex v1 to v1 and triangles ÿi to ÿi. End algorithm **end if** 11: **end**
loop
8:
9:
10:

borders. Such situations often arise when using coarse grid approximations of a given boundary and when constructing grids that are strongly elongated in one direction. The fourth step of Algorithm 33 requires choosing a

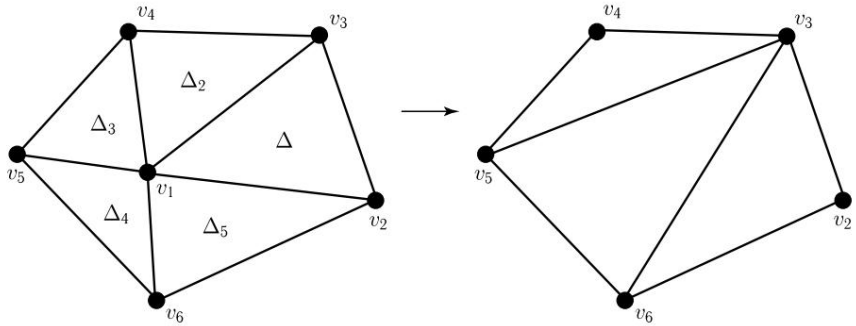
common vertex v1 for v1 and v2. On the one hand, it is natural to consider the problem of the optimal position of the vertex v1. On the other hand, since a similar problem is considered in the basic vertex shift algorithm, the complication of Algorithm 33 is not required. The optimal position of the v1 vertex can be achieved by sequentially applying two basic algorithms. Therefore, we propose to consider 1 (v1 + v2). only one possible position of this vertex: v1 = 2

The fifth basic algorithm removes a node from the grid along with the triangles containing it. The void created in the mesh is filled with a smaller number of triangles, which are constructed using only existing mesh nodes. On fig. Figure 5.7 shows that removing the inner vertex v1 reduces the number of triangles by exactly two.

Algorithm 34 is a special case of algorithm 33, where the edge e15 is removed by moving node v1 to node v5. Such



Rice. 5.7. First local modification of the mesh by deleting node v1



Rice. 5.8. Second local modification of the mesh by deleting node v1

the point of view is presented in [35] and allows one to significantly simplify the software implementation of this algorithm. We distinguish Algorithms 34 and 33 because the new superelement triangulation ÿ(v1) at step 4 of Algorithm 34 is not unique. An alternative new triangulation is shown in fig. 5.8 and corresponds to the offset of node v1 to node v3. Note that this basic operation is faster than the edge removal operation. Let us dwell in more detail on the preservation of the topology of boundaries between different materials. Let in Fig. 5.7 triangles ÿ,

ÿ2 and ÿ3 belong to one material, and triangles ÿ4 and ÿ5 belong to another. This means that the edges e23, e34 and e45 must remain the boundary of the first material after the operation of deleting node v1, i.e. this node can only be moved to node v5 or to node v2.

The combination of basic algorithms opens up wide possibilities for rebuilding computational grids using various criteria described in Chap. 6. Consider what data structures are required for the efficient implementation of basic algorithms.

First, it is necessary to maintain a list of triangles ordered in ascending order of their quality. Basic algorithms change the positions of triangles in this list, remove triangles from the list

Algorithm 34. Deleting a node

1: **loop** over all vertices v of triangle ÿ Let triangles ÿ1, ÿ2, ..., ÿnÿ1 form a superelement 2: ÿ(v) with vertices v2, v3, ..., vn, as shown in Fig. 5.7, where ÿ=ÿ1 and n = 6 **loop** i = 2, ... , Split the superelement ÿ(v) into virtual triangles ÿ ÿ n are
3: not connected with n
4: vi edges If the signs of the algebraic areas of triangles ÿk and ÿ with the same l then boundary edge of the superelement l are different for at least one k, go to the next boundary vertex If the edges of virtual triangles do not approximate
5: the original boundaries, then go to the next vertex Calculate local qualities Q0 = min{Q(ÿ1), ...
6:
7: ..., Q(ÿnÿ1)} and Q1 = min{Q(ÿ 1), ..., Q(ÿ nÿ3)} **if** Q1 > Q0
8: **then** Delete vertex
9: v from the grid. Replace triangles ÿ1, ÿ2, ..., ÿnÿ1 with triangles ÿ ÿ ÿ 2, ..., nÿ3. End algorithm **end if end loop** 12: **end loop**
10:
eleven:

and add new triangles. These elementary operations require the fast search algorithms within a dynamic list, described in § 2.3. For a list of Nf triangles, each elementary operation requires O(log2 Nf) arithmetic operations. The mesh quality is a monotonic function of the number of successfully implemented basic algorithms. Unfortunately, the quality of a particular triangle can fluctuate greatly. This is the main reason for the lack of more efficient search methods inside a dynamic

list.

Second, for each triangle ÿ, it is necessary to quickly find triangles in the superelement ÿ(ÿ). To do this, several optimal methods with arithmetic complexity O(1) are used. The simplest method is to build a structured nearest neighbor list U(ÿh) for each grid triangle and modify it after each basic algorithm. The initial construction of such a list requires O(Nf) arithmetic operations. A detailed description of the algorithm for filling the list U(ÿh) is presented in § 4.2. Each basic operation changes only a few triangles, so updating this list has optimal complexity. Search

triangles included in the superelement ÿ(ÿ) begins with the triangle ÿ. After that, we add their nearest neighbors from the list U(ÿh) to the already existing nonempty list of triangles, and then the neighbors of neighbors that have a common vertex with ÿ. This process is repeated as long as such neighbors exist. It is easy to see that this method has an optimal order of computational complexity if the number of triangles in the superelement ÿ(ÿ) is limited. Thirdly, the basic algorithms change the number of basic mesh objects: nodes, triangles, and boundary edges. These objects are represented by structured lists

(see § 2.3). Adding a new object to a structured list is the same as adding a new row to a two-dimensional array. Deleting an object involves shifting all objects below the deleted object by one line. This is a very time consuming operation and should be avoided. Instead, we will store an additional list of deleted objects, i.e. the numbers of the corresponding rows in a structured array. When adding a new object, we first check for free places in the structured list, and then fill them. If there are no free places, the new object is added to the end of the list. After the grid is rebuilt, empty places in the structured list are filled, for example, by renumbering all grid objects. The main stages of mesh rebuilding are presented in Algorithm 35. Let us pay attention to step 10 of this algorithm, which is performed when none of the basic algorithms could increase the quality of the triangle ÿ. The shift of the pointer by one position means that this triangle is temporarily excluded from the analysis (it is put aside for storage in the trash). During this time, the triangles in its vicinity may change. Moreover, their change can also affect the triangle ÿ itself. If this does not happen, then the algorithm will return to the triangle ÿ when the pointer is set

again to the worst triangle in the grid, i.e. k = 1.

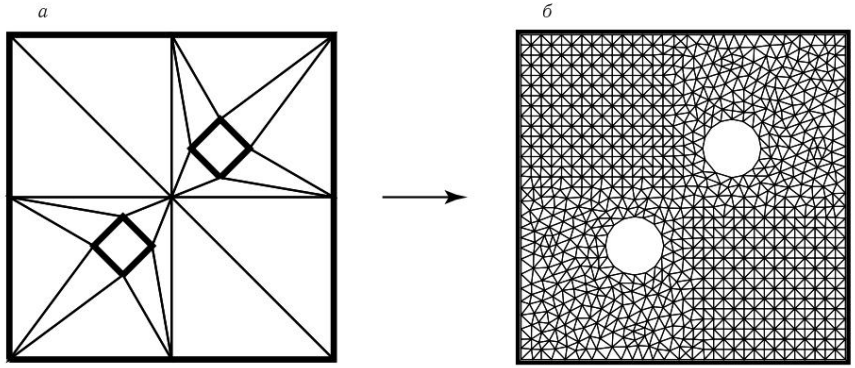
Algorithm 35 requires four parameters Q0, N1, N2 and N3 to be chosen. By definition, the desired final mesh quality Q0 1. Int. In Section 6, we consider several examples with different definitions of the quality Q(ÿ) and formulate the criteria for choosing Q0. The maximum allowable number of basic operations, N1, controls the computational complexity of the algorithm and plays an important role in the development of parallel meshing methods (see § 5.4). For a sequential algorithm, it suffices to choose a number N1, which is several times greater than the expected number of triangles in the final mesh. The maximum allowable size N2 of the basket of stored triangles depends on the number of triangles in the initial and final grids. The recommended value for N2 is approximately 5–10% of the average number of triangles in the grid. Maximum allowable

Algorithm 35. Mesh rebuilding

1: Calculate the qualities of the triangles and create a block-ordered list of them. Initialize helper structures. Choose the desired quality Q0 for the final mesh, the maximum allowable number of basic operations N1, the maximum allowable basket size N2, the maximum allowable number of baskets N3, set the pointer k = 1 and the counter m = 1

2: **loop** from 1 to N1
3: Take the k-th worst triangle ÿ from the list If k = 1 and Q(ÿ) Q0,
4: then terminate the algorithm Calculate the superelement ÿ(ÿ)
5:
6: Apply basic algorithms to ÿ (in no particular order)
until the first successful algorithm **if** a
7: successful algorithm is found **then** Modify the
8: mesh and update the list of triangle qualities and other auxiliary lists **else** Move the pointer k := k + 1. If k>N2 or k is greater
9: than
10: the number of triangles in the mesh, put k = 1 and m := m + 1 If m>N3, go to step 14 **end if** 13: **end loop** 14: Update structured lists of vertices, triangles, and boundary
eleven: edges
12:

the number of bins N3 allows the algorithm to terminate quickly when a further increase in mesh quality is impossible. For example, the geometric features of the computational domain (sharp corners, narrow sections, etc.) may impose restrictions on the quality of triangles. In this case, the baskets of triangles will contain the same triangles. We assume that the number of singular triangles is small and use the constant value N3. Note that N2N3 must be greater than the average number of triangles in the grid. At first glance, it seems strange that Algorithm 35 does not require, as an input, the desired number of triangles in the final mesh. In fact, we assume that this information is contained in the definition of the quality of the grid and the quality of the triangle, which imposes certain requirements on the formation of Q(ÿ). In other words, Q(ÿh) = 1 only if the mesh ÿh contains the desired number of triangles. Algorithm 35 is applicable for constructing computational grids based on a very coarse initial grid. For example, quasi-uniform



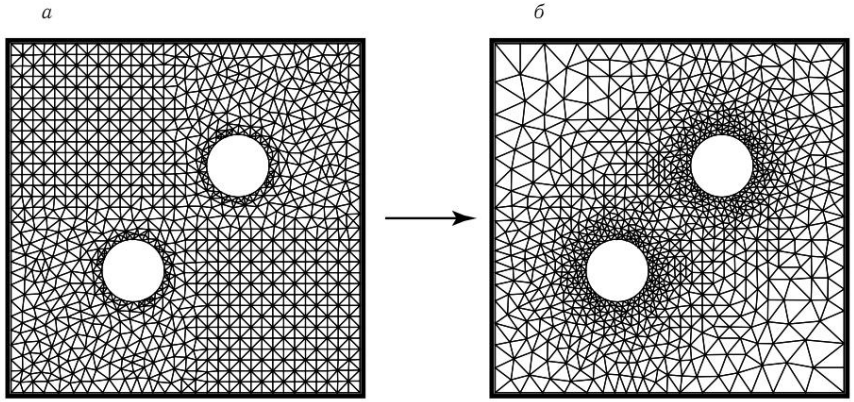
Rice. 5.9. Construction of a quasi-uniform grid based on a coarse initial grids

the grid shown in fig. 5.9, b, was built from the grid in fig. 5.9a, containing only 20 triangles. The initial mesh is a minimal representation of a square model with two round holes. Note a very rough initial approximation of the holes. Each of them is represented by four curvilinear ribs.

The parameter Q0 was chosen so that the quality of an isosceles right triangle was less than Q0. In a number of finite element methods, triangles with angles not exceeding 90ÿ make it possible to construct numerical schemes with additional properties, such as the discrete maximum principle.

Note that reliable rebuilding of computational grids requires all five basic algorithms. We will illustrate this statement with the example of the model shown in Fig. 5.9b . The initial quasi uniform mesh contains 1827 triangles. Let's try to rebuild it into a calculated regular grid with the same number of triangles,

which thickens to two holes. The quality of the triangle Q(ÿ) is determined in such a way that its diameter increases in direct proportion to the minimum distance from the centers of the holes. It would seem that this can be achieved by placing several additional nodes on the boundaries of the holes and shifting the rest of the grid vertices using the basic algorithm 32. Nevertheless, after 13,000 iterations of algorithm 35, the initial grid remained virtually unchanged. The final mesh is shown in fig. 5.10, a. If we add the basic algorithms 30, 33 and 34, which add a node and remove an edge or node, we get a mesh very similar to the mesh shown in Fig. 5.10, b. Algorithm 35 took 47,287 iterations to build this grid. As we expected, 97% of the iterations involved shifting the grid nodes. The quality of this mesh is Q(ÿh) = 0.73, and the average quality of the triangles is 0.88.



Rice. 5.10. Reconstruction of a quasi-uniform grid into a regular grid that thickens towards holes

Adding the last basic algorithm results in the grid shown in fig. 5.10, b. A detailed analysis of Algorithm 35 shows that the number of iterations has been approximately halved: to 22682, the mesh quality has increased to Q(ÿh) = 0.78, and the average quality of triangles has increased slightly: to 0.89.

Thus, the minimal ability to change the mesh topology allows it to be rebuilt in a reasonable number of operations. Extending the set of topological operations leads to faster convergence of the algorithm. Note also that a complete re-meshing requires several iterations per mesh element (12 in this example). The optimal order of the basic algorithms at step 6 of Algorithm 35 depends not only on the properties of the initial and final grids, but also on

the dynamics of the process. For example, rebuilding an anisotropic grid with triangles stretched in the horizontal direction into an anisotropic grid with triangles stretched in the vertical direction can occur through an intermediate quasi-uniform grid. At the beginning of the re-meshing process, the first basic algorithm will be executed much more often than other algorithms. Then the fourth and fifth basic algorithms will begin to dominate. Therefore, the optimal order of the underlying algorithms is practically impossible to determine in advance. The vertices of the square in fig. 5.10 are the singular points of the model. The grid nodes at the vertices of a square are usually set once and for all, and the underlying algorithms must preserve their position. One possible solution is to create an additional list of special mesh node properties and slightly modify the underlying algorithms. For example, Algorithm

32 should not move a grid node marked as fixed.

Determination of specific node properties, in addition to the data already given, can be automated if additional information about the model is available. For example, if the region boundary is represented by several curves, then it is reasonable to fix the grid nodes corresponding to the start and end points of these curves. If the boundary is defined as a set of edges, then the common vertex of two edges forming an acute angle can be marked as fixed. Special properties can also be introduced for boundary edges and triangles. Additional information about mesh objects allows you to effectively control the mesh rebuilding process. We will return to this issue in § 5.4. The estimation of the computational complexity of the algorithm 35 is made up of the estimations of the complexity of executing the basic operations (W1), the support of various data structures (W2), and searching

in dynamic unstructured lists (W3). The computational complexity of a single basic operation and the associated data structure update is independent of the grid size. Unfortunately, fast searching in unstructured lists depends on the size of the grid. Therefore, on very fine grids, one should expect that W3 W1 + W2. Consider a sequence of grids with an increasing number of triangles. Tab. Figure 5.1 shows the relative cost of one iteration of Algorithm 35 in constructing quasi-uniform meshes ÿh, starting from the very coarse mesh shown in fig. 5.9, a, as well as when these grids are rebuilt into regular grids ÿh, condensing to holes and similar to the

grid shown in Fig. 5.10. Recall that the dynamic list of triangle qualities is divided into blocks of length O(log2 Nf), and that an increase in the number of such blocks leads to an increase in the time of one operation with this list. The data presented in the table shows that W3 begins to noticeably dominate when the number of triangles exceeds 105.

Table 5.1

Relative average cost of one iteration of the algorithm 35

Quasi-uniform grids		Regular grids	
Nf (ÿh)	time	Nf (ÿ h)	time
930	1.00	986	1.00
11 916	0.88	9934	1.00
116 726	0.80	108 357	1.11
466 742	1.84	438,003	3.53
904 210	3.36	1,066,444	9.05

§ 5.3. Rearrangement of tetrahedrizations

The minimum set of data structures that define a tetrahedral mesh includes three structured lists for tetrahedra, boundary faces, and vertex coordinates. Building additional data structures based on this minimum information is discussed below.

Reliable rebuilding of tetrahedral meshes is a much more difficult problem than rebuilding triangular meshes. And the reason is not just the extra dimension. The set of fundamental results for triangulations does not extend to tetrahedralizations. For example, it is widely known that regular triangles of the same size cover the plane without intersections and voids. A similar statement for regular tetrahedra is false, although there are identically shaped tetrahedra for which the statement is true. There is also no simple local modification of the tetrahedral mesh for which an analogue of Theorem 5.2.1 on the construction of the Delaunay triangulation would be true. Even the Delaunay tetrahedrization can contain slivers (i.e., tetrahedra with dihedral angles close to 180°), which lead to various problems in the numerical solution of partial differential equations.

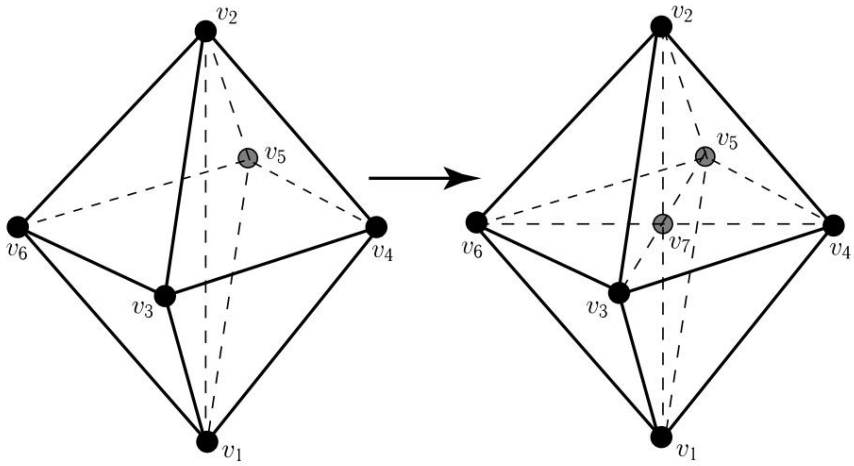
Most of the existing approaches to improve the properties of a tetrahedral mesh use a combination of different methods such as mesh smoothing, mesh topology modification, and mesh refinement.
These methods lead to various local modifications of the mesh, which we will call *basic operations.*

In § 5.2 we showed that expanding the set of basic operations has a positive effect on the convergence of the meshing method. For tetrahedral meshes, seven basic algorithms can be distinguished that change the mesh topology and the shape of tetrahedra. Consider a tetrahedron ÿ(v1, v2, v3, v4) (or ÿ1234 for simplicity of notation) with vertices v1, v2, v3, and v4, whose quality is to be increased. We present the structure of basic algorithms for a tetrahedron located inside a region. Extensions of the algorithms to the boundary tetrahedron will be discussed in the comments to the algorithms.

All basic operations change the mesh inside the superelement ÿ(ÿ1234) formed by tetrahedra that have at least one point in common with ÿ1234:

$$ÿ(ÿ1234) = \bigcup_{i=1}^{4} ÿ(vi).$$

The grid at the boundary ÿ(ÿ1234) does not change. As noted in § 5.1, this property is the key to developing parallel rebuilding methods.
grids.



Rice. 5.11. Placement of node v7 on edge e12

The first basic algorithm puts the node in the middle of the edge of the tetrahedron ÿ and splits the tetrahedra adjacent to the edge in half. The algorithm is similar to algorithm 30 for a triangle. Since the dihedral angle of a tetrahedron is less than 180°, the number of tetrahedra with a common edge is greater than or equal to three. Rice. 5.11 illustrates the operation of this algorithm for an internal mesh edge that is common to the four tetrahedra ÿ1234, ÿ1245, ÿ1256, and ÿ1263.

Algorithm 36. Placement of a node on an edge of a tetrahedron

1: **loop** over all edges e of the tetrahedron
ÿ 2: Find tetrahedra ÿ1, ..., ÿn with a common edge e, as shown in fig. 5.1
and put ÿ1 ÿ ÿ. Determine Q0 = min{Q(ÿ1), ..., Q(ÿn)} Put a test node v7 in the middle of the edge e
3: and split and ÿb each tetrahedron ÿi into two virtual tetrahedra ÿa), 1), .. ., Q(ÿa n), Q(ÿb n)} Define Q1 = min{Q(ÿa **if** Q1 > Q0 **then** Q(ÿb i .
replace each of the tetrahedrons Add node v7 to the mesh and
4: and ÿb ditch ÿi
5: with two tetrahedra ÿa i . Finish Algorithm 6: **end if** 7: **end loop**
i

The algorithm can be easily generalized to the case of a tetrahedron with one or more edges lying on the boundary of the region. Additional analysis is required only for edges lying on curvilinear boundaries. We will call such edges *curvilinear edges.* Note that additional information about curvilinear boundaries

can be available if the computational domain model is specified using CAD. CAD libraries allow you to work effectively and reliably with curved boundaries and should be used whenever possible.
use.

Let us assume that each point of a curvilinear edge is mapped to a single point of the curvilinear boundary, and the function F(x) describes this mapping. The coordinates of the midpoint of the edge e12 and the point projected onto the curvilinear boundary are calculated as follows:

$$v7 = (v1 + v2)/2, \qquad v7 = F(v7).$$

As with triangular meshes, node projection can lead to mesh entanglement. To check the correctness of the mesh, we use an analogue of Lemma 5.2.1. Let us consider the superelement ÿ(e12) formed by tetrahedra with a common edge e12 and select those faces that lie on the surface of this superelement and do not contain the vertex v7. Each of these faces defines a plane that divides the space into two half-spaces. Let ÿi be half-spaces containing the vertex v7. We define the set D as the intersection of the half-spaces ÿi. **Lemma 5.3.1.** *The projection* v7 = F(v7) *leads to a regular grid if* v7 ÿ *D.* The proof of this lemma repeats the arguments of Lemma

5.2.1 and is therefore not given. An important consequence of the proof concerns the

sign of the algebraic volume of the tetrahedron with vertex v7. The algebraic volume of a tetrahedron ÿ1347 is calculated by the formula

$$V_{ÿ1347} = \frac{1}{6}\det\{v3 ÿ v1, v4 ÿ v1, v7 ÿ v1\}, \qquad (5.3.1)$$

where det is the determinant of a 3 × 3 matrix with columns v3 ÿ v1, v4 ÿ v1, and v7 ÿ v1. Consider auxiliary (virtual) tetrahedra constructed using v7 instead of v7. Following the notation introduced by ÿb in Algorithm 36, we denote these tetrahedra by ÿ a The following analog of Lemma 5.2.2 is correct. i *and* ÿb i **Lemma 5.3.2.** *Let* ÿ a

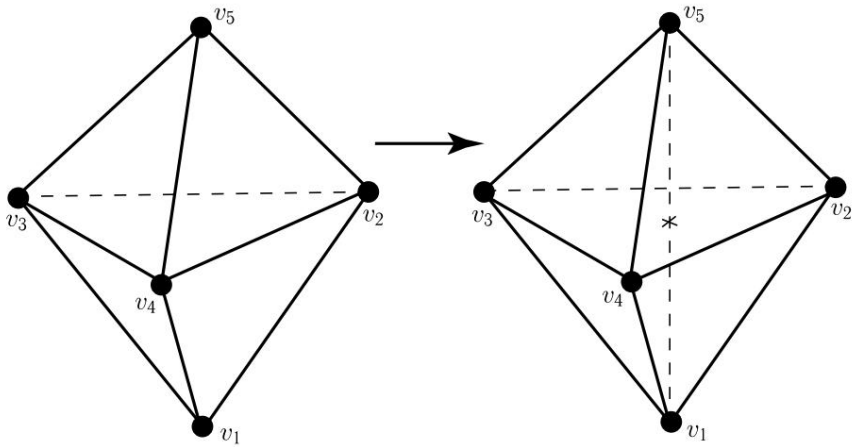
i i *are auxiliary tetrahedra built using* v7 *instead of* v7. *The grid remains correct if the algebraic volumes of the tetrahedra* ÿa *and* ÿa *(similarly for the tetrahedra* ÿb *and* ÿb) *have the same sign.* Let us give another interpretation of formula (5.3.1):

$$V_{ÿ1347} = \frac{1}{6}((v3 ÿ v1) \times (v4 ÿ v1))(v7 ÿ v1) = a(v7 ÿ v1).$$

Many algebraic volumes are calculated for two positions of the tetrahedron vertex. For example, for the vertex v7, an efficient implementation of these calculations is based on the selection of the vector a, which does not depend on the position of this vertex and can be calculated only once.

In computer calculations, the determination of the sign of the algebraic volume depends on roundoff errors. Vgl. 3, we proposed two methods for solving this problem: (a) computing the determinant (5.3.1) with extra precision, and (b) introducing a function d(v1, v2, v3, v4) that returns zero when the sign of the determinant cannot be defined exactly. We recommend using one of these methods whenever you need to find the sign of an algebraic volume. Note that the zero value of the function d(v1, v2, v3, v4) leads to errors of the first kind, which are not critical for rebuilding the grid and only limit the number of allowed basic operations. *The second basic algorithm* is applied to a pair of tetrahedra that have a common face. If the union of these tetrahedra

forms a convex polyhedron in which no four points lie in the same plane, then there is another partition of this polyhedron into three tetrahedra (see Fig. 5.12). Algorithm 37 determines which of the beats improves the quality of the mesh. Unlike the 2D flip algorithm, the number of elements in the grid is increased by 1.



Rice. 5.12. Replacing face f234 with edge e15

Algorithm 37 can be easily generalized to the case of a tetrahedron ÿ with faces lying on the inner boundaries. Since such faces cannot be removed from the mesh, the algorithm does not need to consider them.

The third basic algorithm is applied to a triple of tetrahedra that have a common edge. This algorithm is the inverse of the second basic algorithm, i.e., successive application of these algorithms returns the grid to its original position. If the union of three tetrahedra forms a convex polyhedron, then there is another partition of this polyhedron into two tetrahedra (see Fig. 5.12).

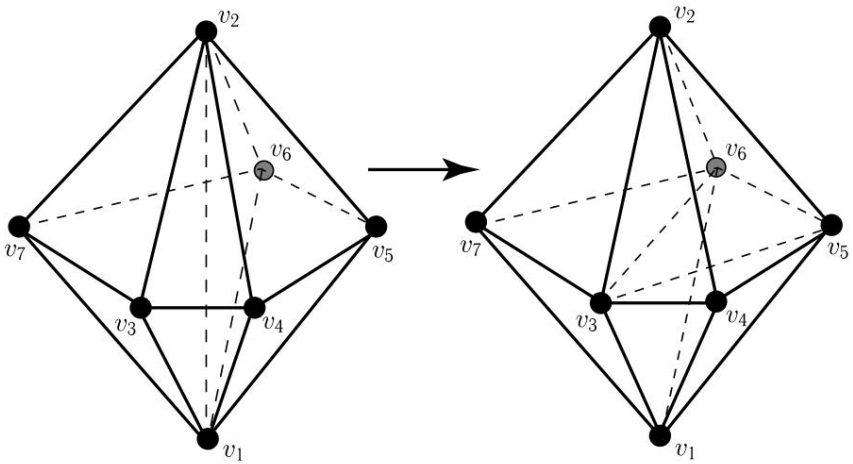
Note that this algorithm is also an analogue of the two-dimensional algorithm 31, but reduces the number of elements in the grid by one. So

Algorithm 37. Replacing a face with an edge

1: **loop** over all faces f of tetrahedron ÿ 2: Find
neighboring tetrahedron ÿ with face f and vertices v2, v3, v4, v5, as shown in fig. 5.12. If the
tetrahedra ÿ and ÿ do not form a strictly convex polyhedron, then go to the next face
Determine Q0 = min{Q(ÿ), Q(ÿ)} Calculate the qualities of tetrahedra ÿ1425, ÿ1235 and
ÿ1345. Determine Q1 =
3: min{Q(ÿ1425), Q(ÿ1235), Q(ÿ1345)} **if** Q1 > Q0 **then**
4: Replace ÿ and ÿ tetrahedra with ÿ1425, ÿ1235 and ÿ1345 tetrahedra. End algorithm 7:
end if 8: **end loop**
5:
6:

Thus, there are two flip operations in space . We do not present the formal structure of the third basic algorithm due to its obviousness. In what follows, referring to Algorithm 37, we will mean both flip operations. *The fourth basic algorithm* is a generalization of the third basic algorithm for the case of a larger number of tetrahedra with a common edge.

To illustrate, we will use the configuration shown in Fig. 5.13. The edge e12 is replaced by a triangulated polygonal face P with vertices v3, v4, v5, v6 and v7. We note that the vertices of this polygon usually do not lie in the same plane, and one can speak of its convexity only in the sense of the convexity of its projection onto the plane that deviates least from its vertices.



Rice. 5.13. Replacing the edge e12 with a triangulated polygonal face f34567

Algorithm 38. Replacing an edge with a polygonal face

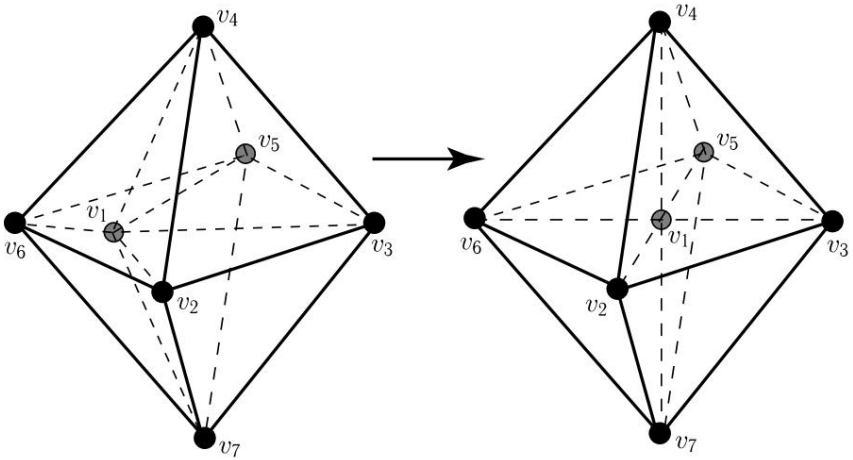
1: **loop** over all edges e of the tetrahedron ÿ 2: Find
tetrahedra ÿ1, ..., ÿnÿ2 with a common edge e, as shown in fig. 5.13 where e ÿ e12; put ÿ1 ÿ ÿ.
Determine Q0 = min{Q(ÿ1), ..., Q(ÿnÿ2)} **loop** i = 3, ... , Split polygon P with vertices v3,
v4, ..., vn (n = 7 in Fig. 5.13) into triangles f3, f4, ..., fnÿ2,
3: connecting vertex vi n
4: with other vertices not connected to vi by an edge and ÿb Construct tetrahedra ÿa
with a common base fk and distinct vertices v1 and v2, where k = 3, ... , n ÿ 2), Q(ÿb
3), ..., Q(ÿa nÿ2), Determine Q1 = min{Q(ÿa Q(ÿb nÿ2)} if Q1 > **Q0** then **Replace**
tetrahedra
5: where k = 3, ..., nÿ2. End the algorithm **end if** 10: **end loop** 11: **end loop**
ÿ1, ..., ÿnÿ2 tetrahedra ÿa and ÿb k
6:
3
7:
8: k
9:

Algorithm 38 considers only (n ÿ 2) partitions of the polygon P into triangles fk, although the number of possible partitions can reach (n ÿ 2)! for a convex polygon. Such a large number of possible partitions makes their exhaustive enumeration impractical for n > 8. In the numerical experiments presented below, Algorithm 38 is always executed, but the number of partitions considered is limited to three hundred. There is no unequivocal opinion among specialists about the need for a fourth basic algorithm for the completeness of the set of topological operations with a grid. We note the works [49, 60, 72], where,

based on the analysis of a large set of meshes, it was shown that the addition of operations with polygonal faces can significantly improve the quality of tunable meshes. The experiment at the end of this section confirms the results of these works, although the additional increase in mesh quality is negligible. *The fifth basic algorithm* changes the position of the grid node. Let v1 be one of the vertices of the tetrahedron ÿ. Consider a superelement ÿ(v1) formed by tetrahedra with a common vertex v1. On fig. 5.14 the node v1 is shifted to the geometric center of the superelement ÿ(v1). This approach is used in methods for improving the shape of tetrahedra. Unlike its two-dimensional counterpart, it often results in poorly shaped tetrahedra. In

general, the optimal position of the vertex v1 should increase the quality of the worst tetrahedron in the superelement ÿ(v1). Quality

Q(ÿ(v1)) is the nonlinear functional of the coordinates of the vertex v1; therefore, as in the case of triangular grids, the search for the optimal position of this vertex requires the use of minimization (or maximization) methods for nonconvex functionals.



Rice. 5.14. Local modification of the mesh by moving node v1

Let v1 = (x1, y1, z1) and, for simplicity, Qÿ = Q(ÿ(v1)). The approximate gradient Qÿ is calculated using finite difference discretization:

$$ÿhQÿ = \left(\frac{Qÿ(x1 + ÿx, y1, z1)}{ÿx}, \frac{Qÿ(x1, y1 + ÿy, z1)}{ÿy}, \frac{Qÿ(x1, y1, z1 + ÿz)}{ÿz} \right)^T.$$

A reasonable value for the increments ÿx, ÿy, and ÿz is the square root of machine precision. For the reliability of the algorithm, it is necessary to check that the calculated increment is significantly less than the diameter of the superelement, for example, less than 1% of the length of the minimum edge:

$$ÿx = ÿy = ÿz = \min\{ ÿ, \min_{vjÿÿ(v1)} |vi ÿ vj|/100\}. vi,$$

The node v1 is shifted along the approximate gradient until the maximum Qÿ is reached:

$$v1 := v1 + ÿ\, ÿhQÿ, ÿ\, 0.$$

Note that a simple replacement of Qÿ by 1 ÿ Qÿ makes it possible to apply standard methods for minimizing functionals without modifying them. Since Qÿ is not a quadratic functional, the gradient descent method (in our case, the gradient ascent method) usually does not lead to the calculation, even approximate, of the local maximum of the functional on the superelement ÿ(v1). We do not recommend

cyclic repetition of the procedure described above, as is done in standard methods for minimizing (or maximizing) functionals. Rebuilding the grid can repeatedly change any of the tetrahedra, so there is a high probability that the configuration of the superelement ÿ(v1) will change rapidly. Algorithm 39 is used to find the parameter ÿ, which can use the bisection method or various versions of the Levenberg–Marquard method.

Algorithm 39. Node shift

1: **loop** over all vertices v1 of the tetrahedron ÿ 2:
Find the tetrahedra ÿ1, ÿ2, ..., ÿn that form the super element ÿ(v1); put ÿ=ÿ1, as shown in fig. 5.14, where n = 8 Calculate the approximate gradient ÿhQÿ Calculate
3: the maximum possible displacement of the vertex
4: v1 in the direction ÿhQÿ that does not violate the topology of the superelement. Let ÿ = ÿmax for this extreme position 5: Find the value of ÿ in the half-open interval [0, ÿmax) that maximizes Qÿ 6: If ÿ > 0, then store the new position of v1

in the grid and finish algorithm 7:
end loop

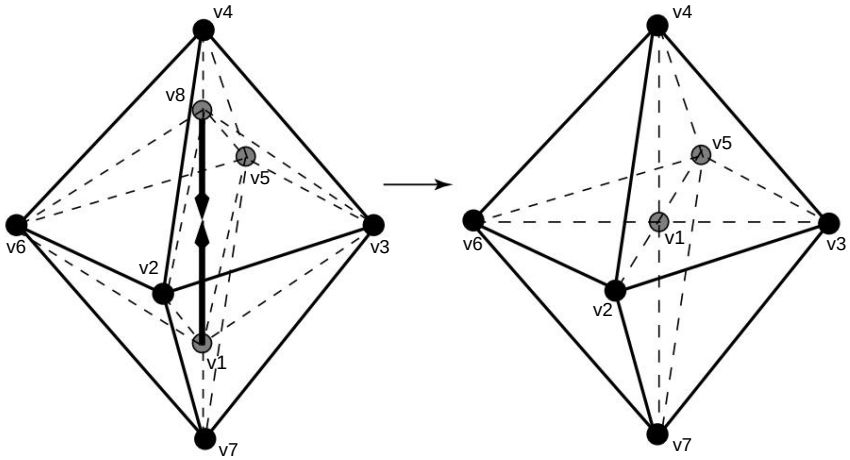
Note that the node v1 cannot go beyond the superelement, since the quality Qÿ tends to zero when v1 approaches its boundary. However, moving a node even inside a superelement can confuse the grid when that superelement is not convex. The analysis of sufficient conditions for mesh correctness is similar to the analysis developed for triangular meshes. For an internal grid node v1, the admissible set D of its positions is defined as the intersection of the half-spaces ÿ1, ..., ÿn. The half-space is defined by one of the boundary faces of the superelement ÿ(v1) and contains the initial node v1. Note that the set D coincides with the superelement ÿ(v1) only when it is convex.

Another method for checking the correctness of the mesh is to calculate the algebraic volumes of tetrahedra ÿ1, ..., ÿn for each new position of their vertex v1. The mesh remains correct if these volumes retain their sign when the v1 vertex is shifted.

Algorithm 39 can be easily generalized to the case of a tetrahedron with vertices lying on the inner and outer boundaries. Such vertices can only move along boundary faces. Movement along curvilinear faces must be accompanied by checking the correctness of the grid. As before, for this it suffices to control the conservation of the sign of the algebraic volumes of the tetrahedra ÿ1, ..., ÿn. Likewise,

the movement of a node lying on an edge that separates the boundary faces, or an edge of a CAD model, must occur only along this edge. *The sixth basic algorithm* removes an edge from the mesh. The algorithm can be

interpreted as follows. We begin to move the vertices of the edge towards each other, possibly at different speeds. When vertices merge into one, the topology of the mesh changes. As shown in fig. 5.15, when vertices v1 and v8 merge, each of the four pairs of edges merges into one edge, and four tetrahedra with a common edge e18 disappear from the grid. In Valgorithm 40, we use the notation shown in Fig. 5.15.



Rice. 5.15. Local mesh modification by merging v1 and v8 vertices

Algorithm 40 can be easily generalized to the case of a boundary edge lying inside a flat section of the model boundary. An edge from one of the vertices lying on the model edge is removed from the mesh by shifting the opposite vertex to this vertex. Removing a mesh edge that lies completely on a model edge, i.e., at the intersection of two flat sections of the boundary, requires checking that this is not the only edge that represents an important part of the geometric model.

Step 4 of algorithm 40 requires choosing a common vertex v1 for vertices v1 and v8. On the one hand, it is natural to consider the problem of the optimal position of the vertex v1. On the other hand, since a similar optimization problem is considered in the basic vertex shift algorithm, the complication of algorithm 40 is not required. The optimal position of the v1 vertex can be achieved by sequentially applying two basic algorithms. That's why we

Algorithm 40. Removing an edge

1: **loop** over all edges e of the tetrahedron ÿ 2: Find the tetrahedra ÿ1, ÿ2, ..., ÿn that form the super element ÿ(e), and set ÿ=ÿ1 as shown in fig. 5.15 where n = 4 3: Find the tetrahedra ÿ ÿ that are: (a) different from the tetrahedra in the superelement ÿ(e), (b) are included in the superelement ÿ(v1) or ÿ(v8) Determine the new common (virtual) position v1 vertices 1 v1 and v8:v1= $\frac{1}{2}$ (v1 + v8) 5: Define new (virtual) tetrahedra ÿ by shifting vertices v1 and v8 of the original tetrahedra into v1 and ÿ 6: If the algebraic volumes of tetrahedra ÿ have different signs for at least one i, then delete all virtual objects and go to the next edge Determine the qualities Q0 = min{Q(ÿ1), Q(ÿ2), ..., Q(ÿn)} and Q1 = 7: min{Q(ÿ 1), Q(ÿ 2), ..., Q(ÿ m)} **if** Q1 > Q0 **then** Delete vertex v8 and tetrahedra ÿ1, ..., ÿn from the grid; replace vertex v1 with v1 and tetrahedra ÿ with ÿ End 8: algorithm 10: **end if** 11: 9: **end loop**

we propose to consider only one possible position of the nova (v1 + v8). The exception is the cases when region. In this case, $\frac{1}{2}$ one vertex: v1 = from the vertices of the edge e18 lies on the boundary of the the vertex lying inside the region is shifted to the boundary vertex.

The generalization of Algorithm 40 for curvilinear boundaries requires the projection of the vertex v1 obtained at step 4 of the algorithm onto the curvilinear boundary. Like any other vertex displacement, the projection must fall within the allowable set of shifts for which the mesh does not get tangled. A sufficient condition for maintaining the correctness of the grid is to check the signs of the algebraic volumes at step 6 of the algorithm. An estimate of the local curvature of the boundary can also be used to determine if a curvilinear edge can be removed. *The seventh basic algorithm* removes a node from the grid along with all tetrahedra containing it. The void created in the grid is filled with fewer tetrahedra. To construct these tetrahedra, Algorithm 41 uses only existing mesh nodes.

Note that checking the signs of algebraic volumes is sufficient for a node v lying inside the domain ÿh and outside the internal boundaries

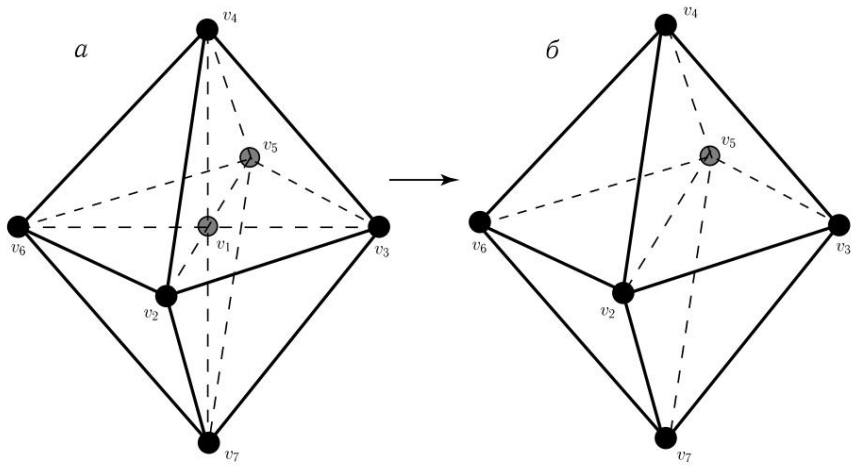
Algorithm 41. Deleting a node

1: **loop** over all vertices v of the tetrahedron ÿ 2: Find the tetrahedra ÿ1, ÿ2, ..., ÿn that form the superelement ÿ(v) and put ÿ=ÿ1. Count the number of vertices Mv on the boundary of the superelement
3: **loop** i = 1, ... , Mv Split the
4: superelement ÿ(v) into virtual tetrahedra ÿ ÿ connecting the boundary vertex vi with the other 1 algebraic ..., boundary vertices not connected to vi by edges If the signs of the volumes of the tetrahedra ÿk and ÿ
5: I with a common boundary face of the superelement ÿ(v) are different for at least one k, then go to the next boundary vertex If the faces of virtual tetrahedra do not approximate
6: the interior and boundary surfaces, go to the next vertex Determine the local qualities ...
7: ..., Q(ÿn)} and Q1 = min{Q(ÿ 1), ..., Q(ÿ m)} **if** Q1 > Q0 **then**
8: Delete node v from the
9: grid, replace tetrahedra ÿ1, ..., ÿn tetrahedra ÿ Finish the algorithm **end if end loop** 12: **end loop** 1, ..., ÿ m.
10:
eleven:

(e.g. material boundaries). In this case, the superelement ÿ(v) is j is a polyhedron.

Otherwise, several additional checks are required for the approximation of the outer and inner boundaries by the faces of virtual tetrahedra. For flat boundaries, it suffices to check that all these boundaries are preserved with machine accuracy. For curved boundaries, you need to make sure that the topology of the new discrete boundaries matches the topology of the original boundaries. For example, suppose that the faces f124, f145, f157, and f172 in Fig. 5.16, a, approximate part of the curvilinear boundary of the geometric model. The topology of this boundary is preserved for one of two pairs of new faces: f245, f257 or f247, f457. As in two dimensions, analysis of the topology of a boundary is made easier by treating the basic node removal operation as a special case of the more general edge removal operation.

The seven basic algorithms described above are most often used in methods for improving mesh quality [86]. We also note a few additional mesh operations that appear in the literature. To improve the shape of tetrahedra, the authors of [60] propose to add Steiner nodes to the mesh using enough



Rice. 5.16. Local modification of the mesh by deleting a node v1

a complex algorithm whose generalization to an arbitrary quality Q(ÿ) is not obvious. The operation inverse to the basic algorithm 40 is considered in [49]. It can be shown that it reduces to a combination of flip algorithms if none of them reduces the mesh quality. The operation of rebuilding the triangulation of the polygonal face f34567 (see Fig. 5.13) is considered in [72] and is a combination of the basic algorithm 38 and its inverse if none of these operations reduces the mesh quality. The combination of basic algorithms opens up wide possibilities for constructing or rebuilding computational grids. Let's consider what data structures are required for efficient implementation of basic algorithms. Note that these structures will be similar to the data structures used to rebuild triangular meshes.

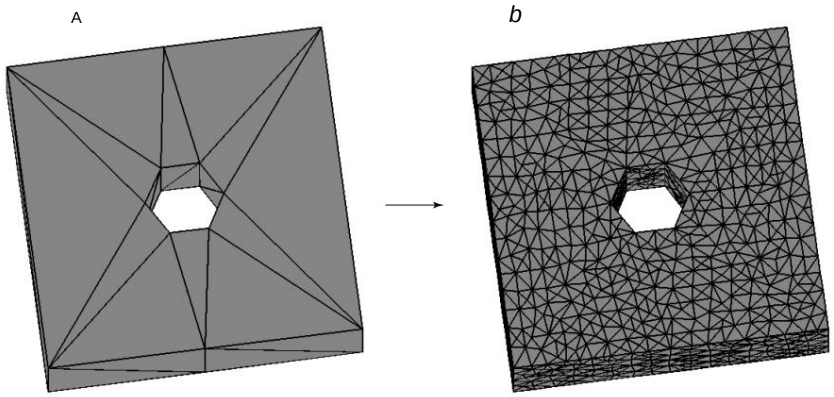
First, as in the case of triangular meshes, it is necessary to maintain a list of tetrahedra ordered in ascending order of their quality. As in the two-dimensional case (see § 5.2), we will use a block structured list and the algorithms for modifying it, described in § 2.3. Second, for each vertex v of the tetrahedron ÿ, it is necessary to quickly find tetrahedra in the superelement ÿ(ÿ). To do this, several optimal methods with arithmetic complexity O(1) are used. The simplest method is to build a structured list of nearest neighbors for each tetrahedron of the grid U(ÿh) and modify it after each basic algorithm. The number of such neighbors is at most four. The initial construction of such an ordered list requires O(Nt) arithmetic, where Nt is the number of tetrahedra in the grid, and is discussed in detail in § 4.3. Each basic operation modifies only a few tetrahedra, so updating this list

has optimal difficulty. The construction of the superelement ÿ(ÿ) is based on the list U(ÿh) and begins with ÿ. After that, we add to the list its nearest neighbors, and then the neighbors of neighbors that have a common vertex with ÿ. This process is repeated as long as such neighbors exist.

Thirdly, the basic algorithms change the number of basic mesh objects: nodes, tetrahedra, and boundary faces. These objects are represented by structured lists (see § 2.3). Adding a new object to a structured list is the same as adding a new row to a two-dimensional array. Deleting an object involves shifting all objects below the deleted object by one line, which is an expensive operation. Instead, we will store additional lists of removed nodes, boundary faces, and tetrahedra, i.e., the numbers of the corresponding rows in structured arrays. When adding a new object, we first check for free places in the structured list and fill them. After rebuilding the grid, empty spaces in the structured list are filled, for example, by moving grid objects from the end of the list to empty spaces. Rebuilding the grid is carried out by algorithm 35, described in § 5.2. Since the adaptation of this algorithm to tetrahedral meshes consists in replacing triangles with tetrahedra and edges with faces, we will not repeat the structure of this algorithm. In what follows, we will refer to Algorithm 35 as an algorithm for rebuilding tetrahedral meshes.

Algorithm 35 is applicable for constructing computational grids based on a very coarse initial grid. For example, the mesh shown in Fig. 5.17a contains 24 nodes and 36 tetrahedra. Note that the initial mesh contains only four more nodes than is required for the minimal representation of a parallelepiped model with a hole in the form of a regular hexagonal prism. The parameter Q0 = 0.81 was chosen so that the quality of the canonical tetrahedron was less than Q0. In a number of finite element methods, tetrahedra with dihedral angles not exceeding 90° make it possible to construct numerical schemes with additional properties, such as the discrete maximum principle.

Note that Algorithm 35 does not guarantee meshing with high quality Q(ÿh), since Q(ÿh) is always equal to the quality of the worst tetrahedron. The geometry of the model, in particular the presence of sharp dihedral angles, imposes a limitation on the maximum quality of near-boundary tetrahedra and hence on the mesh quality that can be achieved. Nevertheless, the average quality of tetrahedra can be rather high. Note also that the given set of basic algorithms does not guarantee that any original mesh can be rebuilt into any other mesh of higher quality.



Rice. 5.17. Construction of a quasi-uniform grid based on a coarse initial grids

by local changes in the mesh topology, which constantly increase its quality. Despite the lack of theoretical results, in practice Algorithm 35 makes it possible to successfully build grids close to optimal for efficient approximate solution of partial differential equations.

Returning to the grid shown in Fig. 5.17b, we note that its quality is Q(ÿh) = 0.45, while the average quality of tetrahedra is 0.67. Although Q(ÿh) is much smaller than Q0, nevertheless, the maximum dihedral angle is 138ÿ and the minimum is 22ÿ, which is an acceptable result for tetrahedral mesh generators. The second line in the table. 5.2 shows the quality distribution of tetrahedra in this casi-uniform mesh. Note that only 28 tetrahedra have a quality in the range (0.4, 0.5].

T a b l e 5.2

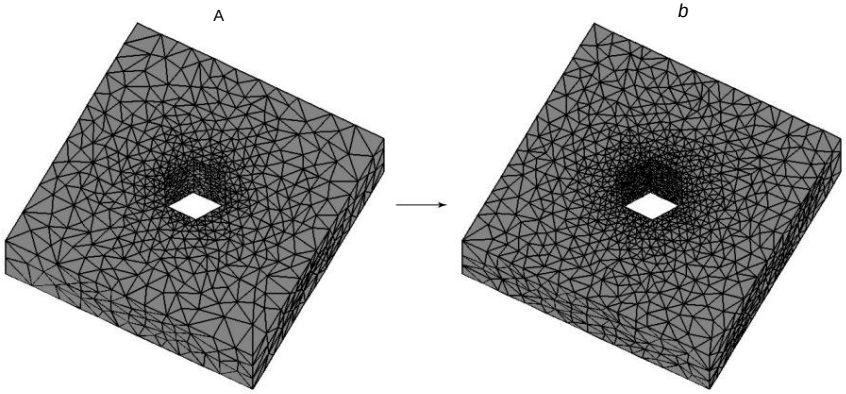
Distribution of tetrahedra by quality

Net \ Quality	0.4	0.5	0.6	0.7	0.8	0.9	1.0
quasi-uniform	0	28	7002	8285	5998	2761	450
regular-1	0	0	3416	9455	6854	3244	506
regular-2	0	5	8389	17 316	13 027	6018	1086

We now rebuild the quasi-uniform grid into a regular grid with approximately the same number of tetrahedra, which thickens towards the hole. The quality of a tetrahedron is determined in such a way that its size increases in direct proportion to the square of the distance to the central axis of the hole. The trace of the grid on the surface of the region is shown in Fig. 5.18, a. The quality of the reconstructed grid Q(ÿh) = 0.53 with an average

as tetrahedra 0.70. The maximum dihedral angle is 135ÿ and the minimum is 24ÿ. Thus, rebuilding the grid made it possible not only to change the spatial distribution of tetrahedra, but also to preserve important properties of the grid. This is confirmed by Table. 5.2, in the third line of which the distribution of tetrahedra by quality is presented.

To show the reliability of the mesh rebuilding algorithm, we will rebuild the quasi-uniform mesh into a mesh that not only thickens towards the hole, but also contains twice as many tetrahedra. The trace of the grid on the surface of the region is shown in Fig. 5.18b . The quality of the reconstructed mesh is Q(ÿh) = 0.49, while the average quality of tetrahedra is 0.70. The maximum dihedral angle is 136ÿ and the minimum is 23ÿ. The last line in the table. 5.2 shows an approximately twofold increase in the number of tetrahedra in each quality range. Thus, both regular grids have approximately the same properties.



Rice. 5.18. Construction of regular grids based on a quasi-uniform grid

Note that the smoothness of the surface mesh is less than that of the meshes constructed by the advanced edge method. The smoothness of the mesh can lead to superconvergence of finite element solutions if the differential solution is sufficiently smooth. In a number of elasticity problems, especially in problems with cracks, the main error is concentrated near the cracks, where the solution is not smooth enough, so that there is no superconvergence effect. In such applications with a non-smooth solution, the visual smoothness of the mesh does not lead to a significant improvement in the accuracy of the finite element solutions. We conclude this section with one useful practical tip for remeshing in domains with curvilinear boundaries. If the analytical representation of these boundaries is unknown, then we recommend fixing the grid nodes

at these boundaries once and for all. In this case, the rebuilt mesh will approximate curvilinear boundaries with the same order as the original mesh.

Moreover, the computer implementation of the basic algorithms is greatly simplified. A more rigorous approach is to locally reconstruct the curvilinear boundary and is discussed in the appendix.

§ 5.4. 3D Algorithm Parallelization

The parallelization of the algorithm 35 is based on its locality. The grid can be rearranged simultaneously in several places provided that any pair of reconfigurable superelements ÿ(ÿ) and ÿ(ÿ) does not have common tetrahedra. As a rule, the rebuilding of triangular grids does not lead to excessive computational work and can be performed on a single processor computer. Therefore, we will focus on rebuilding tetrahedral meshes. The simplest model of a parallel algorithm consists of a main processor (master), which distributes work among other processors (slave), sending them superelements ÿ(ÿ) and including the results of their work

in the grid. Due to the large number of data structures that are associated with the superelement ÿ(ÿ) and are necessary for its rebuilding, the time for sending and receiving packets can exceed the time for rebuilding one superelement. Therefore, this model is designed for a small number of processors. Let P be the number of processors in a parallel computer. The proposed model of the parallel algorithm is based on the partition ÿ(i) i = 1, ... , P, with the number of tetrahedra approximately equal to ÿ(i). An important requirement for the set to be nimization of its boundary, or rather, the number of tetrahedra that have one ÿ(i) The exception is those boundaries of vertices that lie on the boundary of the set, which are also the boundaries of the

original mesh. The application of basic operations to the tetrahedron ÿ lying in the boundary ÿ(i) h requires data exchange with neighboring sets. Therefore, reducing the number of such tetrahedra, grids into connected sets called *interface tetrahedra,* will make it possible to construct a more efficient parallel algorithm. Several methods are known h is in our for partitioning the grid ÿh into P approximately equal parts. The most popular methods include various method [48, 75] is based on the spectral variants of the bisection method. The spectral bisection properties of the grid graph, such as the eigenvector corresponding to the second eigenvalue of the Laplace grid operator. The need to solve the eigenvalue problem multiple times in the course of set layer remeshing makes this method too expensive for our parallel algorithm. The inertial bisection method [85] is much cheaper in terms of computational work, but generates a larger number of interface tetrahedra. Nevertheless, in view of the possibility of a strong increase or decrease in the

number of tetrahedra on any processor during

mesh generation, optimization of the number of interface tetrahedra can be sacrificed to minimize computational work.

The inertial bisection method uses only the coordinates of the grid nodes to calculate the inertia tensor, followed by grid splitting along its main axes. The computational complexity of this method is proportional to the number of tetrahedra Nt(ÿh).

Let us consider a body consisting of unit point masses located at the geometric centers of the tetrahedra of the grid ÿh. The inertia tensor of this body is defined as follows:

$$T(\ddot{y}h) = \sum_{i=1}^{Nt(\ddot{y}h)} \left(pi^2 I_3 - pi\, pt_i \right), \qquad pi = x\,\ddot{y}i - \overline{x\,\ddot{y}h}, \tag{5.4.1}$$

where x ÿi ÿ R3 denotes the geometric center of the tetrahedron ÿi and $\overline{x\,\ddot{y}h}$ is the grid center:

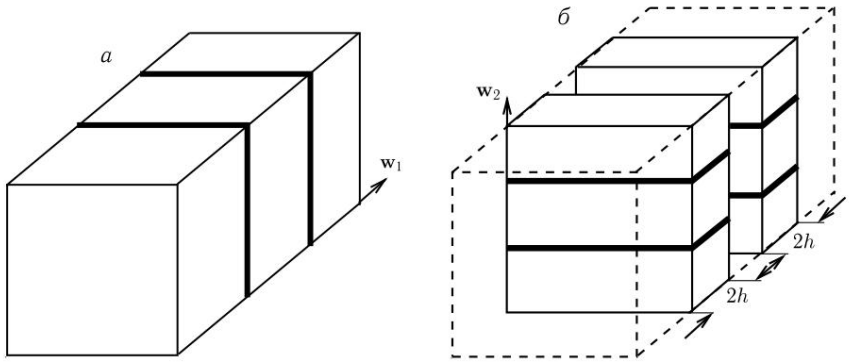
$$\overline{x\,\ddot{y}h} = \frac{1}{Nt(\ddot{y}h)} \sum_{i=1}^{Nt(\ddot{y}h)} x\,\ddot{y}i .$$

Each of the terms in the sum (5.4.1) is a positive semidefinite 3 × 3 matrix. Therefore, the tensor T(ÿh) is positive definite for a system of four or more points not lying in the same plane. Due to the smallness of the order T(ÿh), the calculation of the eigenvectors of the inertia tensor is a cheap operation.

Using the inertia tensor, we can divide the grid into subgrids with approximately the same number of elements by planes orthogonal to one of the eigenvectors of this tensor (see Fig. 5.19). To avoid creating too thin subdomains, we recommend using a small number of splits. Consider a simplified implementation of the inertial bisection method. This implementation is efficient when the number of processors is

small, but each processor has enough RAM to contain minimal information about the entire mesh. Simple calculations show that the storage of a tetrahedral mesh containing 106 elements requires about 34 MB of RAM, which is hundreds of times less than the amount of memory in modern computers. A small number of processors allows meshing only along one direction. After partitioning the grid into P approximately equal subgrids, the grid rebuilding algorithm is applied independently to each subgrid. To preserve the conformality of the global grid, certain restrictions are imposed on operations with tetrahedra at the boundaries between subgrids: admissible operations must not change the grid at these boundaries. The arithmetic complexity of mesh

decomposition (step 3 in Algorithm 42) and conformal mesh assembly (step 5) is proportional to Nt(ÿh). Numerical experiments confirm that these steps do not take



Rice. 5.19. Partition of the computational domain into subdomains by planes orthogonal to the first *(a)* and second *(b)* eigenvectors of the inertia tensor

Algorithm 42. Parallel reconstruction of the tetrahedral mesh

1: Calculate the inertia tensor T(ÿh) on a processor with rank 0 2: **loop** k = 1, 2, 3:
3: A processor with rank 0 selects tetrahedra whose quality is less than Q0, as well as all their neighbors. The set of selected tetrahedra is divided into P approximately equal parts using the inertial bisection method for the kth principal axis of the tensor T(ÿh) and distributed over all processors. A processor with rank m, m = 0, ... , P ÿ 1, rearranges its subgrid in this way that borders
4: with neighboring subgrids
do not change
5: A processor with rank 0 collects subgrids from other processors and builds a new conformal global grid ÿh. If Q(ÿh) Q0, then the algorithm terminates 6: **end loop** 7: The processor with rank 0 allocates tetrahedra whose
vertices
belonged to one of the boundaries between subgrids at each of the three previous steps, rebuilds them, and proceeds to step 1

a lot of time. Thus, rebuilding the subgrid remains the most time-consuming operation.

Algorithm 42 requires several additional data structures to be implemented. Assembling a global grid from subgrids is greatly simplified if grid nodes at the boundaries between subgrids retain information about the original global numbering. Comparison of global indices leads to fast and reliable finding of pairs of identical grid nodes. Note that step 4 of Algorithm 42 can lead to a low quality of near boundary tetrahedra and, as a consequence, to a low quality

subgrids and the global grid. Therefore, changing the direction of cutting the grid into subgrids is very important for the convergence of the algorithm. On fig. 5.19b shows the case when all boundary tetrahedra in a quasi-uniform mesh with step h are of poor quality . These tetrahedra and their neighbors, schematically represented by parallelepipeds of thickness 2 h, are divided into subdomains by planes orthogonal to the second eigenvector of the inertia tensor. The eigenvectors of the inertia tensor are three orthogonal directions in space. However, even after changing

the three directions, the global mesh may contain tetrahedra whose vertices always remain at the boundaries between the submesh. As a result, only a part of the basic algorithms was applied to these tetrahedra. To improve their quality, the last step in Algorithm 42 is necessary. The number of such tetrahedra is proportional to the number of intersections of different triplets of parallel planes, equal to (P ÿ 1)3. Since the number of processors is small, the last step of the algorithm is implemented on one processor. Another interesting observation is that the dynamics of mesh modifications can be quite different in parallel and sequential meshing algorithms, since the parallel algorithm modifies the global mesh in P different places simultaneously [66, 67]. Let's consider a model example showing that this

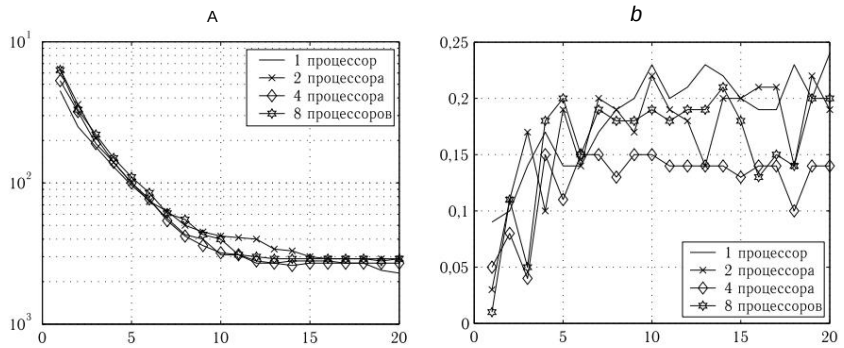
can greatly affect the operation of the algorithm. Numerical experiments [66] were carried out on a COMPAQ Tru64 Cluster parallel computer with ev6 processors with a frequency of 667 MHz. Consider the Poisson equation in the domain ÿ with one input

angle, ÿ=(0, 1)3 \ [0, 0,5] 3, and singular right-hand side:

$$\begin{cases} \ddot{y}\,\ddot{y}u = |x \ddot{y} x0|^{\ddot{y}1} & \text{in } \ddot{y}, \\ u = 0 & \text{on } \ddot{y}\ddot{y}, \end{cases} \qquad (5.4.2)$$

where x0 = (0.5, 0.5, 0.5). The properties of the solution to problem (5.4.2) were studied in [27]. The solution has weak anisotropic edge singularities and a strong isotropic singularity at the vertex of the reentrant angle. To solve the equation, we apply the finite element method with piecewise linear basis functions. To estimate the discretization error, we replace the unknown exact solution u with a discrete solution u computed on a very fine adaptive mesh containing approximately 1.5 × 106 tetrahedra. To construct an adaptive grid, problem (5.4.2) needs to be solved several times on a sequence of grids with an approximately equal number of cells. We use 20 iterations of the adaptive algorithm 45 presented in the application, at each iteration of which algorithm 42 is used.

On fig. Figure 5.20 shows the results of re-meshing for various values of P. Graphs in fig. 5.20a *show* the maximum error rate between u and the finite element solution uh



Rice. 5.20. Error u ÿ uhLÿ(ÿ) *(a)* and grid quality Q(ÿh) *(b)*

as a function of the number L of the adaptive iteration. This error is approximately the same for all parallel calculations. Graphs in fig. 5.20b show that the mesh quality after five adaptive iterations is greater than 0.1 for all values of P, which indicates the robustness of Algorithm 42. The next two tables show the arithmetic complexity of parallel re-

meshing . Arithmetic complexity is measured by the number of basic #mod operations implemented. In table. Figure 5.3 shows the time and number of basic operations performed for the last, twentieth, iteration. The number of local basic operations is proportional to N(ÿh), and the execution time of each operation is approximately proportional to N(ÿh)1/2. Apparently, this dependence is due to the non-optimal implementation of algorithms for working with ordered

list of element qualities.

T a b l e 5.3

Remeshing time, P = 1, L = 20

Nt(ÿh)	9735	19,359	28,151	36,134	52,079	100,075	160,944
#mod	731	1155	3747	3097	6075	11 147	18 904
cpu time 1.2 cpu time	1.9	4.7	4.6	11.2	25.3	58.6	103 1.6
#mod	1.6	1.3	1.5	1.8	2.3	3.1	

Vtab. Figure 5.4 shows the number of implemented basic operations in parallel computations and the total time of mesh rebuilding. It is pertinent to note that only after the 10th adaptive iteration, when the grid was practically established, the time of grid rebuilding became proportional to the number of implemented basic operations. On non-stationary meshes, the number of basic operations may not correlate with the mesh rebuilding time. This is explained by different spatial distributions of local modifications of the grid.

T a b l e 5.4

Number of basic operations and re-meshing time with Nt(ÿh) ÿ 160 000

L	P=1	P=2	P=4	P=6	P=8
	#mod with	#mod with	#mod with	#mod with	
1	30 403 27.2	16 204 15.2	8752 7.9	6448 7.2	5484 6.5
2	30 850 39.6	18 273 27.1	12 340 14.5	8231 11.6	6937 10.8
4	28,027 48.7	22,585 38.4	9804 16.6	6570 12.8	5294 11.8
6	33,018 74.6	19,332 38.8	7643 15.0	4841 11.2	3061 9.4
10	32,986 87.9	13,776 30.2	4996 11.7	2602 8.7	1445 7.2
14	23 878 65.9	10 336 21.5	3741 9.8	1885 7.6	911 6.3
18	25,056 75.8	9802 20.1	2842 8.6	1536 6.3	434 5.7
20	18 904 58.6	7902 15.6	3265 9.7	1384 6.0	785 6.1

Another interesting observation is that the number of basic operations implemented on one processor is not always inversely proportional to the number of processors. At the first adaptive iterations, the total number of basic operations #mod · P grows with P. However, when the grid is established, it decreases, which leads to a superlinear acceleration of the grid construction time (Table 5.5). We emphasize that this is a new and interesting property of the grid generator.

T a b l e 5.5

Acceleration during parallel re-meshing with Nt(ÿh) ÿ 160 000

P ÿÿÿÿ L	1	2	3	4	5	6	7	8	9	10	eleven	12
2	1.8	1.5	1.3	1.3	1.4	1.9	2.2	2.6	2.8	2.9	3.1	3.4
3	2.4	1.9	2.0	2.1	2.9	3.4	4.4	5.3	4.9	5.4	5.1	6.1
4	3.4	2.7	2.3	2.9	3.7	5.0	5.4	6.1	6.5	7.5	7.0	7.5
6	3.8	3.4	3.3	3.8	4.6	6.7	7.5	9.3	10	10.1	9.7	10.1
8	4.2	3.7	3.5	4.1	5.4	7.9	9.4	10.5	11.8	12.2	12.5	12.8

Superlinear acceleration is due to different orders of execution of local modifications of the grid. It turns out that if the mesh is too far from the one established in the course of adaptation, then the most efficient sequence of its local modifications is to choose a tetrahedron with the lowest quality. Therefore, the total number of basic operations is less for P = 1. If the grid is approximately adapted to the solution, then the sequence

its local modifications, which consists in choosing the worst tetrahedra in subgrids. Being separated in space, such local modifications improve the quality of a larger number of tetrahedra, which leads to a faster decrease in #mod on 6 and 8 processors.

§ 5.5. Fixing and unraveling meshes

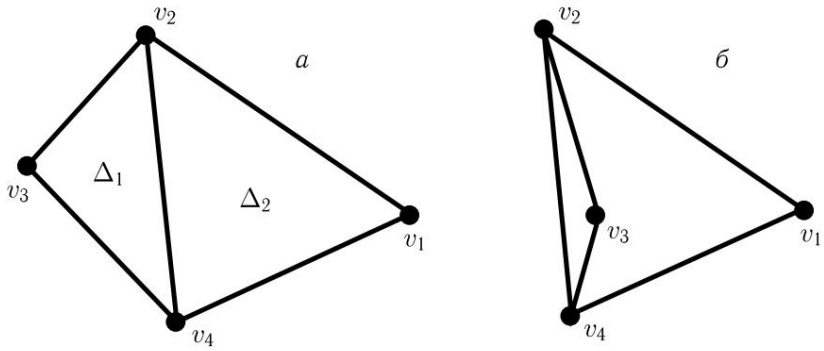
Over the past half century, many methods have been proposed for generating high quality triangular meshes. Unfortunately, a similar statement is not true for tetrahedral meshes, since each of the methods for constructing tetrahedral meshes has its own advantages and disadvantages. For example, both the advancing front method and the Delaunay tetrahedrization method can be used to construct strongly flattened tetrahedra in which all four vertices lie practically in the same plane (slivers). To remove such tetrahedra and correct the mesh, various methods of correcting or improving the quality of the mesh are used. In this section, to correct the mesh, we apply the methods of mesh rebuilding, the distinguishing feature of which is

changing the mesh topology.

For complex geometric models, mesh generators can produce topologically correct but physically incorrect meshes. For example, using simple methods to close the gaps generated in the advancing front method completes the meshing process, but can lead to a confusing mesh. In particular, therefore, in § 3.7 we use a rather complicated method to close gaps. Another source of entangled grids is Lagrangian methods, in which the grid moves along with the flow, becoming entangled in the process. Remeshing methods will help correct the mesh in both cases. Recall that the quality of a regular mesh is defined as a positive function: 0 < Q(ÿh) 1. In order to apply basic meshing algorithms to an entangled mesh, we generalize the concept of the quality of a simplex Q(ÿ) as follows. Let's assume that for

the tangled mesh is topologically correct, which is true in the vast majority of cases. Then checking the mesh entanglement reduces to checking the sign of algebraic areas (in two dimensions) or algebraic volumes (in three dimensions) of pairs of neighboring simplexes. Let us consider triangular meshes in more detail, although everything said will be true for tetrahedral meshes as well. An entangled triangular grid always contains a pair of cells ÿ1 and ÿ2 with a common edge and with the same signs of algebraic areas [see (5.2.1)], as shown in fig. 5.21b :

Sÿ241 Sÿ243 0. (5.5.1)



Rice. 5.21. Pairs of untangled *(a)* and tangled *(b)* triangles

Let's mark both triangles ÿ1 and ÿ2 as entangled and assign negative qualities to them:

Q (ÿ1) = ÿQ(ÿ1), Q (ÿ2) = ÿQ(ÿ2). The qualities

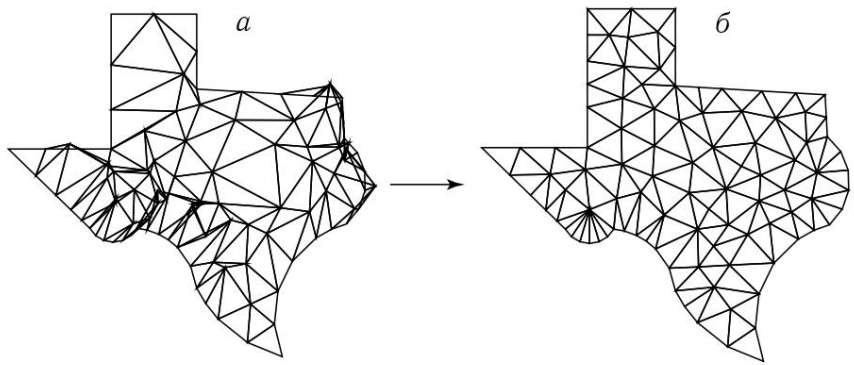
of the remaining triangles do not change, i.e. Q (ÿ) = Q(ÿ). The new quality of the triangle Q (ÿ) changes continuously from ÿ1 to 1 and equals 0 when one of the entangled triangles degenerates into a segment. The continuity of the modified quality Q

(ÿ) is necessary for the successful implementation of basic algorithms. Unraveling the grid occurs due to the growth of quality from a negative value, through zero, to a positive (real) quality. The modified quality does not require changes in the basic algorithms, except for the extension of the quality calculation algorithm. In addition to the existing algorithm, it is required to check the sign in inequalities (5.5.1) and change the quality sign of the triangle, if necessary. Let's consider two examples. The initial entangled mesh shown in Fig. 5.22, a, contains 158 triangles. The mesh was artificially tangled by random

displacement of nodes. Note that some of the nodes were shifted outside the region. The quality of the entangled mesh is Q(ÿh) = ÿ0.93 with the average quality of triangles being 0.24.

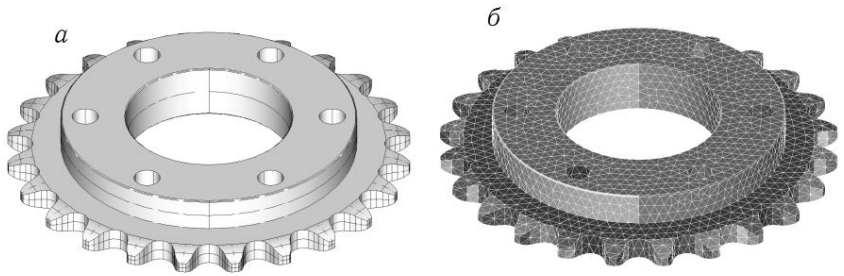
To correct this tangled grid, we apply Algorithm 35 with the modifications described in § 5.2. To preserve the boundary of the region, we will fix the grid nodes on this boundary. The finite quasi-uniform mesh ÿh shown in Fig. 5.22b contains 145 triangles. Its quality is Q(ÿh) = 0.53, while the average quality of triangles is 0.84. The second example illustrates the use of basic meshing algorithms

to improve the quality of a tetrahedral mesh. In this example, the quality of a tetrahedron is defined as the quality of its shape, i.e., Q(ÿ) = 1 only on a regular tetrahedron of any size. A detailed description of the various qualities is given in Chap. 6.



Rice. 5.22. Unraveling a triangular mesh by rebuilding it

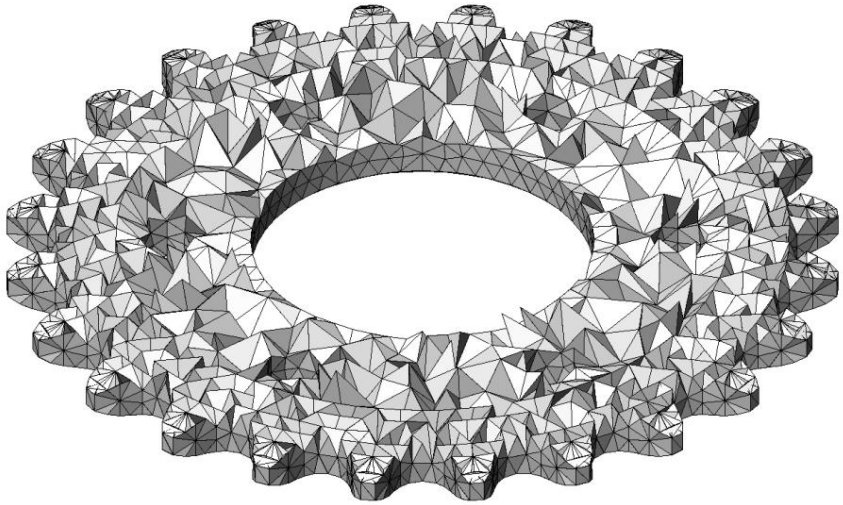
The geometric model of the gear shown in fig. 5.23, a, was created using CAD. It contains 422 nodes, 636 curved edges and 217 curved surfaces. For this model, two tetrahedral meshes with different numbers of elements were constructed by combining the advanced front method with the Delaunay tetrahedrization method, as described in Chap. 3. To build a coarse mesh, the surface of the gear was divided into 7084 triangles with 3530 nodes, as shown in fig. 5.23b . To build a fine grid, the model surface was divided into 38834 triangles with 19405 nodes.



Rice. 5.23. Gear model *(a)* and surface mesh with 7084 triangles

Recall that a reliable algorithm for constructing a three-dimensional mesh (see § 3.7) successively uses the advanced front (F) and Delaunay tetrahedrization (D) methods. The advanced front method is not reliable enough, and in our example it was only able to mesh a patch with a volume of 99.96% of the domain volume. The same behavior of the advancing front method was observed in other examples considered in Chap. 3. After the completion of the method, 32 triangles in the coarse grid and 352 triangles in the fine grid remained in the front. The quality of the unfinished grids turned out to be quite low

(see tables 5.6 and 5.7). The grids were completed by the Delaunay tetrahedrization method. After that, the quality of the meshes dropped even more—to 6.1 10ÿ4 in the fine mesh and to 1.6 10ÿ5 in the coarse mesh. The detailed distribution of tetrahedra by quality is presented in Table. 5.6 and 5.7.



Rice. 5.24. Spatial mesh slice for the gear. The mesh contains 6870 nodes, 7084 boundary triangles and 22456 tetrahedra

T a b l e 5.6

Quality distribution of tetrahedra in a coarse mesh

Method	Q(ÿh)	Nt	10-1 10-2 10-3 10-4 10-5			
F	1.06 10ÿ2 13655 13375280 0 P+D 6.12				0	0
10ÿ4 13687 13390290 5 P+D+PS 1.44 10ÿ1 25485					2	0
25485 0				0	0	0

Mesh rebuilding algorithms (GR) significantly improved their quality - up to 0.14 in a coarse grid and up to 0.2 in a fine grid. The low quality tetrahedrons that appeared after the Delaunay tetrahedrization were successfully rebuilt. Part of the spatial grid is shown in fig. 5.24. Building a fine grid required 9 min 33 s. Most of the resources were spent on the formation of the initial front:

T a b l e 5.7

Distribution of tetrahedra by quality in a fine mesh

Method	Q(ÿh)	Nt	10-1 10-2 10-3 10-4 10-5			
F	1.01 10ÿ3 140,442 139,684,750 8 1.60 10ÿ5				0	0
F+D	140,723 139,817,854 37				5	10
				0	0	0

8 min 40 s. The construction of the spatial grid took 41 s, and its rebuilding took only 12 s.

Other examples of correction of tetrahedral meshes are presented in § 3.7.

Chapter 6

MANAGING MESH PROPERTIES

In this chapter, we will look at various methods for constructing meshes with given properties, including controlling the local size of simplices (triangles or tetrahedra), pulling simplices in a given direction, and controlling the number of simplices in a mesh. The control is carried out by splitting the quality of the simplex into factors that are responsible for its size and shape.

§ 6.1. Controlling properties of regular grids

In a regular grid, simplices can vary considerably in size, while remaining close in shape to regular simplices. More precisely, each simplex ÿ in a regular grid satisfies the following condition:

$$\frac{R\ddot{y}}{r\ddot{y}} \quad c,$$

where Rÿ and rÿ denote the radii of the circumscribed and inscribed spheres, respectively, and the constant C does not depend on the simplex. For C ÿ 1, the regular grid does not contain slivers and anisotropic simplices.

Regular grids represent an important class of grids for applications. First, they can condense to the features of the model or to the features of the solution according to the law specified by the user. Second, the vast majority of theoretical results on the stability of numerical methods and estimates of the accuracy of the numerical solution have been proved for regular grids. To construct regular grids, it is necessary to specify the grid step

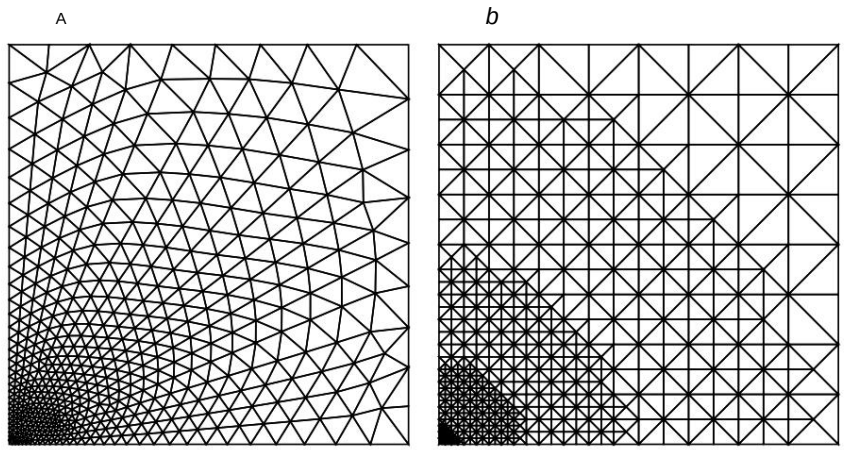
function h(x), which determines the desired diameter of the simplex at the point x. In this section, we assume that the function h(x) is known and consider three methods for constructing regular grids.

Consider the unit square [0, 1]2 and construct a regular grid there, which condenses to the point (0, 0). Define the function h(x) h(x) = 10ÿ4 + 10ÿ2 x21/2, as

(6.1.1)

where x denotes the Euclidean norm of the vector x, x2 = x2 + y2. An unstructured grid constructed by the advancing edge method with a grid step h(x) is shown in Fig. 6.1, a. It contains

999 triangles and 540 knots. The grid constructed by the method of multi-level hierarchical refinement is shown in fig. 6.1b . It contains 908 triangles and 494 nodes.



Rice. 6.1. Grids constructed by the advancing front method *(a)* and the multilevel hierarchical refinement method *(b)*

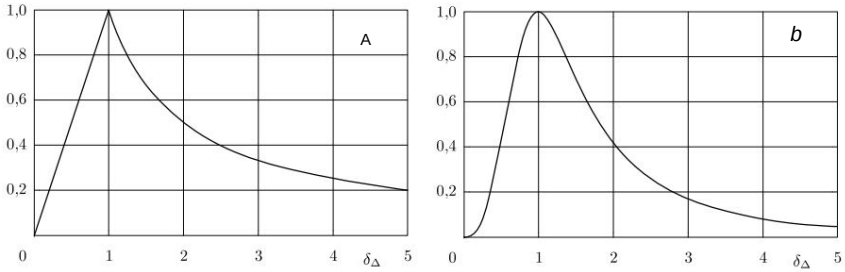
Both grids have advantages and disadvantages. In an unstructured grid, the size of triangles changes smoothly, without large jumps. In the numerical solution of partial differential equations, the smoothness of the grid can lead to a more accurate discrete solution. On the other hand, the implementation of a number of numerical methods is simpler on hierarchical or structured grids. The hierarchical nature of the grid makes it possible to use more efficient methods for solving emerging algebraic problems and interpolating grid solutions between two grids in nonstationary problems.

The third method for constructing regular grids uses the grid rebuilding algorithms described in Chap. 5. These algorithms require the introduction of the quality of the simplex Q(ÿ). Most existing approaches define the quality of a simplex as the product of two factors:

$$Q(\ddot{y}) = Q_{size}(\ddot{y})\, Q_{shape}(\ddot{y}),$$

each of which varies from 0 to 1. The first factor is responsible for the size of the simplex, the second for its shape. Let pd and Vd denote, respectively, the sum of the edge lengths and the volume of a regular simplex with a unit edge, where d = 2 for a triangle and d = 3 for a tetrahedron. Simple calculations give:

$$p_d = \frac{d(d+1)}{2}, \qquad V_d = \frac{\sqrt{d+1}}{d!\sqrt{2^d}}.$$



Rice. 6.2. Size quality Qsize(ÿ) calculated by formulas (6.1.2) *(a)* and (6.1.3) *(b)* as a function of ÿÿ

The multiplier Qsize(ÿ) is usually built on the basis of the grid step function h(x), for example:

$$Qsize(ÿ) = \min_{ÿÿ} \frac{1}{ÿÿ}, \quad ÿÿ = \frac{pÿ}{pdh(x\ ÿ)}, \text{ where} \qquad (6.1.2)$$

x ÿ is the center of mass of the simplex ÿ, p ÿ is the sum of the lengths of its edges:

$$pÿ = \sum_{eÿÿ} le.$$

Thus, regardless of the value of ÿÿ, the quality Qsize(ÿ) 1.

Such a definition of Qsize(ÿ) leads to the discontinuity of the derivatives of Qsize(ÿ) with respect to ÿÿ and the coordinates of the vertices of the simplex. In basic algorithms for optimizing the position of a grid node, the differentiability of the quality of a simplex is an optional but desirable property. Therefore, in the examples below, we will determine the quality of the simplex using the following differentiable function F:

$$Qsize(ÿ) = F(ÿÿ) = \min_{ÿÿ} \frac{1}{ÿÿ}, \quad 2\ ÿ \min_{ÿÿ} \frac{1}{ÿÿ}^3. \qquad (6.1.3)$$

This quality increases for 0 < ÿ ÿ 1, decreases for ÿÿ > 1, and has a single maximum (Qsize(ÿ) = 1) at the point ÿÿ = 1 (see Fig. 6.2). The factor Qshape(ÿ), which is responsible for the shape of the simplex, can also be determined in a nonunique way. In the examples below, we use the following quality:

$$Qshape(ÿ) = \frac{pdd}{Vd} \frac{}{Vÿpdÿ}. \qquad (6.1.4)$$

In a two-dimensional space, another common formula for the shape quality uses the sum of squared edge lengths instead of p2 ÿ. Note that Qshape(ÿ) is a dimensionless quantity with a maximum value of 1, which is achieved only on a regular simplex. In general, any combination of dimensionless quantities that reaches a single maximum on a regular simplex can be used as a definition of the shape quality. For more

For a detailed analysis of various qualities of the simplex, we refer the reader to [62]. Let's put

$$Q(ÿ) = \frac{pdd}{Vd} \frac{Vÿ}{pdh(x\ ÿ)} F(ÿÿ), \quad ÿÿ = pd\ ÿ \frac{pÿ}{}. \qquad (6.1.5)$$

Thus, the maximum quality value Q(ÿ) is achieved on a regular simplex with edge length le = h(x ÿ). Compared to the formulas in § 2.1, formula (6.1.5) contains an additional factor F(ÿÿ). Vtab. Table 6.1 shows the shape quality (6.1.4) and size quality (6.1.3) for an isosceles

triangle with angles ÿ, ÿ = 180ÿ ÿ 2ÿ and base h(x ÿ) = 1. Note that the shape quality Qshape(ÿ) = 0.8 can mean both obtuse and acute isosceles triangles. This is insufficient for constructing good meshes for finite element calculations. The size quality has a more pronounced maximum ÿ = 60ÿ, so that as a result the quality Q(ÿ) 0.8 is achieved only on an acute triangle. The balance of the qualities of shape and size can be adjusted in many ways, for example,

$$Q(ÿ) = Qshape(ÿ)^n\ Qsize(ÿ)^m,$$

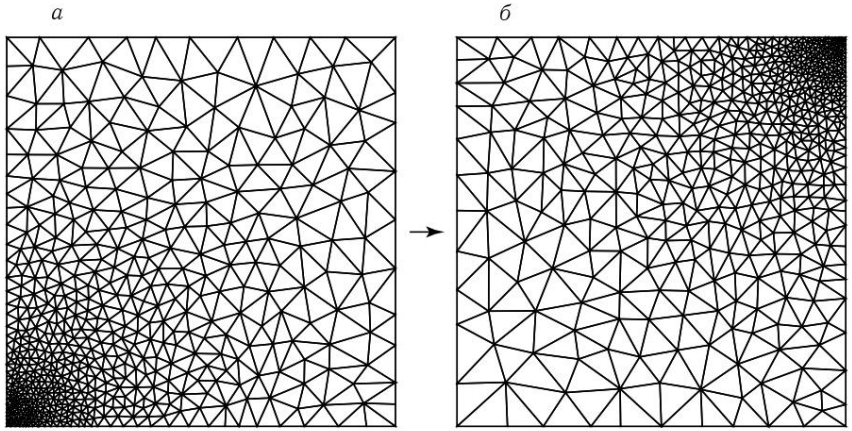
where n and m are some positive parameters. Thus, when analyzing the properties of the constructed grids, it is necessary to use an analog of Table. 6.1 for each new definition of the quality of a triangle or tetrahedron.

T a b l e 6.1

Quality of an isosceles triangle calculated by formula (6.1.5)

ÿ ÿ	5ÿ 15ÿ 25ÿ 35ÿ 45ÿ 55ÿ 65ÿ 75ÿ 85ÿ 170ÿ 150ÿ 130ÿ 110ÿ 90ÿ 70ÿ 50ÿ 30ÿ 10ÿ
Qshape(ÿ) Qsize(ÿ)	0.11 0.34 0.55 0.74 0.89 0.99 0.98 0.82 0.38 0.70 0.72 0.76 0.81 0.89 0.98 0.96 0.62 0.08
Q(ÿ)	0.08 0.24 0.41 0.60 0.79 0.96 0.95 0.51 0.03

In practice, the construction of an ideal grid with quality Q(ÿh) = 1 is unrealistic. First, there is no two-dimensional condensing grid of equilateral triangles. Second, a tetrahedral grid of regular tetrahedra cannot cover space. Thirdly, dihedral and planar angles in the geometric model limit the maximum possible mesh quality. These limitations should be taken into account when choosing the quality Q0 in the remeshing algorithm 35.



Rice. 6.3. Meshes generated by the mesh rebuilding algorithm

Let's return to the function (6.1.1). The result of the operation of the mesh restructuring algorithms is shown in Fig. 6.3, a. The mesh contains 1004 triangles and 545 nodes. Despite the visual sensation of the mesh being non-smooth, its quality is quite high: Q(ÿh) = 0.797. The flexibility of this approach lies in the ability to rebuild any grid into a grid with new properties by replacing the grid step function h(x). For example, the choice

$$h(x) = 10ÿ4 + 10ÿ2 \times ÿ\, e21/2,\ e = (1, 1)T,$$

leads to the grid shown in Fig. 6.3b . This grid was obtained by rebuilding the grid in Fig. 6.3, a.

§ 6.2. Managing the properties of anisotropic meshes

The high quality regular grids considered in § 6.1 usually consist of triangles close in shape to a regular triangle. The size of these triangles is controlled by one parameter, the local grid spacing. The construction of anisotropic meshes, where the orientation of triangles plays an important role, requires the introduction of additional parameters, which inevitably lead to the concept of a tensor metric. The Wuclidean metric is the distance between two points x1 and x2

on a plane is measured according to the Pythagorean theorem:

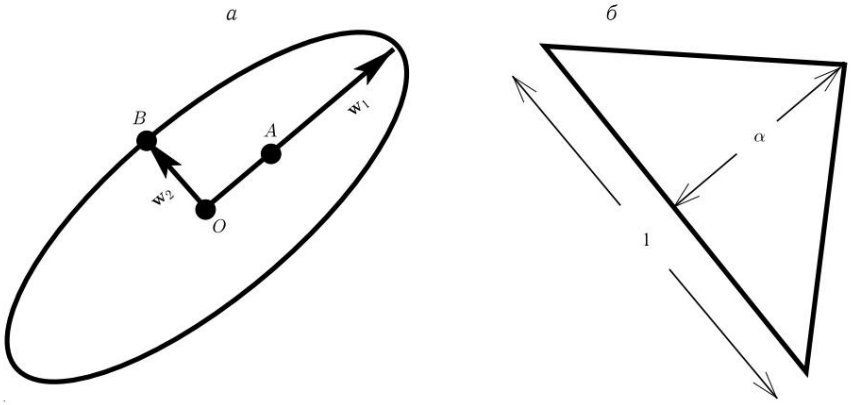
$$ÿ(x1, x2) = \sqrt{(x1 ÿ x2)2 + (y1 ÿ y2)2}.$$

A generalization of the concept of distance requires the introduction of the matrix M, which is a symmetric positive definite 2 × 2 matrix. Distance

between two points x1 and x2 in the constant metric M is calculated as follows:

$$ÿM(x1, x2) = \sqrt{(x1 ÿ x2)TM\,(x1 ÿ x2)}. \qquad (6.2.1)$$

The distance between two points in space is calculated using a similar formula, only in this case M is a 3 × 3 symmetric positive definite matrix.



Rice. 6.4: *a* are the 2 × 2 eigenvectors of the matrix M, *b* the height of the triangle is ÿ = 3ÿ2/(4ÿ1)

Eigenvectors w1 and w2 of a specific matrix 5.89 2.51 2.51

M= 5.01

shown in fig. 6.4, *but* as semi-axes of an ellipse centered at the point O. They correspond to the eigenvalues ÿ1 = 8.0 and ÿ2 = 2.9. In the Clidean metric, points A and B are equidistant from the center O. In the metric M, the distance to point A is ÿ1/ÿ2 times greater than to point B. An equilateral triangle in the metric M is shown in fig. 6.4, b. The height of this triangle is ÿ2/ÿ1 times less than the height of an equilateral triangle in the Euclidean metric. For a strongly anisotropic metric with ÿ1 ÿ2, shown in Fig.

6.4, *b ,* an equilateral triangle in the metric M will look like a strongly flattened one in the Euclidean metric. Another feature of the anisotropic metric is that, as an equilateral triangle rotates in the metric M, its shape will change strongly. For example, when rotated by 90ÿ, instead of a flattened triangle, we get a highly elongated needle-shaped triangle. The ratio of eigenvalues ÿ1/ÿ2 is called *the condition number* of the matrix M, or the anisotropy of the metric. Last

The example shows that an increase in the anisotropy of the metric by a factor of 100 leads to a stretching (or flattening) of the triangle by a factor of 10. In three dimensions, the anisotropy of the metric is defined as the ratio of the maximum eigenvalue of the matrix M to the minimum. A further generalization of the concept of distance assumes that the

elements of the matrix M are functions of the spatial coordinate x. Note that M(x) remains a symmetric and positive definite matrix at every point x. Let ÿ(t) be the simplest parametrization of the edge e = [x1, x2]: ÿ(t) = x1 + t(x2 ÿ x1),

$$0 \quad t \quad 1,$$

for which ÿ t(t) = x2 ÿ x1. The edge length e is calculated as follows:

$$l_{e,M} = \int_0^1 \sqrt{\ddot{y}^T M(\ddot{y}(t))\ \ddot{y}\ t}\ dt = \int_0^1 \sqrt{(x2 \ddot{y} x1)^T M(\ddot{y}(t))\ (x2 \ddot{y} x1)}\ dt. \quad (6.2.2)$$

In the case of a constant metric, formulas (6.2.2) and (6.2.1) coincide. Note also that in the case of a variable metric, a straight line is not always the shortest distance between two points.

When rebuilding the meshes, there is no need to calculate the exact length of the edge in the metric M by formula (6.2.2). It is enough to apply a quadrature of the second order with a central point:

$$l_{e,M} \ddot{y} \sqrt{(x1 \ddot{y} x2)^T M(x12)\ (x1 \ddot{y} x2)}, \qquad x12 = \frac{1}{2}(x1 + x2).$$

In practice, when rebuilding the grid, the metric is calculated either at each simplex or at each node of the grid. In the first case, the metric is piecewise constant and discontinuous between simplices, i.e., the length per mesh edges depends on the simplex on which it is calculated. In the second case, the metric is assumed to be linear in each simplex and continuous in the entire domain, so the edge length is the same for all simplices containing this edge. Therefore, the continuous metric leads to faster rebuilding of the mesh. All meshing examples in this book use a continuous metric. In terms of the Wuclidean metric, the volume of a simplex is calculated by the formula

$$V\ddot{y} = \int_{\ddot{y}} dV.$$

In a metric space, a unit of length in the direction of an eigenvector is multiplied by the square root of the corresponding eigenvalue. Since the determinant of the matrix M is the product of its eigenvalues, we have the following formula for the volume:

$$V\ddot{y},M = \int_{\ddot{y}} \sqrt{\det M(x)}\ dV. \quad (6.2.3)$$

In the case of a constant metric, we obtain the relation

$$V\ddot{y},\ M\ = V\ddot{y}\ \ddot{y}\sqrt{\det M}.$$

Using the formulas for the length of an edge and the volume of a simplex in the metric M, we define the new quality of a simplex as the product of the qualities of size and shape:

$$Q(\ddot{y}) = Q_{size,\,M}(\ddot{y})\ Q_{shape,\,M}(\ddot{y}), \quad (6.2.4)$$

Where

$$Q_{size,\,M}(\ddot{y}) = F(\ddot{y}\ddot{y},M),\quad Q_{shape,\,M}(\ddot{y}) = \frac{pdd}{Vd}\ \frac{}{V\ddot{y},M pd}. \quad (6.2.5)$$

Here the function F is defined by formula (6.1.3), where

$$\ddot{y}\ddot{y},\ M\ = \frac{p\ \ddot{y},\ M}{pdh\ M(x\ddot{y})}, \qquad p\ \ddot{y},M\ = \int_{e\ddot{y}\ddot{y}} l_{e,M},$$

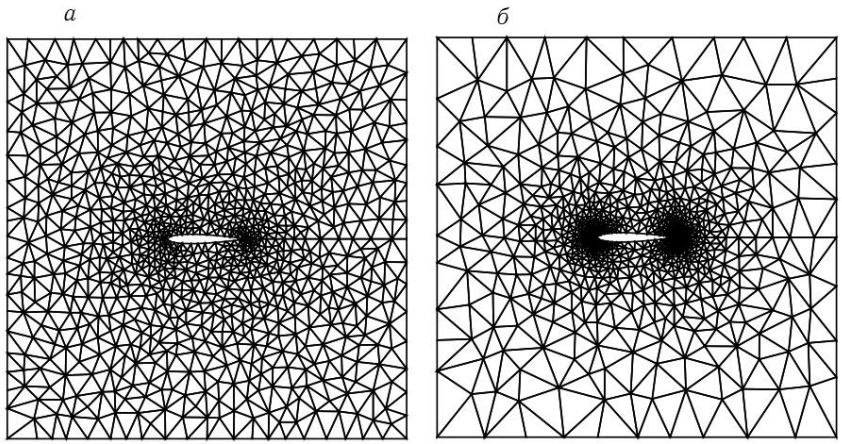
and the function h M(x ÿ) defines the step of a regular grid in a space with metric M. The new quality of the simplex is constructed in such a way that it coincides with the quality (6.1.5) for the isotropic metric:

$$M(x) = h(x)^{\ddot{y}2}\ id, \qquad hM(x\ddot{y}) = 1,$$

where Id denotes the identity matrix of order d. Indeed, is
Using formulas (6.2.3) and (6.2.2), we obtain Vÿ, M = h(x ÿ)ÿ1p ÿ, which = Vÿ h(x ÿ)ÿd
(6.2.4) transforms formulas (6.2.4) and (6.2. 5) to for and p ÿ, M mule (6.1.5). Quality reaches its

maximum when Qsize, M(ÿ) = Qshape, M(ÿ) = 1. The first factor is equal to 1 if all edges of ÿ have the same length h M(x ÿ) in the metric of M. The second the factor is equal to 1 when the ratio Vÿ, M/pd ÿ, M reaches its maximum value, i.e. also on a regular simplex considered in the metric M.

Thus, in a metric space, there is no need to introduce a separate function h(x), which describes the size of the simplex, and parameters that determine its orientation. The metric M controls both the orientation of the simplex and its size.



Rice. 6.5. Isotropic grids with a = 0.5 *(a)* and a = 1 *(b)*

Consider an example of controlling the properties of regular grids. Using the isotropic metric M = h(x)ÿ2 I2, where

$$h(x) = \max\{|(x \ddot{y} 0.4)^2 + (y - 0.5)^2 + 10\ \ddot{y}4|a/2, |(x \ddot{y} 0.6)^2 + (y - 0.5)^2 + 10\ \ddot{y}4\ |a/2\},$$

and by changing the value of the parameter a, we will change the degree of concentration of the isotropic grid to two points (0.4, 0.5) and (0.6, 0.5) on the boundary of the wing-shaped obstacle (see Fig. 6.5). Both grids in this figure contain about 1800 triangles each and are of approximately the same quality. Note that the grid in Fig. 6.5, *b* was built from the grid in fig. 6.5, *and* with the help of the triangular mesh rebuilding methods described in Chap. 5. The main drawback of the metric introduced above is the lack of control over the number of simplices in the

constructed grid. To return this control, it is necessary to establish a correspondence between the desired number of simplices N and the parameter h M(x ÿ). For a visual grid consisting of regular simplices of the same diameter in the metric M, this parameter does not depend on the simplex ÿ; therefore,

$$h\ M(x\ \ddot{y}) = h, \qquad (6.2.6)$$

where h is to be estimated. The area (volume) of the computational domain in two (three) dimensions is calculated by the formula

$$|\ddot{y}|M = \int_{\ddot{y}} N\ Vd\ hd\ .$$

Thus, to estimate the parameter h, it suffices to calculate the area (volume) of the domain ÿ in the given metric M:

$$h = \frac{1}{NVd} \int_{\ddot{y}} \sqrt{\det M(x)}\ dV^{1/d}\ .$$

Using some triangulation (tetrahedrization) ÿ0 of the computational domain ÿ and applying a quadrature with a central point, we obtain a simple estimate of this parameter:

$$h\ \ddot{y} \quad \ddot{y}\ \frac{1}{NVd} \sum_{i=1}^{N(\ddot{y}0\ h)} V\ddot{y}i \sqrt{\det M(x\ddot{y}i)}^{1/d} \ddot{y}\ . \qquad (6.2.7)$$

Thus, to construct a grid with N simplices, we first calculate the parameter h by formula (6.2.7), using some initial triangulation (tetrahedrization) of the domain, and then the quality of the simplex by formulas (6.2.4)–(6.2.6). In the case of a rough initial estimate of the parameter h, the number of simplices in the constructed mesh N(ÿh) may differ significantly from N. In this case, the constructed mesh can be used to more accurately estimate the parameter h and the mesh rebuilding process can be repeated. As we noted above, in the overwhelming majority of problems it is unrealistic to construct a mesh with the maximum quality Q(ÿh) = 1. Therefore, the

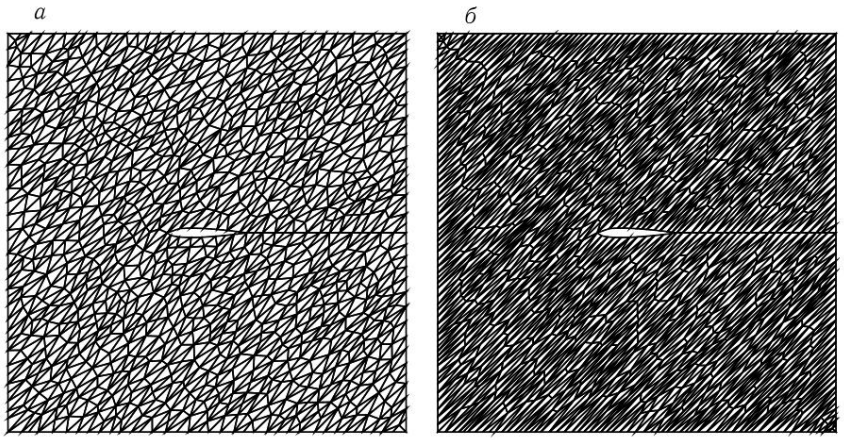
desired number of simplices N will almost always be reached only approximately. The deviation of N from N(ÿh) depends on the quality Q(ÿh) of the constructed grid. The lower the average quality of the grid elements, the greater the deviation of N(ÿh) from the desired value.

Consider an example of controlling the properties of anisotropic meshes. Let 10

$$M = R\ddot{y}/4 \begin{pmatrix} a & 0 \\ 0 & 1 \end{pmatrix} RT\ddot{y}/4,$$

where Rÿ/4 is the 45ÿ rotation matrix. Thus, as a increases, the metric becomes stronger in the direction (ÿ1, 1). A grid that is quasi-uniform in such a metric must stretch in the perpendicular direction (1, 1). Let N = 2000. Consider two values a = 1 and a = 2. In both cases, we estimate h using the grid

shown in Fig. 6.3, a. The result of rebuilding the grid is shown in Fig. 6.6. The grid in fig. 6.6, *a* contains 1850 triangles, and its quality is Q(ÿh) = 0.574, while the average quality of elements is 0.870. Only 9 triangles have quality below 0.8. Note that, despite the relatively high average quality, the deviation of N(ÿh) from N is 7.5%. The grid in fig. 6.6, *b* contains 1867 triangles, and its quality is Q(ÿh) = 0.208 with an average quality of elements of 0.864. Multiple triangles near the top left and bottom right corners cannot



Rice. 6.6. Anisotropic grids with a = 1 *(a)* and a = 2 *(b)*

be strongly elongated in the direction (1, 1), which leads to a decrease in their quality and, consequently, in the quality of the grid. However, due to the high average quality of triangles, the deviation of N(ÿh) from N is only 6.7%.

Our experience shows that the construction of highly anisotropic tetrahedral meshes is a much more difficult problem than the construction of anisotropic triangular meshes. In practice, the algorithms described in Chap. 5 make it possible to build triangular meshes in which elements can be easily stretched by a factor of 103–104 in any direction. For tetrahedral meshes, the degree of extrusion of elements is usually limited to values of 10–100.

Application

SOME PROBLEMS OF GRID ADAPTATION

By *grid adaptation,* we mean the process of constructing or rebuilding a computational grid that is consistent with the geometric features of the computational domain and/or with the features of the grid solution. The process of grid adaptation, as a rule, is aimed at the approximate minimization of some functional under given constraints. In the appendix, we consider several typical examples of such functionals and restrictions on rebuilding some initial simplicial grid.

§ A.1. Adaptation to external and internal boundaries

A common example of grid adaptation is the adaptation of the initial grid to a curvilinear boundary, which can be either external or internal. The main purpose of this adaptation is to improve the approximation of the original boundary by boundary edges or faces of the adaptive mesh. As a functional to be minimized, one can choose, for example, the error of approximation of a curvilinear boundary by a piecewise linear boundary in the maximum norm, and as a constraint, the maximum admissible number of grid nodes. This formulation corresponds to the problem of constructing an adequate discrete model in the class of simplicial grids for a given domain with a curvilinear boundary.

It is obvious that a decrease in the approximation error of a curvilinear boundary by a piecewise linear boundary implies a refinement of the given mesh in the vicinity of the boundary, i.e., an improvement in the resolution of the original discrete model. Depending on the mesh refinement method, it is necessary to implement either a shift of the boundary nodes along the boundary or the placement of new nodes on the boundary. Both operations assume the existence of curved boundary data. Below we

consider several ways to use curvilinear boundary data to adapt the grid in its vicinity. For simplicity of presentation, we restrict ourselves to the case when all boundary nodes of the original grid lie on the boundary of the domain.

Clause 1.1. Adaptation of triangulation to a curvilinear parameterized boundary

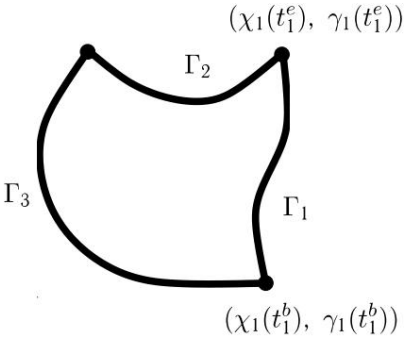
In the two dimensional case, the curvilinear boundary can be parametrized by a set of functions of one real variable called the parameter t. In this case, the boundary is covered by a finite set of curves, each of which is described by continuous functions of the point coordinates:

$$ÿ = \bigcup_{i=1}^{m} ÿi,$$

Where

$$ÿi = \{(x, y) \mid x = ÿi(t), y = ÿi(t), t ÿ [t^{b}_{ti}, t^{e}_{i}]\}.$$

Here tb_{i}, te_{i} are the ends of the curve ÿi in the parametric space.
The notation introduced is shown in Figs. P.1.



Rice. P.1. An example of an area with three curved borders

Since all boundary nodes of the original grid lie on the boundary ÿ, then for any point v ÿ ÿi with coordinates (xv, yv) lying between the two nearest boundary nodes v1 and v2 with parameters t1 and t2, respectively, there exists a parameter tv ÿ [t1, t2] such that xv = ÿi(tv), yv = ÿi(tv).

Thus, the shift of any point v with the parameter tv along the boundary segment ÿi reduces to a successive increase or decrease in the parameter tv. Inserting a new node v with coordinates (xv, yv) and parameter tv into the middle of a curved segment parametrized by the functions ÿ, ÿ and connecting two vertices v1, v2 with parameters t1, t2 depends on the definition of the middle of a curved segment. If the midpoint is the image of the midpoint of a segment in the parametric space, then

$$tv = (t1 + t2) /2. \qquad (A.1.1)$$

If the midpoint of the curved segment is a point equidistant from the vertices v1 and v2, then tv is a solution to the nonlinear equation

$$(x1 ÿ ÿ(tv))2 + (y1 ÿ ÿ(tv))2 = (x2 ÿ ÿ(tv))2 + (y2 ÿ ÿ(tv))2 . \qquad (A.1.2)$$

The root tv belonging to the interval (t1,t2) can be found by the bisection method or by Newton's method. If the midpoint of the curvilinear

segment is a point from which the paths along the segment to the vertices v1 and v2 are equal, then tv is a solution to a more complex nonlinear equation

$$\int_{t1}^{t_v} \sqrt{ÿ2(t) + ÿ2(t)}\, dt = \int_{t2}^{t_v} \sqrt{ÿ2(t) + ÿ2(t)}\, dt. \qquad (A.1.3)$$

Vtab. Item 1 shows the values of the parameter tv for the middle of a curved segment (part of a parabola), defined as follows:

$$ÿ(t) = t, ÿ(t) = t 2, 0 t \quad 1.$$

T a b l e A.1
Parameter of the midpoint of a segment calculated by formulas (A.1.1)–(A.1.3).

Formula (A.1.1)	(A.1.2)	(A.1.3)	
tv	0.5	0.618	0.611

Although from a mathematical point of view, the definition of the midpoint of a curved segment by the formula (A.1.3) is the only correct one, our practical experience shows that the use of simpler definitions (A.1.1) and (A.1.2) does not lead to serious distortions of the mesh quality when its adaptation to a curvilinear boundary, since in the methods of re-meshing (Chapter 5) the position of the middle of the segment is corrected by the basic node shift operation. We also note that if the length of the grid edge tends to zero, then the solutions found by formulas (A.1.2) and (A.1.3) converge to each other with the second order.

Clause 1.2. Adaptation of tetrahedralization to a smooth parametrized surface

For 3D regions, the availability of parameterization of external or internal boundaries depends on the specific application. In some cases such parametrizations are known explicitly, as, for example, when the boundary is specified by a finite set of parametrized surfaces. In other cases, these parameterizations are known implicitly, as, for example, when interacting with the CAD kernel. In this case, regardless of the method of setting, each piece of the surface ÿi is parametrized by three continuous functions of two real variables, called parameters p and s, which

are given in some domains ÿi of the two dimensional parametric space. Let

$$ÿ = \bigcup_{i=1}^{m} ÿi,$$

Where

ÿi = {(x, y, z) : x = ÿi(p, s), y = ÿi(p, s), z = ÿi(p, s), (p, s) ÿ ÿi}.

Due to the continuity of the parametrizing functions, for any point v ÿ ÿi with coordinates (xv, yv, zv) there exists a point in the parametric domain (pv, sv) ÿ ÿi such that xv = ÿi(pv, sv), yv = ÿi(pv, sv) and zv = ÿi(pv, sv). One of the basic operations for rebuilding a mesh is

shifting a node along a curved surface. A shift of a point v with parameters (pv, sv) along a piece of the parametrized surface ÿi reduces to changing these parameters. Insertion of a new vertex v with coordinates (xv, yv, zv) and parametric coordinates (pv, sv) on the surface parametrized by the functions ÿ, ÿ, ÿ, midway between two vertices v1, v2 with parameters (p1, s1), (p2 , s2) can be implemented in several ways. First, we can construct the image of the midpoint of the segment in the parametric space:

$$pv = (p1 + p2)/2, \; sv = (s1 + s2)/2. \qquad (A.1.4)$$

Secondly, one can construct a point equidistant from the vertices v1 and v2, whose inverse image in the parametric space lies on the segment connecting (p1, s1) and (p2, s2), i.e.

$$pv = p1ÿ + p2(1 ÿ ÿ), \; sv = s1ÿ + s2(1 ÿ ÿ),$$

where ÿ ÿ (0, 1) is the solution of the non-linear equation

(x1 ÿ ÿ(pv(ÿ), sv(ÿ)))2 + (y1 ÿ ÿ(pv(ÿ), sv(ÿ)))2 + + (z1 ÿ
ÿ(pv(ÿ), sv(ÿ)))2 = (x2 ÿ ÿ(pv(ÿ), sv(ÿ)))2 +
+ (y2 ÿ ÿ(pv(ÿ), sv(ÿ)))2 + (z2 ÿ ÿ(pv(ÿ), sv(ÿ)))2 . (A.1.5) This equation

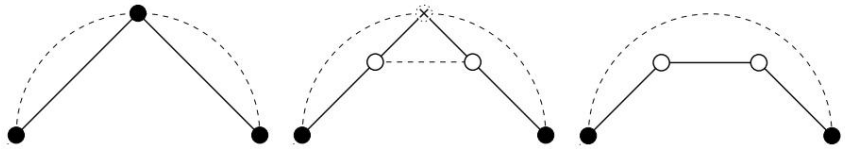
can be solved by the bisection method in the interval (0, 1) or by Newton's method. Note that the bisection method relies exclusively on the procedure for estimating a function of one variable at some points; therefore, the method for calculating the functions ÿ(pv(ÿ), sv(ÿ)), ÿ(pv(ÿ), sv(ÿ)) and ÿ (pv(ÿ), sv(ÿ)) is not important for this approach.

Another way to search for a pair (pv, sv) is to compose and solve a system of two nonlinear equations expressing the equidistance of v from the vertices v1 and v2 and the minimum distance from v to (v1 + v2)/2. In addition to the possible non-uniqueness of the solution of such a system, the search for a solution can be difficult (see also Section 3.4.3).

Clause 1.3. Adaptation of meshes to boundaries with unknown parameterization

In those cases where parameterization of the domain boundary is not available, it is difficult to construct a mesh adapted to the boundary. In this case, as a rule, the boundary of the region is specified using surface triangulation. Let us consider ways of adapting a tetrahedral mesh to a boundary of this type.

The simplest approach is to place new nodes on the current (in the process of adaptation) boundary of the grid area. As the simple two dimensional example shown in Fig. A.2, the removal of boundary nodes, which is possible during adaptation, can lead to a significant error in the approximation of the original boundary by the boundary edges of the grid, even though the step of the adapted mesh is less than the step of the initial mesh.



Rice. P.2. An example of mesh adaptation that degrades boundary approximation

The simplest solution to this problem is to preserve the initial surface triangulation by refining it or shifting the node only along the original piecewise linear surface [43]. Such an approach, which "freezes" the original boundary, naturally preserves the approximation error of the curvilinear boundary by the original set of boundary faces. In the numerical solution of the equations of mathematical physics, an insufficiently high resolution of the boundary surface can lead to a significant error and even negate the efforts expended on constructing an adaptive grid inside the domain.

Another solution to the problem is to use the result of the adaptive calculation as feedback from the CAD system to build a new surface triangulation. This approach requires direct user intervention and may be too complex for some applications. The third solution of the problem is based on the assumption that if the unknown surface of the boundary

is sufficiently smooth (or piecewise smooth), then its triangulation implicitly carries additional information about this surface. Below, we consider a method for constructing a new discrete surface for which the error in the approximation of a smooth boundary is substantially smaller than for a piecewise linear triangulation.

The accuracy of surface approximation can be substantially improved by restoring a higher order surface based on a given piecewise linear surface. Several

methods of local restoration of such a surface, see [53, 70, 71, 73] and references therein. We describe a method [43] that uses discrete differential geometry to compute a piecewise quadratic continuous function approximating the reconstructed surface. The continuity of this function is a hallmark of this method. The Hessian (matrix of second derivatives) of this function is calculated based on the weak formulation,

by analogy with the finite element method.

Consider a smooth piece of the surface ÿ with boundary ÿ. Let ÿh be some piecewise linear approximation of ÿ with discrete boundary ÿh. We assume that the nodes ÿh and ÿh lie on ÿ and ÿ, respectively, although this assumption is not necessary in practice. A piecewise quadratic extrapolation ÿh of a triangulation ÿh is defined as a continuous surface consisting of local quadratic extrapolations ft ÿ ÿh over triangles ft ÿ ÿh.

Let the local extrapolation ft be described by a quadratic function ÿ2, t. Below we will omit the index t if this does not lead to confusion. Let us describe the function ÿ2 in the local coordinate system ÿ = (ÿ1, ÿ2) convenient for us and associated with the plane of the triangle ft, ÿ2 ÿ2 using its Hessian Hÿ2 = {Hÿ2 km}2 ÿÿkÿÿ m

$$H\ddot{y}2 \quad \ddot{y} \quad \text{———} :$$

$$k,m=1,$$

$$1\ \ddot{y}2(\ddot{y}) = \ddot{y} \quad \overline{2} \quad (H\ddot{y}2\ (\ddot{y}\ \ddot{y}\ \ddot{y}i), \ddot{y}\ \ddot{y}\ \ddot{y}i)\ \ddot{y}i(\ddot{y}).$$

$$3$$

$$i=1$$

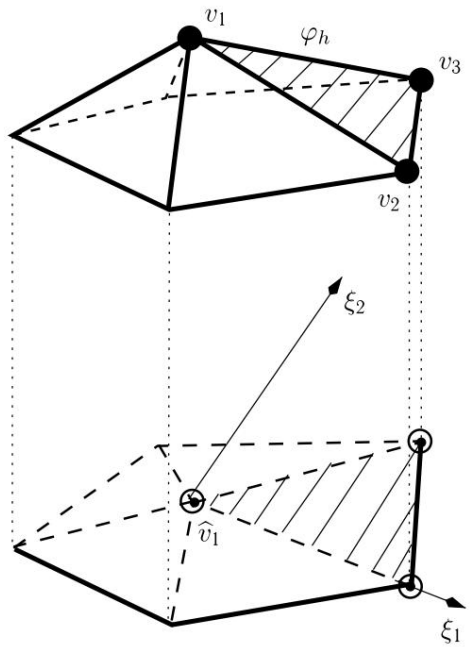
Here ÿi is the coordinate vector of the vertex of the triangle vi, ÿi(ÿ) is the value in ÿ of a linear function equal to 1 in vi and 0 in the remaining vertices of the triangle ft. This formula for specifying a quadratic function is a consequence of Taylor's multipoint formula [39] for representing the error function of linear interpolation of a doubly differentiable function on a triangle ft. The construction of the Hessian Hÿ2 consists of two steps. First, we calculate the Hessian at the

nodes of the grid ÿh and then extend it into triangles ft ÿ ÿh in such a way that the functions ÿ2, t(ÿ) restored on ft form a continuous surface ÿh. Step 1. For each internal node vi of the grid ÿh, consider the superelement ÿ(vi) and define a plane approximating the nodes of this superelement in the sense

of minimum squares and approximating the plane tangent to ÿ at vi. Let (ÿ1, ÿ2) denote the local coordinate system for this plane. We define the superelement ÿi as the orthogonal projection of triangles from ÿi onto the (ÿ1, ÿ2)-plane. On fig. Item 3 shows the superelement ÿi, its projection ÿi, and some triangle from ÿi and its projection onto the plane (ÿ1, ÿ2) are also shaded. Using a change of variables, we define a continuous function ÿi (ÿ1, ÿ2) locally representing the surface ÿ,

and a continuous piecewise linear function ÿi h(ÿ1, ÿ2) locally representing ÿh. Let us assume that both functions are single-valued over ÿi. Finally, we denote the Hessian ÿi by Hÿ and the discrete Hessian ÿh by Hh.

i



Rice. P.3. Local coordinate system for the superelement ÿ(v1)

The eigenvalues and vectors of the Hessian Hÿ(vi) are related to the principal curvatures and principal directions of the surface ÿ at the point vi; therefore, its projection (Hÿ(vi)e, e) onto any unit vector e in the plane tangent to ÿ is the normal curvature in the direction e. If the components of the discrete Hessian Hhkm(vi) at the site vi approximate the components of the differential Hessian Hÿkm(vi), then the quantity (Hhe, e) approximates the normal curvature of the surface ÿ in the direction e. In [51], a method was proposed for calculating the curvatures of a discrete surface ÿh converging to the curvatures of a smooth surface ÿ in the maximum norm as h tends to zero. We use a different approach to estimating normal curvatures, which is simple and works well in practice, despite the lack of theoretical justification. In this approach, the components of the discrete Hessian Hh at the node vi are defined in terms of the weak formulation:

$$Hhkm(vi)\ddot{y}h\ dS = \ddot{y} \quad \frac{\ddot{y}\ddot{y}i\ h\ \ddot{y}\ddot{y}h}{\ddot{y}\ddot{y}k\ \ddot{y}\ddot{y}\ m}dS, \qquad k, m = 1, 2, \qquad (A.1.6)$$

$$si \qquad si$$

which must hold for any continuous piecewise linear functions ÿh that vanish on ÿÿi.

Step 2. The extension of the Hessian inside the triangles is based on the quantities ÿij representing the projections of the Hessian onto the edges eij of the triangle ft. Here and below, we assume that the vector eij starts at the vertex vi and ends at vj = vi+1, where v4 ÿ v1. In the local coordinate system, the vectors eij are given by two coordinates: eij = (eij1 , eij2). Then, by definition of ÿij, we have: eij1 eij2 which generates a system of three linear equations with three unknown matrix elements:

$$\begin{matrix} Hÿ2_{11} & Hÿ2_{12} \\ Hÿ2_{12} & Hÿ2_{22} \end{matrix}, \quad \begin{matrix} e_{ij1} \\ e_{ij2} \end{matrix} = ÿij ,$$

$$e_{ij1}\, e_{ij1}\, Hÿ2_{11} + e_{ij2}\, e_{ij2}\, Hÿ2_{22} + 2\, e_{ij1}\, e_{ij2}\, Hÿ2_{12} = ÿij , \qquad i = 1, 2, 3. \qquad (A.1.7)$$

The following result shows that the matrix of the system is nondegenerate, i.e., the solution (A.1.7) exists and is unique. **Lemma A.1.1.** *The matrix* B of system (A.1.7)

coefficients is nondegenerate. Proof. Direct calculations of the determinant of the matrix B

given e12 + e23 + e31 = 0 give

$$| de\, B| = 2\, e_{121}\, e_{232}\, ÿ\, e_{231}\, e_{132} = 16|ft|^3 > 0,$$

where |ft| is the area of the triangle ft.

Since the function ÿ2, t is quadratic on the edge of the triangle, it is uniquely determined by the values at the vertices of the edge and the corresponding value ÿij . Therefore, to construct a continuous surface ÿh, we assume that it contains the nodes of the original grid ÿh and that the value of ÿij on the edge eij is calculated in the same way for all triangles containing this edge. We define ÿij as the average of two nodal approximations of the Hessian:

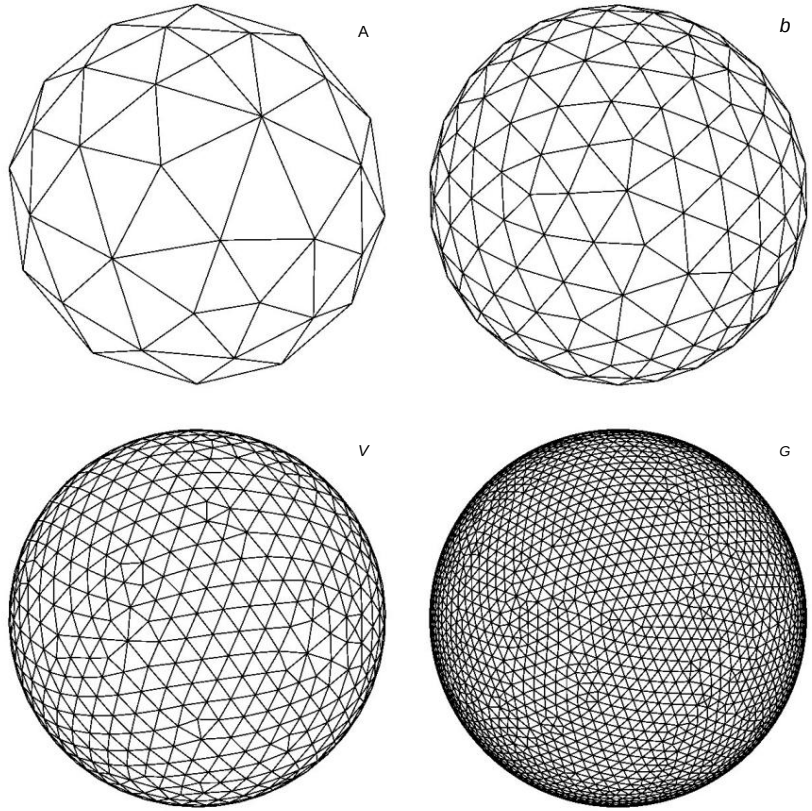
$$aij = \frac{1}{2}\,((Hh(vi)\, eij , eij)+(Hh(vj)\, eij , eij)). \qquad (A.1.8)$$

This definition has two exceptions. If vi ÿ ÿh and vj ÿ/ ÿh, then ÿij = (Hh(vj)eij , eij). If vi ÿ ÿh and vj ÿ ÿh, then ÿij = 0. This is the
It is believed that the nodal approximation of the Hessian is not restored at the boundaries of smooth pieces. This also means that the traces of ÿh and ÿ h coincide on ÿh.

If the surface ÿ can be represented by a smooth function, then the proposed piecewise quadratic extrapolation ÿh provides a better approximation error ÿ than ÿh [43]. In other words,

$$ÿ\ ÿ\ ÿ2, tLÿ(ft) ÿ\ ÿ\ ÿhLÿ(ft).$$

As an illustration, consider a sphere ÿ of radius 0.18, a smooth surface without boundary, ÿ = ÿ. Consider a sequence of quasi uniform surface triangulations ÿh with steps h = 0.1; 0.05; 0.025; 0.0125 shown in fig. P.4.



Rice. P.4. Quasi-uniform triangulations of a sphere of radius 0.18 with steps h = 0.1 (a); 0.05 (b); 0.025 *(c)* and 0.0125 *(d)*

Vtab. Item 2 presents the maximum error rate for approximating the surface ÿ by piecewise linear surfaces ÿh and reconstructed piecewise quadratic surfaces ÿh. These tables confirm that piecewise quadratic completion can significantly reduce the representation error of a smooth surface given by surface triangulation. Note that the components of the discrete Hessian reconstructed by formula (A.1.6) may not converge in the maximum norm to the corresponding components of the Hessian Hÿ as h ÿ 0. This explains

why we do not observe a cubic decrease in the error in the right

T a b l e A.2
Errors in representing the sphere by piecewise linear (ÿh) and reconstructed piecewise quadratic (ÿh) surfaces

h	ÿ ÿ ÿhLÿ ÿ ÿ ÿhLÿ	
0.1	1.3 10ÿ2	1.4 10ÿ3
0.05	3.5 10ÿ3	2.3 10ÿ4
0.025	8.9 10ÿ4	5.7 10ÿ5
0.0125	2.4 10ÿ4	3.4 10ÿ5

column. Nevertheless, the use of quantities (A.1.6) gives a significant gain when using piecewise quadratic completion in practical calculations. The use of a more complicated method [51] for estimating the curvatures of a discrete surface ÿh can further reduce the error in representing the surface ÿ by a reconstructed piecewise quadratic surface ÿh.

§ A.2. Adaptation to the solution through local hierarchical refinement

The fundamental principle of constructing adaptive computational grids is the principle of equipartition of the numerical solution error u ÿ uh over grid cells ÿh. According to this principle, a grid provides the minimum error rate, i.e., minimizes the error functional, among all grids with a given number of cells, if the error rate is the same on all its cells. In this case, the specific type of norm does not matter. The application of the error equipartition principle is based on the monotonic dependence of the error rate on each grid cell on its size and on the possibility of modifying a grid with an unequally distributed error in such a way that the error rate decreases. In many applications, however, the principle of error equipartition cannot be realized, since the error of the numerical solution cannot be calculated due to the unavailability of the exact solution. In these cases, the principle of

equipartition *of the a posteriori error estimate* ÿ ÿ is applied, which can be calculated without an exact solution. The error rates of a numerical solution on a grid with an equally distributed error and on a grid with an equally distributed a posteriori error estimate will be comparable if the local error estimate approximates the local error well:

cÿ ÿ u ÿ uhÿ, ÿ Cÿ ÿ,

where c, C are constants of the order of unity, and ÿ, ÿ are some error norm on the cell ÿ. Various approaches to calculating a posteriori error estimates will be considered at the end of the section. The adaptive construction of the computational

grid by local hierarchical refinement can be implemented based on the available a posteriori error estimate ÿ ÿ and a given threshold ÿ for the maximum error. Algorithm 43 serves as an example of such an implementation. Let us note the following features of the algorithm. This adaptation algorithm does not provide a strict equipartition of the error, but only

does not refine the grid where the local estimate is below a given threshold. The initial grid must be coarse enough to satisfy ÿ ÿ > ÿ on all its cells. Otherwise, on those cells where ÿ ÿ ÿ, the principle of equipartition of the error estimate will be violated.

Algorithm 43. Adaptive meshing by local hierarchy
chemical grinding

1: Construct the initial grid ÿh. Put ÿmax = ÿ 2: **while** while ÿmax > ÿ
do 3: Find the numerical solution
uh on the grid ÿh. Put M = ÿ
4: **loop** over all cells ÿ ÿ ÿh
5: Calculate the local a posteriori error estimate ÿ ÿ Add a cell to the set M
6: if ÿ ÿ > ÿ
7: **end loop**
ÿ ÿh Construct a new grid by bisecting Calculate ÿmax = max ÿ cells
9: from the set M (see Algorithms 21 and 24) 10: end **while**

In the case of a domain with a curvilinear boundary, hierarchical refinement in the vicinity of the boundary, strictly speaking, does not make sense, since new nodes may not lie on the boundary of the domain. In such cases, newly appeared boundary nodes must be projected onto the boundary (see § A.1) and the hierarchical nesting of meshes will be violated. However, from a topological point of view, the hierarchical nesting of grids is preserved and the method of multilevel grid refinement can be modified for this case.

If, in addition to the local mesh refinement procedure, there is a local mesh coarsening procedure, then the adaptive algorithm can be strengthened. To do this, we need to require that the initial mesh admits multilevel coarsening (see § 4.4). Algorithm 44 generates a grid in which the error estimate for all cells lies in the interval [ÿÿ, ÿ], 0 <ÿ< 1. The parameter ÿ is chosen in such a way that

so that local refinement (roughening) of the cell reduces (increases) the error by no more than ÿ ÿ1 once.

Algorithm 44. Adaptive meshing by local hierarchical refinement and coarsening

1: Construct an initial grid ÿh that allows multilevel coarsening. Set ÿmax = ÿ, ÿmin = 0 2: **while** ÿmax > ÿ and ÿmin < ÿÿ **do** 3:
Find the numerical solution uh on the
grid ÿh. Put M1 = ÿ,
M2 = ÿ
4: **loop** over all cells ÿ ÿ ÿh
5: Calculate the local a posteriori error estimate ÿ ÿ Add a cell to the
6: set M1 if ÿ ÿ > ÿ Add a cell to the set M2 if ÿ **ÿ** < ÿÿ from
7: the set M1 11: **end while**

9: ÿ ÿ ÿh ÿ ÿ and ÿmin = min ÿ ÿ ÿh ÿ ÿ
10:

The above adaptive algorithms are mainly applicable to the approximate solution of stationary equations of mathematical physics. For nonstationary problems, the following circumstances should be taken into account.

1) In addition to adaptive control over the spatial grid step, adaptive control over the time step is required. Time discretization should provide an error comparable to the spatial discretization error.

2) Adaptive re-meshing based on a posteriori error estimate can be done not at every time step, but once in several time steps [46]. 3) Cell coarsening is an integral part of mesh adaptation.

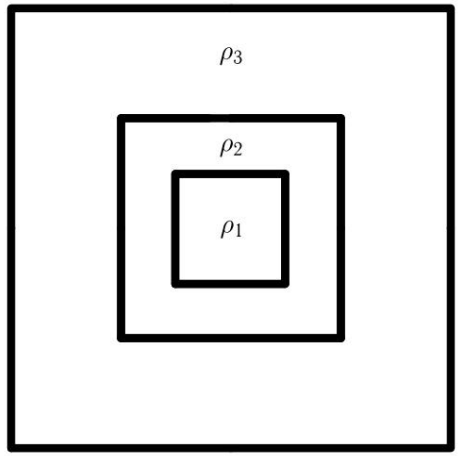
Otherwise, an increase in the number of grid cells over time can slow down or even block the calculation.

In some applications, hierarchical mesh adaptation can be accompanied by node shifting to achieve a better approximation of solution discontinuities such as shock waves and shocks in gas dynamics, material discontinuities and cracking in solid mechanics, and model geometry.

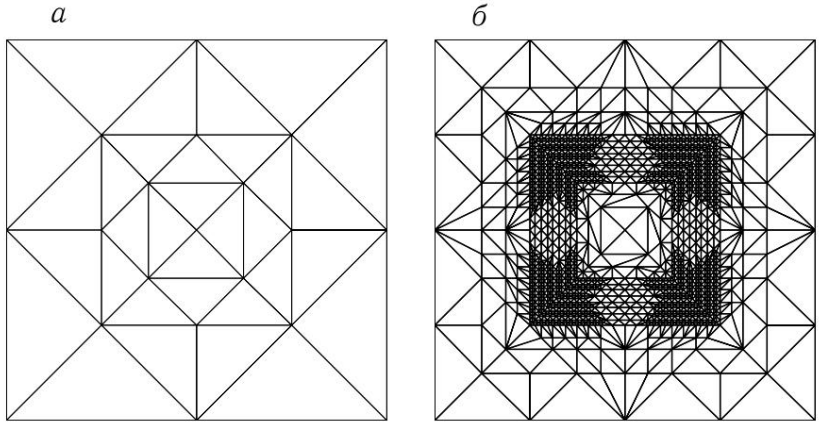
As an example of local hierarchical refinement, consider the approximate solution of the diffusion equation

ÿdiv ÿ ÿ u = f (A.2.1)

with the homogeneous Dirichlet condition by the finite element method on an adaptive triangular grid [57]. The equation is considered in a unit square divided into three subdomains ÿ1, ÿ2, and ÿ3, with ÿ2 being the inner ring (see Fig. A.5). The diffusion coefficient ÿ(x) = ÿi in the subdomain ÿi is defined as ÿ1 = ÿ3 = 1, ÿ2 = 10000, and the right hand side f(x) = 1. The mechanical analogy of the considered boundary value problem is the strain under load of a composite material with fixed boundaries .



Rice. P.5. Split into three subregions



Rice. P.6. Initial mesh *(a)* and refined hierarchical mesh *(b)*

The initial mesh is shown in fig. P.6, a. To construct an adaptive grid shown in Fig. A.6, b, Algorithm 43 was used, in which the a posteriori

residual error estimate given below, and local refinement was provided by the red–green partitioning method [30], an alternative to the bisection method. The error is proportional to the diffusion coefficient, which leads to a strong refinement of the grid in the subregion ÿ2. Let us briefly describe several approaches to the formation of a posteriori error estimates for a piecewise linear finite element solution uh of a two dimensional elliptic equation on regular triangulations. For simplicity, we consider the Poisson equation with the Dirichlet boundary condition:

$$\begin{aligned} \ddot{y}\ \ddot{y}u &= f \text{ in } \ddot{y}, \\ u &= 0 \text{ on } \ddot{y}\ddot{y}. \end{aligned} \qquad \text{(A.2.2)}$$

More details about the methods of a posteriori error estimation and their application to a wider class of problems can be found in [83].

Clause 2.1. A posteriori estimation of the residual error

The a posteriori residual error estimate is based on the identity

$$\ddot{y}(u\ \ddot{y}\ uh)\ \ddot{y}v\ dV = \int_{\ddot{y}} fv\ dV\ \ddot{y} \int_{\ddot{y}} \ddot{y}uh\ \ddot{y}vdV \quad ,$$

valid for a generalized solution u ÿ H1 (ÿ) of problem (A.2.2) and any function v ÿ H1 (ÿ), where H1 (ÿ) is the Sobolev space of functions with zero trace on ÿ ÿ and an integrable square of the generalized derivative [39] . Because the

$$\ddot{y}v = \sup_{w\ddot{y}H1_0(\ddot{y}),\ddot{y}w=1} \int_{\ddot{y}} \ddot{y}v\ \ddot{y}w\ dV \quad ,$$

the gradient error ÿ(u ÿ uh) can be estimated in terms of the residual norm of the finite element solution in the dual space with the norm

$$\sup_{w\ddot{y}H1_0(\ddot{y}),\ddot{y}w=1} \int_{\ddot{y}} fw\ dV\ \ddot{y} \int_{\ddot{y}} \ddot{y}uh\ \ddot{y}wdV .$$

Integrating by parts and using the fact that the solution is linear on each triangle, we can show that

$$\int_{\ddot{y}} fw\ dV\ \ddot{y} \int_{\ddot{y}} \ddot{y}\ uh\ \ddot{y}w\ dV = \int_{\ddot{y}} fw\ dV\ \ddot{y}\ddot{y}\ddot{y}h\ \ddot{y} \sum_{e\ddot{y}\ddot{y}h,\ e\ /\ddot{y}\ddot{y}\ddot{y}\ e} \int_e [ne\ \ddot{y}\ uh]ew\ dS,$$

where [ne ÿ uh]e denotes the jump of the normal component of the flow of the function uh through the inner mesh edge e separating two triangles ÿ ÿ ÿ:

$$[ne\ \ddot{y}\ uh]e = ne\ \ddot{y}\ uh|_{\ddot{y}+}\ \ddot{y}\ ne\ \ddot{y}\ uh|_{\ddot{y}\ddot{y}} .$$

Taking these considerations into account, the local a posteriori estimate of the residual error can be written as

$$\ddot{y}R,\ddot{y} = (diam(\ddot{y}))2\ f2_{L2(\ddot{y})}\ 2 + \frac{1}{2} e\ddot{y}\ddot{y},_{e\ /\ddot{y}\ddot{y}\ddot{y}} |e|\ [ne\ \ddot{y}uh]e2_{L2(e)} \Big)^{1/2} . \qquad \text{(A.2.3)}$$

Factor $\frac{1}{2}$ in the second term takes into account that when summing values of ÿ2, $_{R,\ \ddot{y}}$ over all triangles, the integral over each internal its edge enters it twice. In practice, the calculation of the L2-norm of the right parts on each triangle is carried out with the help of quadrature formulas. Moreover, it is possible to replace the integrable function f by its mean value

$$1\ f\ddot{y} = \frac{1}{|\ddot{y}|} \int_{\ddot{y}} f\ dV \quad ,$$

where |ÿ| denotes the area of triangle ÿ.

Clause 2.2. A posteriori error estimate based on the solution of local subproblems

The main principle of all a posteriori error estimates based on the solution of local subproblems is the formation of a large number of problems with a small number of unknowns. When solving auxiliary problems, it is necessary to use higher order finite element spaces than when solving the original problem (A.2.2). For each triangle ÿ, we form a subdomain ÿ ÿ containing ÿ and triangles adjacent to ÿ along an edge. In each subdomain ÿ ÿ we consider an approximate solution Uh of the local problem

$$\begin{aligned} \ddot{y}\ \ddot{y}\ U &= f && \text{in } \ddot{y}\ddot{y}, \\ U &= uh && \text{on } \ddot{y}\ddot{y}\ \ddot{y} \end{aligned}$$

by the finite element method, in which on each triangle ÿ(v1, v2, v3) the space of polynomials of the first order is enriched with the bubble function bÿ = ÿ ÿ, 1 ÿ ÿ, 2 ÿ ÿ, 3. Here and below, ÿ ÿ, i denotes a linear finite element basis function on ÿ, i.e., ÿ ÿ, i(vj) = ÿij . Note that the piecewise cubic function Uh is piecewise linear on ÿÿ ÿ. A posteriori error estimate based on the solution of local

subtasks, calculated by the formula

$$\ddot{y}\ L,\ \ddot{y}\ = \ddot{y}(Uh\ \ddot{y}\ uh)L2(\ddot{y}\ \ddot{y}). \qquad \text{(A.2.4)}$$

Other options for the formation of local subtasks are also possible. For example, instead of the superelement ÿ ÿ, we can consider the superelement ÿ(v) formed by triangles with a common vertex v. Besides the local Dirichlet problem, one can also consider the local Neumann problem on the inner triangle and the mixed boundary value problem on the boundary triangle:

$$\begin{aligned} &ÿ\ ÿu = f\ ÿ && \text{in } ÿ,\\ &u = uh && \text{on } ÿ\ ÿ\ ÿ\ ÿ,\\ &\frac{ÿu}{ÿn} \quad \frac{1}{2}\,[ne\ ÿuh]e && \text{on } ÿ\ ÿ \setminus ÿ\ ÿ. \end{aligned}$$

This problem is solved by the space enriched finite element method described above.

Clause 2.3. Hierarchical posterior error estimate

The hierarchical a posteriori error estimate [41] for solving problem (A.2.2) is based on the enrichment of the original space of piecewise linear finite elements. Let ÿi denote the piecewise linear basis function associated with node vi. For each edge eij we define a bubble function bij = 4 ÿiÿj . The enriched space is constructed by adding edge functions of bubbles bij to the original basis ÿi . Let Uh denote a finite element solution of problem (A.2.2) in an enriched space. The main assumption of the method is

that the difference between the piecewise linear solution uh and the piecewise quadratic solution Uh is a good approximation to the error u ÿ uh:

$$ÿ(uh\ ÿ\ Uh)L2(ÿ)\ ÿ\ ÿ(uh\ ÿ\ u)L2(ÿ). \text{ To estimate} \qquad (A.2.5)$$

ÿ(uh ÿ Uh)L2(ÿ), we decompose uh and Uh
for basic functions:

$$uh = \sum_{vi} u\,L, iÿi,\ Uh = \sum_{vi} UL, iÿi + \sum_{eij} UQ, ij\ bij\ .$$

The coefficients of this expansion satisfy the systems of linear equations

$$ALL\ u\ L\ = FL \qquad (A.2.6)$$

And

$$\begin{pmatrix} ALL & ALQ \\ AQL & AQQ \end{pmatrix} \begin{pmatrix} UL \\ UQ \end{pmatrix} = \begin{pmatrix} FL \\ FQ \end{pmatrix}.$$

If we expand the function uh ÿ Uh in terms of basis functions:

$$uh\ ÿ\ Uh = \sum_{vi} D\,L, iÿi + \sum_{eij} DQ, ijbij\ ,$$

then the coefficients of this expansion satisfy the system of linear equations

$$\begin{pmatrix} ALL & ALQ \\ AQL & AQQ \end{pmatrix} \begin{pmatrix} D\,L \\ DQ \end{pmatrix} = \begin{pmatrix} FL\ ÿ\ ALL\ u\ L \\ FQ\ ÿ\ AQL\ u\ L \end{pmatrix}, \qquad (A.2.7)$$

where the vectors FL, FQ and u L are known. However, the solution of the system (A.2.7) is too costly, requiring more arithmetic work than the solution of the original system (A.2.6). To construct an easily computed a posteriori estimate, we replace system (A.2.7) with a simplified system

$$\begin{pmatrix} ALL & O \\ About & AQQ \end{pmatrix} \begin{pmatrix} D\,L \\ DQ \end{pmatrix} = \begin{pmatrix} FL\ ÿ\ ALL\ u\ L \\ FQ\ ÿ\ AQL\ u\ L \end{pmatrix},$$

from which it follows that D L = 0, and the vector DQ satisfies the system

$$AQQ\ DQ = FQ\ ÿ\ AQL\ u\ L. \qquad (A.2.8)$$

The matrix AQQ is well conditioned on grids with regular cells, so system (A.2.8) can be effectively solved, for example, by the conjugate gradient method. It can also be shown [41] that

$$\sum_{eij} DQ,ij\ ÿbij \quad ÿ(uh\ ÿ\ Uh)L2(ÿ). \qquad (A.2.9)$$
$$L2(ÿ)$$

Therefore, the value

$$ÿ\ H, ÿ = \frac{1}{2} \sum_{eijÿÿ} DQ, ij\ ÿbij \quad L2(ÿ)$$

is an easily computed hierarchical a posteriori estimate of the local gradient error.

Clause 2.4. A posteriori error estimate from the averaged gradient

The a posteriori estimate of the error over the mean gradient is based on on an easily computable approximation Guh of the function ÿ u, for which

$$ÿ\ u\ ÿ\ Guh\ ÿÿ(u\ ÿ\ uh),\ 0\ ÿ < 1.$$

The triangle inequality leads to a two-sided error estimate:

$$\frac{1}{1+ÿ}\ Guh - ÿ\ uh \qquad ÿ(u\ ÿ\ uh) \qquad \frac{1}{1\ ÿ\ ÿ}\ Guh - ÿ\ uh.$$

The approximation Guh is defined as the discrete L2 projection of a piecewise constant vector function ÿ uh onto the space of continuous piecewise linear vector functions. Projection resolution

is expressed in a special choice of the scalar product based on the replacement of the integral over a triangle by the quadrature formula of rectangles:

$$\ddot{y}\ dV\ \ddot{y}\ (\ddot{y}(v1) + \ddot{y}(v2) + \ddot{y}(v3)).\ 3$$

$$\ddot{y}(v1, v2, v3)$$

Such a replacement leads to the local calculation of the function Guh at any node of the grid v:

$$Guh(v) = \frac{|\ddot{y}|}{\ddot{y}\ddot{y}\ddot{y}(\ddot{v})}\ \ddot{y}uh|\ddot{y}.\ |\ddot{y}(v)|$$

Thus, the a posteriori error estimate for the averaged gradient is defined as follows:

$$\ddot{y}\ Z, \ddot{y}\ = Guh\ \ddot{y}\ \ddot{y}\ uhL2(\ddot{y}).$$

In a certain sense, all considered a posteriori error estimates are equivalent, since they provide upper and lower error estimates for the finite element solution [83].

§ A.3. Adaptation to the grid solution through local modifications

As shown in § 6.2, controlling the properties of a grid using a tensor metric is the most flexible means of controlling the properties of grid cells. The main reason that distinguishes metric control from other methods of control is the possibility of constructing anisotropic grids with highly elongated cells. The advanced front and Delaunay triangulation methods become much less reliable when constructing anisotropic meshes. Section 6.2 considers a method for constructing anisotropic meshes using the tensor metric M(x) as a control. The method reduces to constructing a grid that is quasi uniform in a given metric M(x) by means of a sequence of local modifications of the current grid. We will call such a grid *an M-quasi-uniform grid.*

When developing methods for constructing triangular or tetrahedral meshes that are adapted to the mesh solution, the main problem is the construction of a tensor metric based on the current mesh solution. The choice of a metric is associated with the approximate minimization of a certain error norm of the grid solution on the set of conformal grids ÿh with a limited number of cells N(ÿh):

$$\min_{N(\ddot{y}h)\ N}\ u\ \ddot{y}\ uh\ddot{y}.$$

Suppose that the method for constructing a tensor metric is given, then the algorithm for constructing an adaptive grid with a given number of cells looks very simple.

Algorithm 45. Building an adaptive grid with N elements

1: Construct the initial grid ÿh, find the grid solution uh and calculate the tensor metric M

2: **while** the grid ÿh is not M-quasi-uniform **do**
3: Construct an M-quasi-uniform grid ÿh with N elements Calculate the grid solution
4: uh Calculate a new tensor metric M from
uh 5: 6: **end while**

Below, we consider several methods for constructing a tensor metric based on a grid solution and illustrate the resulting adaptive algorithms with numerical results.

Clause 3.1. Tensor metric based on Hessian recovery

The first method for constructing a tensor metric uses methods for restoring the Hessian of the grid function uh. Recall that the Hessian H of a scalar doubly differentiable function u is the matrix of second partial derivatives of this function with elements ÿ2 u Hkm(u) = ÿxkÿxm In applications, grid solutions are not classically doubly differentiable functions, the classical partial derivatives are , where x1 = x, x2 = y, x3 = z. Because in most replaced by into generalized ones. We present two methods for computing a continuous piecewise linear grid Hessian Hh from a continuous piecewise linear function uh. To do this, it suffices to determine the values of the Hh components at the grid nodes. Let d denote the dimension of the space, d = 2, 3. The most common method for calculating the Hessian values of the grid function uh at grid nodes, based on the idea of finite element discretization of second order elliptic equations, was briefly described in § A.1.

Consider the superelement ÿ(vi) as the union of all simplices containing vi. The components of the grid Hessian Hh, km(vi), k, m = 1, ... , d, are reconstructed at the internal node vi of the grid ÿh as follows:

$$Hh,\ km(vi)\ \ddot{y}h\ dx = \ddot{y} \qquad \frac{\ddot{y}uh}{\ddot{y}xk}\ \frac{\ddot{y}\ddot{y}h}{\ddot{y}x_m}\ dx \qquad (A.3.1)$$

$$\ddot{y}(vi) \qquad\qquad \ddot{y}(vi)$$

for any continuous piecewise linear finite element function ÿh vanishing at the boundary of the superelement ÿ(vi). In the boundary

grid node, the Hessian Hh(v) is defined as a convex linear combination of the Hessian values at the nearest internal nodes: ÿi Hh(vi),

$$Hh(v) = \sum_{vi\ddot{y}\ddot{y}(v),vi=v} \ddot{y}i = 1.$$

In the choice of weights, we follow [23], where the weight is defined as the measure of the intersection of two superelements:

$$\ddot{y}i = |\ddot{y}(v) \ddot{y} \ddot{y}(vi)| \quad \ddot{y} \quad |\ddot{y}(v) \ddot{y} \ddot{y}(vi)| \quad \ddot{y}^{\ddot{y}1} \quad .$$

The grid Hessian Hh(vi) is a symmetric matrix that may be indefinite. Therefore, to construct a metric (a positive definite matrix), the spectral modulus |Hh(vi)| matrices Hh(vi). Consider the spectral decomposition of this matrix:

$$Hh(vi) = W \ddot{y} WT,$$

where W is an orthonormal matrix of eigenvectors Hh(vi), ÿ is a diagonal matrix of eigenvalues ordered in non-decreasing order of their absolute value:

$$\ddot{y} = diag\{\ddot{y}i\}, \quad |\ddot{y}1| \quad \cdots \quad |\ddot{y}d|.$$

We define the following tensor metric:

$$Mh(vi) = |Hh(vi)| = W |\ddot{y}| WT, \text{ where} \qquad (A.3.2)$$

|ÿ| is the diagonal matrix of the absolute values of the eigenvalues values.

In the case of a degenerate Hessian Hh, the metric is formed by formula (A.3.2) using the matrix of perturbed eigenvalues

$$|\ddot{y}| := diag\{max\{|\ddot{y}1|; \ddot{y}\}, ..., max\{|\ddot{y}3|; \ddot{y}\}\} .$$

The main advantage of the above method for constructing a tensor metric from a grid function uh is its independence from the problem the solution of which is this function. The disadvantage of this approach is the possibility of a large error in estimating the grid Hessian using formula (A.3.1), which is clearly manifested in the vicinity of the singularities of the solution. This caused the impossibility to carry out a theoretical analysis of the adaptive algorithm, although the algorithm has proven itself well in practice [5, 9, 66].

The mathematical substantiation of the adaptive construction of an Mh-quasi-uniform grid with metric (A.3.1)–(A.3.2) is the following result for the optimal interpolation problem [5, 23]. For a continuous function u, we define the interpolation operator

Iÿh(u), which uses the value of the function at the grid points and returns a continuous piecewise linear function. **Theorem A.3.1.**

Let u ÿ C2(ÿ), det H = *0,* and ÿh *be an* |*H*|*-quasi-uniformly dimensional grid with* N *elements. Further, let* ÿ ÿ ÿh *be the element at which the maximum error of piecewise linear interpolation is reached, and* H = H(x ÿ), *where* x ÿ | detH(x)|. *Finally,* = arg max xÿÿ, *let* Hh *be a consistent approximation of the Hessian* H *on* ÿ *for which*

$$H \ddot{y} HhL \ddot{y}(\ddot{y}) < \frac{}{q2} |\ddot{y}1(H)|, \qquad q > 0,$$

where ÿ1(H) *denotes the eigenvalue of H closest to zero. Then the error estimate*

$$u \ddot{y} I\ddot{y}h (u)L \ddot{y}(\ddot{y}) C(q, |H|, \ddot{y}) N \ddot{y}2/d \quad . \qquad (A.3.3)$$

Note that the requirement of consistent approximation is always satisfied for a fixed function u for sufficiently large values of N. It can also be shown [5, 23] that for a

function u ÿ C2(ÿ)
born Hessian for sufficiently large N, we have the estimate

$$CN\text{-}2/d \quad \min_{\ddot{y}\ h:N(\ddot{y}\ h)\ N} \quad u \ddot{y} I\ddot{y}_{\ h}(u)L\ddot{y}(\ddot{y}).$$

Thus, under reasonable constraints, an |H|-quasi-uniform mesh with N elements ensures an asymptotically optimal rate of decay of the interpolation error in the maximum norm, i.e., it is quasi-optimal. The error estimates in the maximum norm can be generalized [6] to the Lp-norm 1), p > 0. Moreover, the metric M = |H| should be replaced by the following metric:

$$Mp = (det|H|)\ \ddot{y}1/(2p+d)\ |H|. \qquad (A.3.4)$$

Let ÿh be an Mp-quasi-uniform mesh with N simplices. Then the asymptotic decay rate of the interpolation error in the Lp-norm will be the same as for the maximum norm:

$$u \ddot{y} I\ddot{y}_{\ h}(u)Lp(\ddot{y}) \ CN \ddot{y}2/d \quad ,$$

with a constant C depending on q, |H|, ÿ, but independent of p and N.

The above analysis is directly confirmed by a numerical experiment. Consider the problem of optimizing the error of piecewise linear interpolation of the function

$$u(x, y) = yx2 + y3 + th(6(sin(5y) \ddot{y} 2x)) \qquad (A.3.5)$$

1) For 0 <p< 1, the norm becomes a quasi-norm.

on grids with a given number of cells. To minimize the error in the Lÿ- and L1-norms, we use Algorithm 45, in which the metric is recovered by formulas (A.3.1), (A.3.2), and (A.3.4).

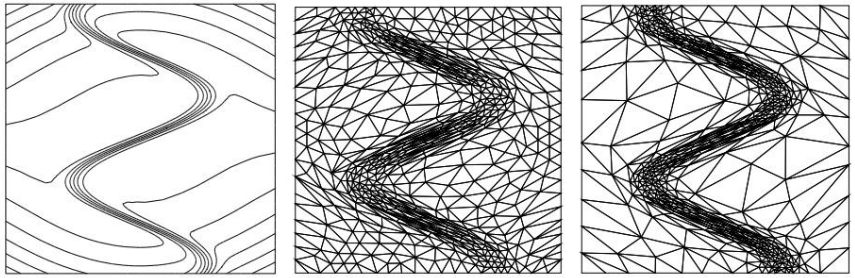
Table A.3 shows that the measured error rates are inversely proportional to the number N of triangles in the grid. On fig. Item 7 shows isolines of the function u and adaptive grids for p = 1 and p = ÿ. As can be seen from the figure, a larger value of p leads to a more intensive refinement of the grid around the singularity of the function.

Table A.3

Interpolation errors of function (A.3.5) on quasi-optimal grids in the L1 and Lÿ norms

N	u ÿ uhL1(ÿ)	u ÿ uhLÿ(ÿ)
2.2	4.4 10ÿ2	1000 1.2 10ÿ2
2000	10ÿ2 5.6 10ÿ3	9.0 10ÿ3 2.8
4000	10ÿ3 5.0 10ÿ3	
8000		

abs



Rice. P.7. Isolines of the interpolated function *(a)* and quasioptimal grids with approximately 1000 triangles minimizing the interpolation error in the L1 *(b)* and Lÿ *(c)* norms

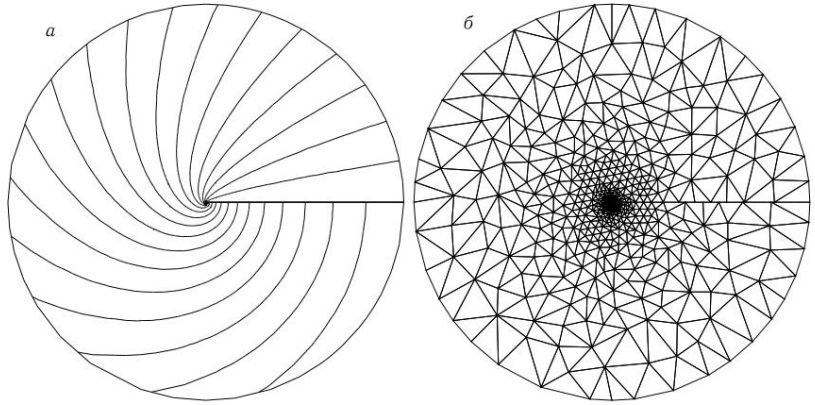
A theoretical analysis of the asymptotic properties of |Hh|-quasiuniformly dimensional grids is presented only for the problem of piecewise linear interpolation. Note that Algorithm 45 can use any grid solution uh on the grid ÿh, for example, a finite element solution of a boundary value problem. Despite the lack of substantiation of both the convergence of adaptive iterations and the asymptotic properties of the adaptive grid constructed in this way, numerical experiments show the applicability of such an extension even for problems with singularities.

Consider, for example, the classical boundary value problem of a crack with the exact solution u(r, ÿ) = r1/4 sin(ÿ/4) in polar coordinates

(r, ÿ) (see Fig. A.8). The problem is defined in a unit circle ÿ centered at the origin without a cut S given by the conditions x > 0 and y = 0:

$$\begin{aligned} \ddot{y}u &= 0 && \text{in } \ddot{y}\backslash S,\\ u &= \sin \ddot{y}4^{-}\text{on } \ddot{y}\ddot{y} \setminus S,\\ u &= 0 && \text{on } S+,\\ \frac{\partial u}{\partial n} &= 0 && \text{on } S\ddot{y}, \end{aligned} \qquad \text{(A.3.6)}$$

where S+ and Sÿ denote the sides of the cut S facing the half planes x > 0 and y < 0, respectively.



Rice. P.8. Isolines for solving the problem of a crack *(a)* and an adaptive grid *(b)*

We will look for a solution to this problem in the space of continuous piecewise linear functions defined on conformal triangulations, and reconstruct the grid Hessian using formulas (A.3.1) and (A.3.2), ignoring the singularity at the origin of the polar coordinate system. As can be seen from Table. A.4, on the constructed adaptive grids (see Fig. A.8, b), the maximum error rate of the finite element solution demonstrates an almost optimal convergence rate: $\|u - u_h\|_{L_\infty(\Omega)} = O(N^{-0.9})$. Cause of slight speed deviation convergence from unity lies in the fact that the finite element method mentov minimizes the energy error norm ÿ(u ÿ uh)L2(ÿ).

T a b l e A.4

Maximum error rate of the finite element solution for the crack problem

N	1000	4000	16000	64000
u ÿ uhLÿ(ÿ)	0.056	0.016	0.005	0.0014

Because the maximum error rate is not subject to the energy norm, it can fall at a slower asymptotic rate.

Note that the constructed adaptive grids provide a much smaller error than conventional quasi-uniform triangulations: the error rate on a grid with 64,000 triangles is 0.16, which is two orders of magnitude greater than the value of 0.0014 from Table 1. P.4.

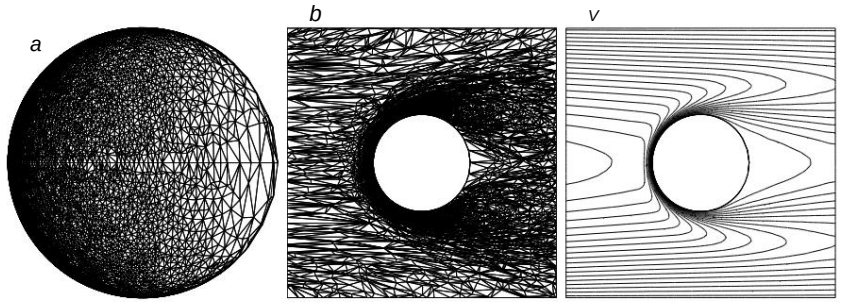
Algorithm 45 and the metric recovery method (A.3.1), (A.3.2) are successfully used for adaptive solution of boundary value problems with non self adjoint operators and anisotropic singularities of the solution, such as boundary layers [5, 9, 43, 66]. Moreover, quadratic extrapolation of piecewise-linear approximations of curvilinear boundaries (see § A.1) makes it possible to efficiently adapt the mesh around a curvilinear boundary. As an illustration, consider the convection–diffusion equation

$$\begin{aligned} &\ddot{y}\,\ddot{y}\,0.01\,\ddot{y}\,u + b\,\ddot{y}\,u = 0 \text{ in } \ddot{y}, \\ &u = g \text{ on } \ddot{y}in,\ \ddot{y}u \qquad (A.3.7) \\ &\frac{\partial u}{\partial \ddot{y}n} = 0 \text{ on } \ddot{y}out, \\ &u = 0 \text{ on } \ddot{y}\,\ddot{y} \setminus (\ddot{y}in\ \ddot{y}\,\ddot{y}out). \end{aligned}$$

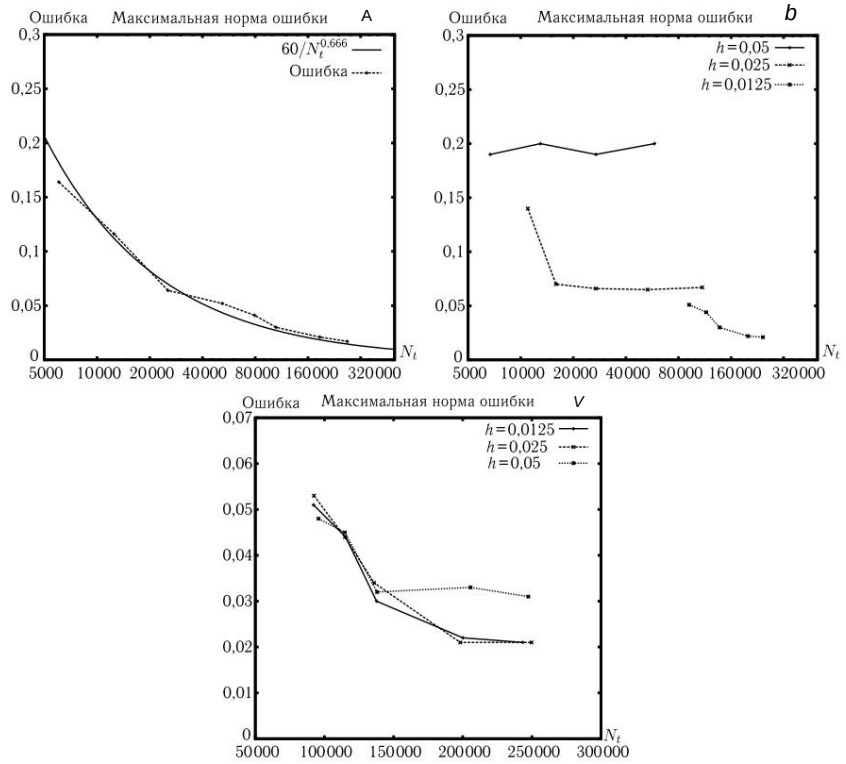
Here b = (1, 0, 0)T is the constant velocity field, ÿ = (0, 1)3 \ B0.5(0.18) is the cubic computational domain with the removed ball B0.5(r) of radius r = 0.18 centered at (0.5, 0.5, 0.5). The boundaries of the computational domain are the surface of a sphere, ÿ = ÿB0.5(0.18), and the surface of a cube. On the surface of the cube, we select the planes ÿin = {x ÿ ÿÿ: x = 0} and ÿout = {x ÿ ÿÿ: x = 1}. Finally, g(y, z) = 16 y(1 ÿ y)z(1 ÿ z) denotes the standard Poiseuille flow profile. The solution to problem (A.3.7) has a boundary layer along the leeward part of the spherical boundary ÿ and is very smooth in the

shadow zone behind the obstacle. Since the exact solution is unknown, in the experiments it was replaced by a piecewise linear finite element solution uÿ calculated on a very fine adaptive (quasi-optimal) grid containing more than 1.28 × 106 tetrahedra (see Fig. A.9). To generate this adaptive grid, the analytical representation ÿ ÿ was used. The first experiment (plot in Fig. A.10, a) confirms the asymptotic result (A.3.3) with uÿ instead of u. The error in the maximum ÿ2/3 norm almost coincides with the analytical curve 60 N In the second experiment (graph in Fig. A.10, b), the boundary ÿ is

approximated by a quasi-uniform triangulation ÿh. We present the maximum adaptive decision error rate as a function of N for three different values of h. The graph shows the saturation of this .



Rice. P.9. Adaptive mesh trace on the obstacle (a), mesh cut *(b)* and from the solution line uÿ *(c)* in the xy plane passing through the center of the obstacle



Rice. P.10. Convergence of the grid solution: in the domain with the analytical representation of the sphere ÿ (a), in the domains with three discrete models ÿ0.05, ÿ0.025, and ÿ0.0125 for the sphere ÿ *(b;* denoted by lines 0.05, 0.025, 0.0125), in domains with discrete models ÿ0.0125, ÿ0.025, and ÿ0.05 for the sphere ÿ *(c;* denoted by lines 0.0125, 0.025, 0.05)

errors due to inaccurate resolution of the curvilinear boundary. Note that the saturable error ÿh is almost inversely proportional to h2: ÿ0.05 = 0.2, ÿ0.025 = 0.067, and ÿ0.0125 = 0.021. This is probably related to the second-order approximation by the piecewise linear manifold ÿh for a smooth boundary ÿ.

The third experiment (graph in Fig. A.10, c) studies the effect of piecewise quadratic completion ÿh for ÿh on the accuracy of a discrete solution. Saturation errors are compared for three fixed surface grids with the same number of nodes: the first grid is a quasi-uniform triangulation ÿ0.0125, the second and third grids are obtained from the grid ÿ0.0125 by projecting its nodes onto the reconstructed piecewise-quadratic surfaces ÿ0.025 and ÿ0.05 respectively, i.e. they use edge information with a resolution of h = 0.025 and h = 0.05. For convenience, we also denote the second and third grids by ÿ0.025 and ÿ0.05. The dependence plots of the adaptive solution error in regions with fixed boundaries ÿ0.0125 and ÿ0.025 actually coincide, which leads to an approximate equality of saturation errors: ÿ0.025 ÿ ÿ0.0125 ÿ 0.021. Note that the saturation error ÿ0.025 in a region with a fixed boundary ÿ0.025 with a resolution h = 0.025 exceeds ÿ0.025 by more than three times. A coarser piecewise linear representation of the boundary with a resolution h = 0.05 demonstrates an even greater difference in saturation errors: ÿ0.05 = 0.2 for the region with the boundary ÿ0.05 and ÿ0.05 = 0.03 for the region with the boundary ÿ0 .05. Thus, piecewise quadratic completion makes it possible to significantly reduce the saturation error of grid adaptation in regions with curvilinear boundaries, which are represented by surface triangulations.

Clause 3.2. Tensor metric based on edge error estimates

The disadvantages of the adaptive procedure based on the metric obtained by restoring the Hessian of the grid function are large errors in restoring the Hessian in the vicinity of the singularities of the solution, the lack of control over the interpolation or discretization error, and the fact that it generates grids minimizing the Lp-norm of the error, 0 < p ÿ, while finite element solutions tend to minimize the energy norm of the error. Minimization of the energy error rate is of greater interest for elasticity problems, since it leads to a more accurate calculation of stresses in prefracture zones. The energy norm is related to the L2-norm *of the gradient* error; therefore, to optimize the finite element discretization error, one must be able to construct a metric that generates grids minimizing the L2-norm of the gradient error. The proposed method for constructing a tensor metric is based on the use of edge a posteriori estimates of the energy error

finite element solution. Its main advantages are its applicability to a wider class of functions, including functions with singularities, the possibility of a posteriori estimating the energy error rate, and the minimization of the error rate, which is natural for the finite element method. Consequently, an adaptive procedure using such a metric ensures the optimal rate of convergence of the grid solution to the differential solution and control of the discretization error uh ÿ u [25]. In § A.2 we considered a hierarchical method for estimating the a posteriori error based on the assumption (A.2.5) that the difference between the piecewise linear finite element solution uh and the

piecewise quadratic finite element solution Uh gives a good approximation to the error uh ÿ u . For an efficient approximate calculation of uh ÿ Uh on each triangle ÿ, a linear combination (A.2.9) of bubble functions bij associated with the edges eij of the triangle ÿ was taken. The coefficients of this combination satisfy the system (A.2.8):

$$\text{uh ÿ u ÿ ÿh ÿ} \qquad \text{eijÿÿ} \qquad \text{DQ, ijbij .} \tag{A.3.8}$$

We fix a triangle ÿ and introduce a vector d ÿ R3 with coefficients DQ, ij corresponding to the edges of this triangle. The gradient L2-norm of the a posteriori estimated error ÿh can be written as follows:

$$\text{ÿÿh2}_{L2(ÿ)} = \sum_{k=1}^{3} \text{dkÿbk}^2_{L2(ÿ)} = |ÿ|(B\ d,\ d), \tag{A.3.9}$$

where the summation goes over the edges e12, e23, e31, and B denotes a 3 × 3 symmetric positive definite matrix with entries

$$Bk,m = \int_{ÿ} \frac{1}{|ÿ|}\ ÿbk\ ÿbm\ dV.$$

Unfortunately, the error rate on a triangle is a number that does not provide enough information to determine the tensor metric M. In order to determine the tensor metric, we split the error into three edge components ÿk 0 such that

$$\text{ÿÿh2}_{L2(ÿ)} = |ÿ| \sum_{k=1}^{3} ÿk,\ ÿk = |dk|(B\ d,\ d) \qquad \sum_{k=1}^{3} |dk| \ ÿ1 \ . \tag{A.3.10}$$

The splitting of the error rate into three edge quantities ÿk has a simple justification. Such a choice additionally equally distributes the maximum discretization error rate on the edges of the triangle, and, consequently, on all edges of the grid [24, 26].

Three numbers ÿk allow us to determine three elements of the metric tensor M ÿ on the triangle ÿ. To do this, we introduce the quadratic function

$$1\ v2 = ÿ \sum_{k=1}^{3} ÿkbk$$

and denote its Hessian by H2. If det H2 = 0, we set

$$M = (\det|H2|)\ ÿ1/4\ |H2|, \qquad (A.3.11)$$

where |H2| is the spectral modulus of the matrix H2. Otherwise, we slightly increase the largest of the three edge errors so that the Hessian of the modified quadratic function v2 becomes a nondegenerate matrix. In practice, a 1% increase is sufficient.

The constructed metric M ÿ connects the L2-norm of the gradient error (A.3.10) with the geometric properties of the triangle ÿ. The following inequalities were proved in [24, 26]:

$$\frac{2}{5} V_M^2 \qquad ÿÿh2_{L2(ÿ)} \qquad VMÿ, ÿ\ |p\ M\ ÿ, ÿ|\ 2. \qquad (A.3.12)$$

The principle of error equipartition underlying adaptive meshing leads to the equipartition of the following geometric properties:

$$VMÿ, ÿ \quad ÿ\ |\ p\ M\ ÿ, ÿ|^{2\ ÿ} \quad \frac{1}{N(ÿh)} ÿÿh|M. \qquad (A.3.13)$$

Thus, we need to build grids with the same areas of triangles and the same perimeters of triangles measured in the tensor metric M, which is composed of piecewise constant metrics M ÿ. According to (A.3.13), an M-quasi-uniform mesh ÿh containing N triangles ensures the asymptotically optimal decay rate of the gradient error estimate

$$ÿÿhL2(ÿ)\ ÿ\ N\ ÿ1/2 \quad .$$

In the case of tetrahedral meshes, the derivation of the metric repeats calculations (A.3.9)–(A.3.11) with the correction that the tetrahedron has six edges, and the scaling in (A.3.11) is changed: = (det|H2|) ÿ1/5

$$M \quad |H2|. \qquad (A.3.14)$$

As for triangular meshes, M-quasi-uniform meshes containing N tetrahedra provide an asymptotically optimal decay rate for the gradient error estimate [24, 26]:

$$ÿÿhL2(ÿ)\ ÿ\ N\ In^{\ ÿ1/3} \quad .$$

computational practice, the piecewise constant tensor metric is replaced by a continuous metric, which ensures faster convergence of the algorithm 45. The continuous tensor metric is formed based on the values at the grid nodes and linear interpolation

inside each element. The value at the node v is taken equal to the value of the metric on the simplex from the set ÿ(v) that has the largest determinant.

T a b l e A.5

Energy norms of the error of the adaptive finite element solution ÿ(uh ÿ u)L2(ÿ) of the crack problem, for two methods for constructing the tensor metric

Method ... metric recovery ... N	1000 4000 16000 64000
(A.3.1), (A.3.2)	0.11 0.053 0.026 0.013
(A.2.8), (A.3.9)–(A.3.11)	0.11 0.053 0.027 0.015

T a b l e A.6

Energy norms of error and error estimates for an adaptive finite element solution for the crack problem

N error	1000 4000 16000 64000
ÿ(uh ÿ u)L2 0.11	0.053 0.027 0.015 0.10 0.051
ÿÿhL2	0.026 0.013

As the experiment on constructing an adaptive grid in the crack problem (A.3.6) shows, both methods of constructing the metric (using the restoration of the grid Hessian and using the a posteriori hierarchical error estimate) provide the asymptotically optimal (N ÿ1/2) rate of fall of the energy norm ki errors. Vtab. A.5 shows the energy norms of the finite element error for two methods of constructing the metric: a) by formulas (A.3.1), (A.3.2) and b) by formulas (A.2.8), (A.3.9)–(A.3.11). The data in the table confirm a twofold drop in the energy norm of the finite element error with a fourfold increase in the number of grid elements, which confirms the asymptotic behavior of N ÿ1/2. error rates due to the known a posteriori estimate (A.3.8). Vtab. In Section 6, we compare the energy norms of the exact error uh ÿ u and the a posteriori estimated error ÿh. Table ·

values demonstrate the closeness of the norms of both errors. Controlling the error of a finite element solution in engineering calculations is an important property of this computational

technologies.

Subject index

Bibliography

1. B. N. *Azarenok,* Variational methods for constructing structured grids and their applications to gas dynamics: Dis... doctorate. Phys.-Math. Sciences. — M.: VTs RAN, 2009.
2. V. N. *Apanovich* and T. A. *Dolgova ,* Practical convergence of the method of external finite element approximations in solving three dimensional problems of elasticity theory. — Dep. in VINITI 17.03.93, No. 645-V.
3. V. G. *Boltyanskii* and V. A. *Efremovich ,* Visual topology. — M.: Nauka, 1983.
4. S. N. *Borovikov ,* I. E. *Ivanov ,* and I. A. *Kryukov ,* “Construction of a constrained Delaunay tetrahedrization for bodies with curvilinear boundaries,” Zh. Vychisl. math. and mat. physical 2005. V. 45. S. 1407–1423.
5. Yu . V. *Vasilevsky* and K. N. *Lipnikov ,* “An adaptive algorithm for constructing quasi-optimal grids,” Zh. Vychisl. math. and mat. physical 1999. V. 39, No. 9. S. 1532–1551.
6. Yu. V. *Vasilevskii* and A. *Aguzal ,* “Unified Asymptotic Analysis of Interpolation Errors on Optimal Grids,” Dokl. 2005. V. 405, No. 3. S. 1–4.
7. *Vasilevsky* Yu. V., *Vershinin* A. V., *Danilov* A. A., *Plenkin* A. V. Technology for constructing tetrahedral meshes for domains specified in CAD // Matrix Methods and Technologies for Solving Large Problems / Ed. E. E. Tyrtyshnikova. — M.: INM RAN, 2005. — S. 21–32. 8. *Vasilevsky* Yu . V. Parallel technologies for solving boundary value problems: Dis... doc. Phys.-Math. Sciences. - M.: INM RAN, 2006.
9. Yu . V. *Vasilevskii* and K. N. *Lipnikov ,* “Using the Hessian Restoration Technique in the Construction of Adaptive Grids,” Vopr. atom. in science and technology: ser. Mat. physical modeling. processes. 2006. Issue. 3. P. 37–53.
10. V. A. *Garanzha ,* Discrete Curvatures, Quasi-Isometric Mappings, and Quasi-Optimal Computational Grids: Dis... doctorate. Phys.-Math. Sciences. — M.: VTs RAN, 2011.
11. A. A. *Danilov ,* “Technology for constructing unstructured grids and monotonic discretization of the diffusion equation,” Cand. Phys.-Math. in UK. - M.: INM RAN, 2010.
12. A. A. *Danilov ,* “Construction of tetrahedral meshes for domains with given surface triangulations,” Computational Methods, Parallel Computations, and Information Technologies, vol. — M.: MGU, 2008. — S. 119–130.
13. N. *Dolbilin ,* “Three theorems on convex polytopes,” Kvant. 2001. No. 5, pp. 7–12.
14. *Ivanenko* S. A. Adaptive-harmonic grids. — M.: VTs RAN, 1997.
15. V. D. *Liseikin ,* Yu. V. *Likhanova ,* and Yu. I. *Shokin ,* Difference grids and coordinate transformations for the numerical solution of singularly perturbed problems. — Novosibirsk: Nauka, 2007. 16. *Liseikin* VD , *Shokin* Yu. I., *Vaseva* IA , *Likhanova* Yu. V. Technology of constructing difference grids. — Novosibirsk: Nauka, 2009. 17. *Mishchenko* A. S., *Fomenko* A. T. A course in differential geometry and pology. - M .: Factorial Press, 2000.
18. *Skvortsov* A. V. Delaunay triangulation and its application. - Tomsk: Publishing House Vol. un-ta, 2002. 19. Electronic resource: Open CASCADE Technology: http://www.opencascade.org/
20. Electronic resource: The Common Geometry Module: http:// cubit.sandia.gov/cgm.html 21. Electronic resource: The Common Geometry Module, Argonne (CGMA): http://trac.mcs.anl.gov/projects/ITAPS/wiki/CGM
22. Advances in grid generation / Ed. O. Ushakova. — NY: Nova Science Publishers, 2005.
23. *Agouzal* A., *Lipnikov* K., *Vassilevski* Y. Adaptive generation of quasioptimal tetrahedral meshes // East-West J. Numer. Math. 1999. V. 7. P. 223–244.
24. *Agouzal* A., *Lipnikov* K., *Vassilevski* Y. Hessian-free metric-based mesh adaptation via geometry of interpolation error, Comput. math. and mat. physical 2010. V. 50, No. 1. S. 131–145.
25. *Agouzal* A., *Lipnikov* K., *Vassilevski* Y., “On optimal convergence rate of finite element solutions of boundary value problems on adaptive anisotropic meshes,” Math. Comput. Simul. 2011. V. 81. P. 1949–1961.
26. *Agouzal* A., *Vassilevski Yu.* Minimization of gradient errors of piecewise linear interpolation on simplicial meshes // Comput. Meth. Appl. Mech. a. Eng. 2010. V. 199. P. 2195–2203.
27. *Apel* T. Anisotropic finite elements: Local estimates and applications. — Stuttgart: Teubner, 1999.
28. *Arnold* D., *Mukherjee* A. *Pouly* L. Locally adapted tetrahedral meshes using bisection // SIAM J. Sci. Comput. 2000. V. 22, No. 2. P. 431–448.
29 *Auricchio* F., *Beirao da Veiga* F. , *Hughes* T. J. R. *et al.* Isogeometric collocation for elastostatics and explicit dynamics // Comput. Meth. Appl. Mech. a. Eng. Dec 1, 2012 V. 249–252. P. 2–14.
30. *Bank* R. E. PLTMG: a software package for solving elliptic partial differential equations, users' guide 6.0. — Philadelphia: SIAM, 1990.
31. *Bansch* E Local mesh refinement in 2 and 3 dimensions // IMPACT of Comput. in Sci. a. Eng. 1991. V. 3. P. 181–191.
32. *Bazilevs* Y., *Calo* V. M., *Cottrell* J. A. *et al.* Isogeometric analysis using T-splines // Comput. Meth. Appl. Mech. a. Eng. 2010, 1 Jan. V. 199, is. 5–8. P. 229–263.
33. *Borden* M. J., *Verhoosel* C. V., *Scott* M. A. *et al.* A phase-field description of dynamic brittle fracture // Comput. Meth. Appl. Mech. a. Eng. 2012, 1 Apr. v. 217–220. P. 77–95.
34. *Borouchaki* H., *Hecht* F., *Saltel* E., *George P.-L.* Reasonably efficient Delaunay based mesh generator in 3 dimensions // Proc. 4th Int. Mesh Roundtable. - Albuquerque, 1995. - P. 3-14.

35. *Buscaglia* G. C., *Dari* E. A. Anisotropic mesh optimization and its application in adaptivity // Int. J. Numer. Meth. engng. 1997. V. 40. P. 4119–4136.
36. *Carey* G. F. Computational grids: generation, adaptation, and solution strate. gies. — Washington: Taylor & Francis, 1997.
37. *Chugunov* V., *Vassilevski Yu.* Parallel multilevel data structures for a non-conforming finite element problem on unstructured meshes // Russ. J. Numer. Anal. Math. Modelling. 2003. V. 18, No. 1. P. 1–11.
38. *Chugunov* V., *Svyatski* D., *Tyrtyshnikov* E., *Vassilevski Yu.* Parallel iterative multilevel solution of mixed finite element systems for scalar equations // Concur. a. Comput.: Pract. a. Expert. 2006. V. 18, No. 5. P. 501–518.
39. *Ciarlet* P. The finite element method for elliptic problems. - Amsterdam: North-Holland, 1978.
40. *Danilov* A. Unstructured tetrahedral mesh generation technology // Comput. math. and mat. physical 2010. V. 50, No. 1. S. 146–163.
41. *Deuflhard* P., *Leinen* P., *Yserentant* H. Concepts of an adaptive hierarchical finite element code // IMPACT. 1989. V. 1. P. 3–35.
42. *Du* Q., *Wang* D. Recent progress in robust and quality Delaunay mesh generation // J. Comput. a. Appl. Math. 2006. V. 195. P. 8–23.
43. *Dyadechko* V., *Lipnikov* K., *Vassilevski Yu.* Hessian based anisotropic mesh adaptation in domains with discrete boundaries // Russ. J. Numer. Anal. Math. Modelling. 2005. V. 20, No. 4. P. 391–402.
44. *Elguedj* T., *Hughes* T. J. R. Isogeometric analysis of nearly incompressible large strain plasticity, Comput. Meth. Appl. Mech. a. Eng. Jan 1, 2014 V. 268. P. 388–416.
45. *Eymard* R., *Gallou¨et* T. R., *Herbin* R. The finite volume method // Handbook of Numerical Analysis. V. 7/Ed. PG Ciarlet and JL Lions. - North Holland, 2000. - P. 713-1020.
46. *Farrell* P. E., *Maddison* J. R. Conservative interpolation between volume meshes by local Galerkin projection // Comput. Meth. Appl. Mech. a. Eng. 2011. V. 200. P. 89–100.
47. *Fary* I. On straight line representations of planar graphs // Acta Sci. Math. 1948. V. 11. P. 229–233.
48. *Foster* I. Designing and building parallel programs. — NY: Addison-Wesley, 1995.
49. *Freitag* L. A., *Ollivier-Gooch* C. Tetrahedral mesh improvement using swap ping and smoothing // Int. J. Numer. Meth. Eng. 1997. V. 40. P. 3979–4002.
50. *Frey* P. J., *George* P. L. Mesh generation: application to finite elements. — Oxford: Hermes Science Publishing, 2000.
51. V. L. *Garanzha,* “Discrete extrinsic curvatures and approximation of surfaces by polar polyhedra,” Comput. math. and mat. physical 2010. V. 50, No. 1. S. 71–98.
52. *Garimella* R. V. Anisotropic Tetrahedral Mesh Generation. — Rensselaer Polytechnic Institute, Report 1998-31, PhD Thesis. — 1998.
53. *Garimella* R. V., *Swartz* B. K. Curvature estimation for unstructured triangulations of surfaces. - Los Alamos Report LA-UR-03-8240; http://math.lanl.gov/Research/Publications/Docs/garimella-2003-curvature.pdf.
54. *George* P. L., *Borouchaki* H. Delaunay triangulation and meshing: application to finite elements. - Paris: Editions Hermes, 1998.

55. *George* P. L., *Borouchaki* H., *Saltel* E. 'Ultimate' robustness in meshing an arbitrary polyhedron, Int. J. Numer. Meth. Eng. 2003. V. 58. P. 1061–1089.
56. Handbook of grid generation/ Ed. J. Thompson, B. Soni, N. Weatherill. — Boca Raton, FL: CRC Press, 1999.
57 *Hoppe* R., *Iliash* Yu., *Kuznetsov Yu. et al.* Analysis and parallel implementation of adaptive mortar element methods // East-West J. Numer. Math. 1998. V. 6. P. 223–248.
58. *Hughes* T. J. R., *Cottrell* J. A., *Bazilevs* Y. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement, Comput. Meth. Appl. Mech. a. Eng. (Elsevier). Oct 1, 2005 V. 194. P. 4135–4195.
59. *Ivanenko* S. A. Selected chapters on grid generation and applications. — Moscow: Dorodnicyn Computing Center of the Russian Academy of Sciences, 2004.
60. *Klingner* B. M., *Shewchuk* J. R. Aggressive tetrahedral mesh improvement // Proc. 16th Int. Meshing Roundtable/Ed. M. Brewer, D. Marcum. - Berlin-Heidelberg: Springer-Verlag, 2007. - P. 3-23.
61. *Knupp* P., *Steinberg* S. Fundamentals of grid generation. — Boca Raton: CRC Press, 1994.
62. *Knupp* P. Algebraic mesh quality metrics // SIAM J. Sci. Comput. 2001. V. 23. P. 193–218.
63. *Kuratowski* K. Sur le probleme de courbe gauches en topologie // Fund. Math. 1930. V. 15. P. 271–283.
64. *Lachaud J.-O.* Topologically defined iso-surfaces // Proc. 6th DGCI. — Berlin: Springer-Verlag, 1996, pp. 245–256.
65. *Levenberg* K. A Method for the Solution of Certain Non-Linear Problems in Least Squares, Quart. Appl. Math. 1944. V. 2. P. 164–168.
66. *Lipnikov* K., *Vassilevski Yu.* Parallel adaptive solution of 3D boundary value problems by Hessian recovery // Comput. Meth. Appl. Mech. a. Eng. 2003. V. 192, nos. 11–12. P. 1495–1513.
67. *Lipnikov* K., *Vassilevski Yu.* On control of adaptation in parallel mesh generation // Eng. with Comput. 2004. V. 20, No. 3. P. 193–201.
68. *Liseikin* V. Grid generation methods. — Berlin: Springer-Verlag, 1999.
69. *Liseikin* V. A computational differential geometry approach to grid genera tion. — Berlin: Springer-Verlag, 2006.
70. *McIvor* A. M., *Valkenburg* R. J. A comparison of local surface geometry estimation methods // Machine Vision a. Appl. 1997. V. 10. P. 17–26.
71. *Meyer* M., *Lee* H., *Barr* A. H., *Desbrun* M. Generalized barycentric coordinates on irregular polygons // J. Graph. tools. 2002. V. 7, No. 1. P. 13–22.
72. *Misztal* M. K., *Baerentzen* J. A., *Anton* F., *Erleben* K. Tetrahedral mesh improvement using multi-face retriangulation // Proc. 18th Int. Meshing Roundtable/ Ed. BW Clark. - Berlin-Heidelberg: Springer-Verlag, 2009. - P. 539-555.
73. *Mokhtarian* F., *Khalili* N., *Yuen* P. Curvature computation of free-form 3-D meshes at multiple scales, Comput. vision a. image understanding. 2001. V. 83. P. 118–139.
74. *Nicolaides* R. A. Direct discretization of planar div-curl problems // SIAM. J. Numer. Anal. 1992. V. 29. P. 32–56.

75. *Pothen* A., *Simon* H., *Liou* K. Partitioning sparse matrices with eigenvectors of graphs // SIAM J. Math. Anal. Appl. 1990. V. 11. P. 430–452.

76 Proc. 16th Int. Meshing Roundtable/Ed. M. Brewer, D. Marcum. — Berlin-Heidelberg: Springer-Verlag, 2007.

77 Proc. 18th Int. Meshing Roundtable/Ed. BW Clark. - Berlin-Heidelberg: Springer-Verlag, 2009.

78. *Rivara* M. Mesh refinement processes based on the generalized bisection of simplexes // SIAM J. Numer. Anal. 1984. V. 21. P. 604–613.

79. *Rivara* M. Selective refinement/derefinement algorithms for sequences of nested triangulations, Int. J. Numer. Meth. Eng. 1989. V. 28. P. 2889–2906.

80. *Rado* T. Uber den Begriff der Riemannschen Fl Mathem. (Szeged). ¨achen // Acta Scient. 1925. V. 2. P. 101–121.

81. *Shewchuk* J. R. Constrained Delaunay tetrahedralizations and provably good boundary recovery // Proc. 11th Int. Mesh Roundtable. - Ithaca, 2002. - P. 193-204.

82. *Tautges* T. J. The Common Geometry Module (CGM): A Generic, Extensible Geometry Interface // Eng. with Comput. 2001. V. 17. P. 299–314.

83. *Verfurth R* ¨A review of a posteriori error estimation and adaptive mesh-refinement techniques. - Stuttgart: Wiley-Teubner, 1996.

84. *Verhoosel* C. V., *Scott* M. A., *Borden* M. J. *et al.* Discretization of Higher Order Gradient Damage Models Using Isogeometric Finite Elements // Damage Mechanics of Cementitious Materials and Structures. — NY: Wiley, 2012. — P. 89–120.

85. *Williams* R. Performance of dynamic load balancing algorithms for unstructured mesh calculations // Concurrency: Practice and experience. 1992. V. 3. P. 457–481.

86. *Zavattieri* P., *Dari* E., *Buscaglia* G. Optimization strategies in unstructured mesh generation // Int. J. Numer. Meth. Eng. 1996. V. 39. P. 2055–2071.

TABLE OF CONTENTS

CAE Fidesys

STRENGTH ENGINEERING ANALYSIS
APPLICATION PACKAGE **A trial version** (1 month) **of**

the product and a user manual with step-by-step examples
are available at http://www.cae-fidesys.com Tel.: +7 (495)
930-87-53, e -mail:
contact@cae-fidesys.com

DEAR COLLEAGUES!

We offer you cooperation in a number of areas: 1. If you have an industrial module for an engineering analysis package, then we offer its integration with CAE Fidesys for their joint promotion to the market. 2. If you have a scientific software product (for any type of engineering analysis) or you are working on its creation, then we are ready to assist in its commercialization by integrating with the Fidesys CAE package.

3. If you are cooperating with industrial enterprises, then we offer to work with you on the basis of the Fidesys CAE package to create a new software product - a specialized industry solution - and transfer it to the enterprise (industry). Scientific and methodological guidance remains with you and / or representatives of the enterprise (industry). 4. If you are developing a new direction in engineering analysis, then we are ready to consider algorithmization and software implementation and, if necessary, act as an "industrial partner".

We are also interested in the industrial use of modern numerical methods, for example, the discontinuous Galerkin method, isogeometry cal analysis.

5. If you are interested in introducing an innovative component into the educational process at your university, then we suggest creating (deploying) a “Virtual Laboratory for Strength Engineering Analysis Courses” as a private cloud of your university. This will allow teachers to conduct classes remotely, and students to use tablets or even smartphones when teaching. In this case, we can offer the joint development of laboratory work.

The specified private cloud can be supplemented with the ability to use the Fidesys CAE package in the work of the engineering center of your university. In addition, when a university purchases one commercial license, we are ready to supply up to 5 educational licenses in an industrial configuration. At the same time, we propose the creation of joint training centers with your university for industrial users of CAE Fidesys.