Chapter 4
The Discretization Process

Abstract This chapter introduces the different steps of the discretization process, which include: (i) modeling of the geometric domain and the physical phenomena of interest; (ii) discretization of the modeled geometric domain into a grid or mesh that forms the computational domain (this process, also known as meshing or domain discretization, results in a set of non-overlapping elements, denoted also by cells, that cover the computational domain); (iii) numerical or equation discretization that transforms the set of conservation partial differential equations governing the physical processes into an equivalent system of algebraic equations defined over each of the elements of the computational domain; and (iv) the solution of the resulting set of equations using an iterative solver to yield an intermediate or final solution field. Throughout the chapter, computer implementation issues are introduced.

4.1 The Discretization Process

The numerical solution of a partial differential equation consists of finding the values of the dependent variable ϕ at specified points from which its distribution over the domain of interest can be constructed. These points are called *grid elements*, or *grid nodes and result from the discretization of original geometry into a set of non overlapping discrete elements, a process known as meshing. The resulting nodes or variables* are generally positioned at *cell centroids* or at vertices depending on the adopted discretization procedure. In all methods the focus is on replacing the continuous exact solution of the partial differential equation with discrete values. The distribution of ϕ is hence *discretized*, and it is appropriate to refer to this process of converting the governing equation into a set of algebraic equations for the discrete values of ϕ as the *discretization process* and the specific methods employed to bring about this conversion as the *discretization methods*. The discrete values of ϕ are typically computed by solving a set of *algebraic equations* relating the values at neighboring grid elements to each other; these discretized or

F. Moukalled et al., *The Finite Volume Method in Computational Fluid Dynamics*,
Fluid Mechanics and Its Applications 113, DOI 10.1007/978-3-319-16874-6_4

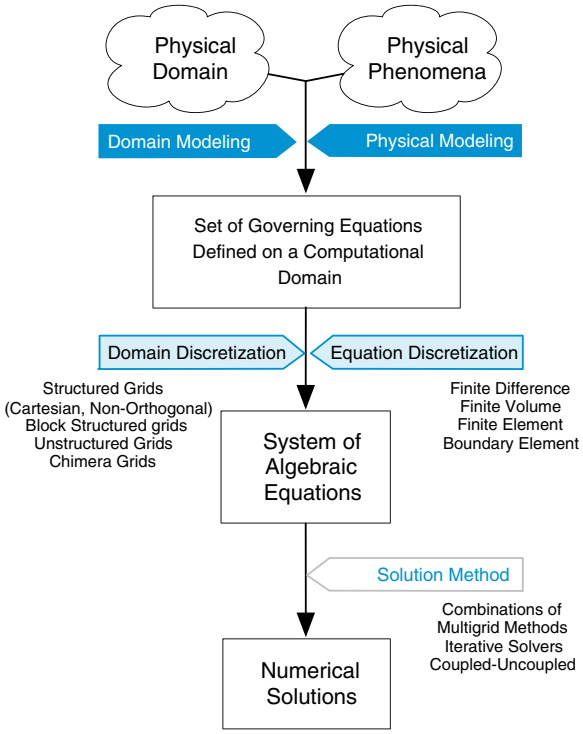


Fig. 4.1 The discretization process

algebraic equations are derived from the conservation equation governing ϕ. Once the values of ϕ are computed, the data is processed to extract any needed information. The various stages of the discretization process are illustrated in Fig. 4.1.

Figure 4.2 shows the process applied to the study of heat transfer from a microprocessor connected to a heat sink with a copper base that acts as a heat spreader. The example of Fig. 4.2 will be used to present various concepts related to the discretization process. Since the intention is to introduce a numerical technique for solving the physical processes of interest and since the method has to be implemented in a computer program, the discretization process will be explained along that spirit. For example, reference will be made to how to store values in a code as related to interior elements, boundary elements, variables, etc. Throughout the book a Matlab® based program denoted by "uFVM" will be used as a development vehicle to present various details relating to the implementation of these methods. Furthermore OpenFOAM®, a very popular finite volume-based open source code will also be presented both from a user and a developer perspective, again with the aim of moving from the numerics to the implementation details.

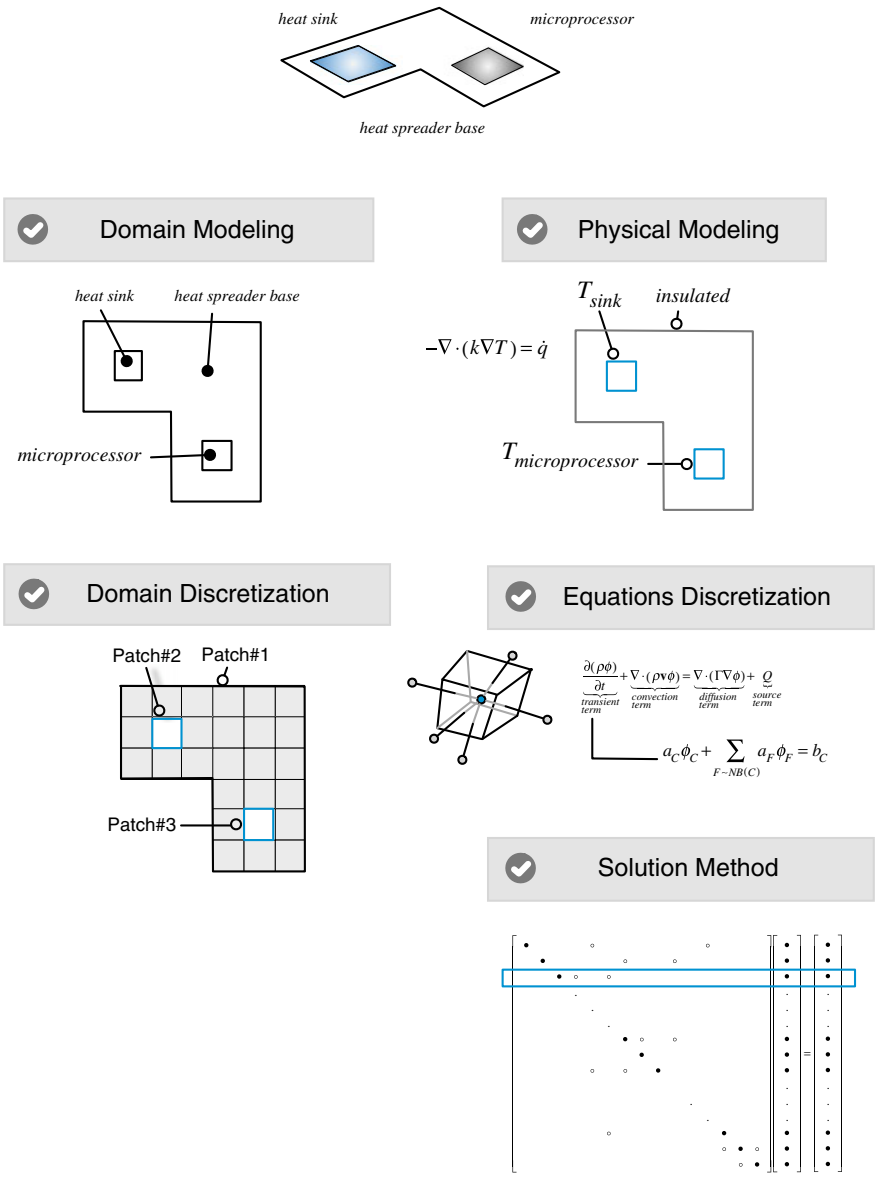


Fig. 4.2 An illustration of the discretization process

4.1.1 Step I: Geometric and Physical Modeling

Modeling of physical phenomena is in a way at the heart of the scientific enterprise. A physical phenomenon cannot generally be considered as understood unless it can

be mathematically formulated, and this formulation tested and validated. For our purpose two levels of modeling are performed, one in relation to the geometry of the physical domain and a second in relation to the physical phenomena of interest. At both levels details that are neither relevant nor of interest are ignored or simplified. For example a three dimensional domain could be turned into a two dimensional depiction, or symmetry can be taken into account to decrease the size of the study domain. In some cases, physical components may be removed and replaced with appropriate mathematical representations.

In the example of Fig. 4.2 a microprocessor is connected to a heat sink with a copper base that acts as a heat spreader. A first model of this system simplifies both its physics and its geometry. The heat sink and processor are replaced by boundary conditions that specify the estimated temperature of the heat sink and the expected operating temperature of the processor, respectively. The physical domain is modeled as a two dimensional computational domain since the temperature variation through the thickness of the heat sink will be minimal. For a steady state solution of the heat flow and temperature distribution in the copper base, only heat conduction is considered. The result of the modeling process is a system of linear (or non-linear if k depends on T) partial differential equations, which in this case involves a simplified form of the energy equation given by

$$-\nabla \cdot (k\nabla T) = \dot{q} \tag{4.1}$$

where k is the conductivity of the heat spreader base, and $\dot{q}$ is the heat source/sink per unit volume.

4.1.2 Step II: Domain Discretization

The geometric discretization of the physical domain results in a mesh on which the conservation equations are eventually solved. This requires the subdivision of the domain into discrete non-overlapping cells or elements that completely fill the computational domain to yield a grid or mesh system. This is accomplished by a variety of techniques resulting in a wide range of mesh types. These meshes are classified according to several characteristics: structure, orthogonality, blocks, cell shape, variable arrangement, etc. In all cases the mesh is composed of discrete elements defined by a set of vertices and bounded by faces. For the mesh to be a useful platform for equation discretization, information related to the topology of the mesh elements, in addition to some derived geometric information, are needed. These include element to element relations, face to elements relations, geometric information of the surfaces, element centroid and volume, face centroid, area and normal direction, etc. This information is usually inferred from the basic mesh data. For certain mesh topologies, details about the mesh can be easily deduced from the element indices as in structured grids, while for others it has to be constructed and stored in lists for later retrieval, as is the case with unstructured grids.

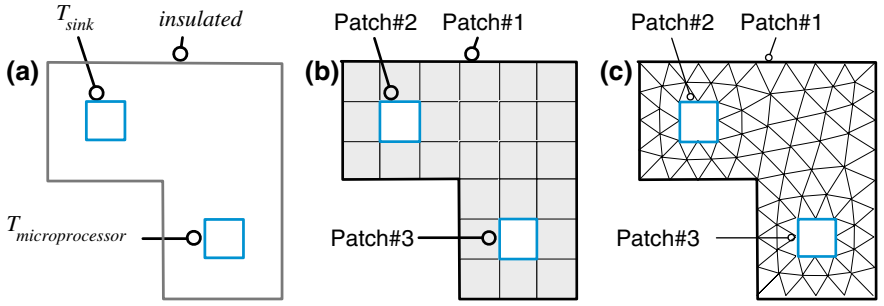


Fig. 4.3 **a** Computational domain; **b** computational mesh (*quadrilateral*); **c** computational mesh (*triangular*)

Consider the simple domain shown in Fig. 4.3a. The domain consists of a volume (area for the two dimensional case) and boundaries that account for the heating or cooling of the microprocessor, a heat sink, and the heat spreader base. The domain is shown discretized with a simple mesh in Fig. 4.3b. The mesh boundary is divided into three patches of boundary faces that are assigned numbers, i.e., Patch#1, Patch#2, and Patch#3. These patches are used to define the physical boundary conditions for the problem at hand. The mesh consists of 25 non-overlapping elements whose geometry is defined by 40 points (vertices of the cells). The elements are also bounded by 66 faces (lines in a two dimensional case), 34 of which are interior faces. The algebraic equations that result from the discretization of the governing equations, as will be explained in step III, are described for each element in the computational domain with the solution expressed as an element field with values defined at the centroid of each element. In this example the elements have a square shape, though other shapes could have been used (e.g., triangular elements, as shown in Fig. 4.3c).

The mesh can be described from different perspectives. At the most elementary level it is a list of **vertices** or **points** representing locations in one dimensional, two dimensional, or three dimensional spaces. The mesh also represents the discretized domain subdivided into non-overlapping elements, which can be of arbitrary convex polyhedral shapes. Elements are completely bounded by faces that are generally shared by neighboring elements, except at the boundaries. Elements can be defined either in terms of the points that delimit them or in terms of the faces that bound them. The mesh faces, which are stored in a list, are of two types: (i) interior faces that are shared by (or connect) two elements, and (ii) boundary faces that coincide with the domain boundary; these boundary faces have only one contiguous element. While interior faces are derived from information related to the element topology, it is essential to provide boundary faces as they define the domain physical boundary. In two dimensions faces are described in terms of their defining points. In three dimensions the defining points describe edges that bound the face. The direction of the normal to an interior face is usually defined based on the topology of the neighboring elements. On the other hand, the direction of the

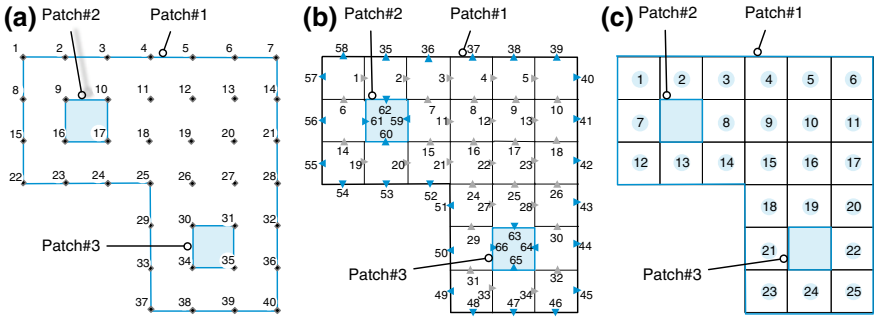


Fig. 4.4 **a** Mesh vertices, **b** faces, and **c** elements

normal to a boundary face always points outward of the domain. Figure 4.4 shows some of the components (vertices, faces, and elements shown in Fig. 4.4a–c respectively) of a mesh. Furthermore the boundary faces are organized into lists of faces based on the boundary patch to which they belong.

4.1.3 Mesh Topology

During discretization, the partial differential equations are integrated over each element in the mesh resulting in a set of algebraic equations with each one linking the value of the variable at an element to the values at its neighbors. The algebraic equations are then assembled into global matrices and vectors and the coefficients of every equation stored at the row and column locations corresponding to the various element indices. The integration of the equations over each element is referred to as local assembly while the construction of the overall system of equations from these contributions is referred to as global assembly. Thus while the discretization of the equations is derived in terms of neighbor elements, the assembly of the equations in the global matrix accounts for the actual indices of the elements. This procedure will be detailed in later chapters, however the enabling ingredients of this procedure are briefly introduced next at their most elementary level, which is in the form of topological information about elements, faces, and vertices that are represented in terms of connectivity lists.

Element connectivity relates the local assembly matrix to the global matrix so that the equations formed for one element are consistent with the equations formed for the other elements in the computational domain. Generally element to element, element to face, and element to vertex connectivities are setup. These relate the element to the neighboring elements, bounding faces, and defining vertices, respectively. Considering Fig. 4.4, the connectivity for element 9 is shown in Fig. 4.5.

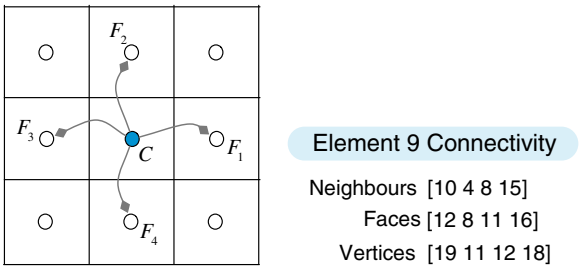


Fig. 4.5 Element connectivity

Generally for arbitrary elements it is more efficient to assemble flux terms by looping over faces. In this case it is essential that information about the face element neighbors be readily available; this is defined in the **Face connectivity**. For faces, information about elements sharing the face is stored for use during computations. The orientation of the face is such that the normal vector to the face points from one element denoted by *element 1* or *owner* to the second element denoted by *element 2* or *neighbor*. Boundary faces bound only one element, defined as element 1, thus the normal vector of boundary faces is always oriented outside of the domain. The connectivity for Face 12 is shown in Fig. 4.6.

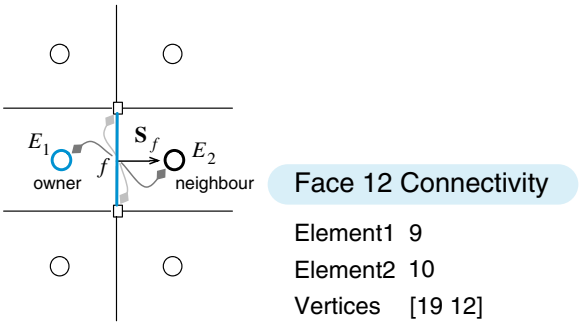


Fig. 4.6 Face connectivity

Vertex connectivity is useful for post processing and for gradient computation. As shown in Fig. 4.7, generally it involves the lists of elements and faces that share the vertex.

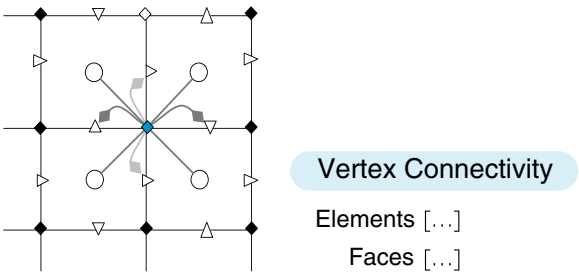


Fig. 4.7 Vertex connectivity

The mapping between local and global indices is briefly illustrated in Fig. 4.8 for a mesh of five elements.

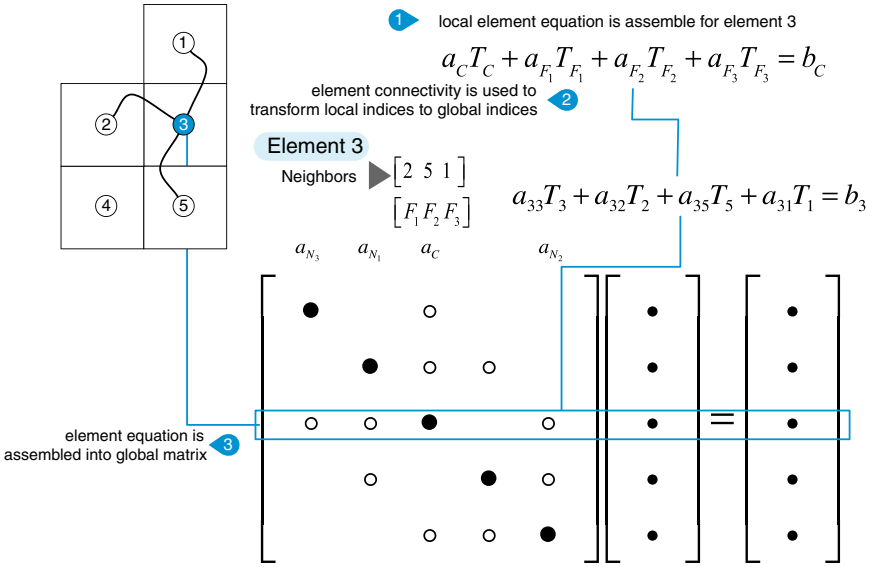


Fig. 4.8 Local element matrix assembly into global matrix

Example 1

For the mesh shown below, derive the element connectivity and represent it in a global matrix (Fig. 4.9)

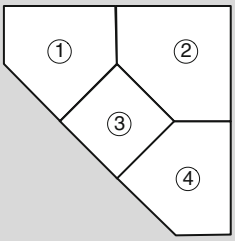


Fig. 4.9 Mesh for example 1

Solution

In this mesh element counting starts from 1. The connectivity for the various elements are given by

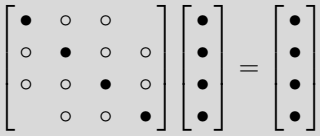
$1 \rightarrow 2, 3$

$2 \rightarrow 1, 3, 4$

$3 \rightarrow 1, 2, 4$

$4 \rightarrow 2, 3$

this can be represented in a global matrix as



(4.2)

4.1.4 *Step III: Equation Discretization*

In step III, the governing partial differential equations, are transformed into a set of algebraic equations, one for each element in the computational domain. These algebraic equations are then assembled into a global matrix and vectors that can be expressed in the form

$$\mathbf{A}[T] = \mathbf{b} \tag{4.2}$$

where the unknown variable T is defined at each interior element and at the boundary of the computational domain. Boundary values for T are generally obtained from the specified boundary conditions. To this end an element field has to be defined for T, and generally for each governing equation.

As schematically depicted in Fig. 4.10, the **element field** consists of an array of values defined at the centroid of each element, designated by the interior element field, which is represented by one array of size equal to the total number of interior and boundary elements.

The equation discretization step is performed over each element of the computational domain to yield an algebraic relation that connects the value of a variable

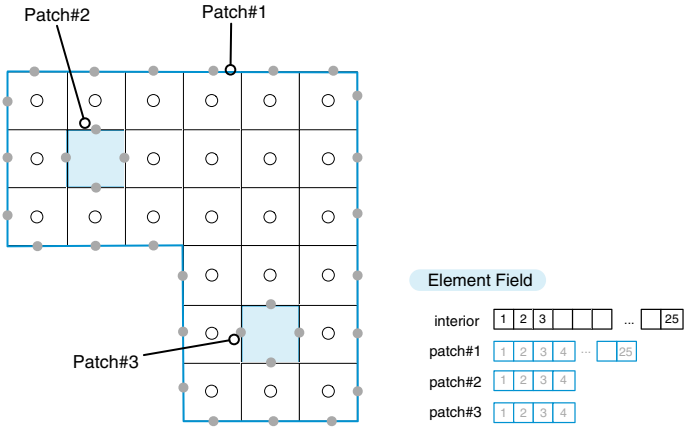


Fig. 4.10 Element field

in an element to the values of the variable in the neighboring elements. This algebraic equation is derived by discretizing the differential equation, which for the example considered is the energy equation written in terms of temperature T, (i.e., T is the unknown variable). As shown below, in the finite volume method the discretization of the equation is performed by first integrating the differential equation over a control volume or cell to obtain a semi discretized form of the equation and then approximating the variation of the dependent variable between grid elements through imposed profiles to obtain the final discretized form. The fact that only a few grid elements participate in a given discretization equation is a consequence of the piecewise nature of the chosen profiles. The value of T at a grid point thereby influences the distribution of T only in its immediate neighborhood. As the number of grid elements increases, the solution of the discretized equations is expected to approach the exact solution of the corresponding differential equation. This follows from the consideration that, as the grid elements get closer together, changes in T between neighboring grid elements become small, and then the actual details of the profile assumption become unimportant.

For a given differential equation, the possible discretization equations are by no means unique, although all types of discretization techniques in the limit of a very large number of grid elements are expected to give the same solution. The different types arise from the differences in the profile assumptions and the methods of derivation.

As an example of the equation discretization step using the finite volume method, the discretized form of the energy equation over the control volume C shown in Fig. (4.11) is sought. The process starts by integrating Eq. (4.1) over element C that enables recovering its integral balance form, which was described in Chap. 3, as

$$-\iint_{V_C} \nabla \cdot (k \nabla T) dV = \iint_{V_C} \dot{q} dV \tag{4.3}$$

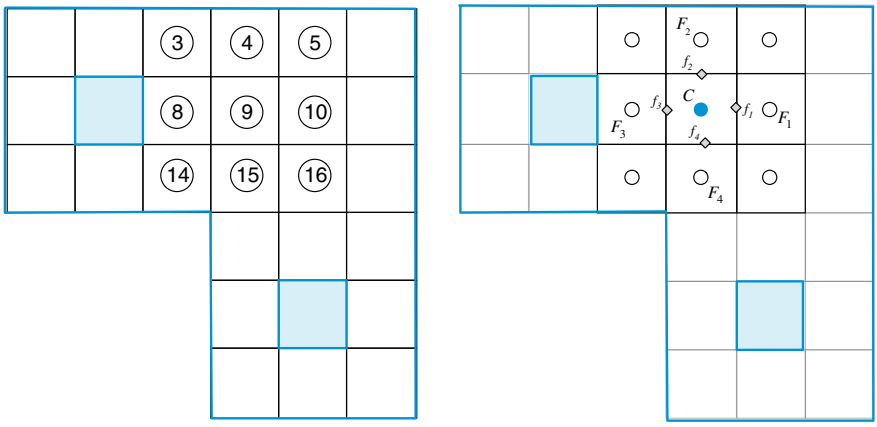


Fig. 4.11 Discretization stencil

Then, using the divergence theorem, the volume integral is transformed into a surface integral yielding

$$-\int_{S_C} (k\nabla T)\cdot d\mathbf{S} = \dot{q}_C V_C \tag{4.4}$$

This equation is actually a heat balance over element C. It is basically the integral form of the original partial differential equation and involves no approximation. Replacing the surface integral by a summation over the control volume faces, Eq. (4.4) becomes

$$-\sum_{f\sim nb(C)} (k\nabla T)_f \cdot \mathbf{S}_f = \dot{q}_C V_C \tag{4.5}$$

where f represents the integration point at the centroid of the bounding face. This transformation is the first approximation introduced. Therefore the integral in Eq. (4.4) is numerically approximated by the fluxes at the centroids of the faces. This is a second order approximation as will be demonstrated in a later chapter.

Expanding the summation, Eq. (4.5) can be written as

$$-(k\nabla T)_{f_1}\cdot\mathbf{S}_{f_1} - (k\nabla T)_{f_2}\cdot\mathbf{S}_{f_2} - (k\nabla T)_{f_3}\cdot\mathbf{S}_{f_3} - (k\nabla T)_{f_4}\cdot\mathbf{S}_{f_4} = \dot{q}_C V_C \tag{4.6}$$

Considering face f_1 shown in Fig. (4.12), the surface vector and temperature gradient in Eq. (4.6) are given by

$$\begin{aligned} \mathbf{S}_{f_1} &= \Delta y_{f_1}\mathbf{i} \\ \delta x_{f_1} &= x_{F_1} - x_C \\ \nabla T_{f_1} &= \left(\frac{\partial T}{\partial x}\right)_{f_1}\mathbf{i} + \left(\frac{\partial T}{\partial y}\right)_{f_1}\mathbf{j} \end{aligned} \tag{4.7}$$

where

x_C is the x-coordinate of the centroid of element C.
x_{F_1} is the x-coordinate of the centroid of element F_1.
Δy_{f_1} is the area of face f_1.
$\mathbf{S}_{f_1}$ is the surface vector of face f_1 directed out of element C.
∇T_{f_1} is the gradient of T at the centroid of face f_1.

and by substitution, the first term in Eq. (4.6) is converted to

$$\begin{aligned} \nabla T_{f_1}\cdot\mathbf{S}_{f_1} &= \left(\frac{\partial T}{\partial x}\mathbf{i} + \frac{\partial T}{\partial y}\mathbf{j}\right)_{f_1}\cdot\Delta y_{f_1}\mathbf{i} \\ &= \left(\frac{\partial T}{\partial x}\right)_{f_1}\Delta y_{f_1} \end{aligned} \tag{4.8}$$

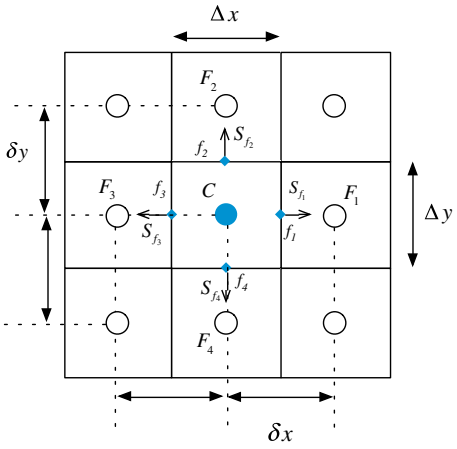


Fig. 4.12 Finite volume notation

To proceed further, a profile approximating the variation of T between C and F_1 is needed. Assuming linear variation of T, the x-component of the gradient at the face f_1 can be written as

$$\left(\frac{\partial T}{\partial x}\right)_{f_1} = \frac{T_{F_1} - T_C}{\delta x_{f_1}} \tag{4.9}$$

thus Eq. (4.8) can be approximated as

$$\nabla T_{f_1} \cdot \mathbf{S}_{f_1} = \frac{T_{F_1} - T_C}{\delta x_{f_1}} \Delta y_{f_1} \tag{4.10}$$

or more generally as

$$-(k\nabla T)_{f_1} \cdot \mathbf{S}_{f_1} = a_{F_1}(T_{F_1} - T_C) \tag{4.11}$$

where

$$a_{F_1} = -k\frac{\Delta y_{f_1}}{\delta x_{f_1}} \tag{4.12}$$

Repeating for each of the remaining faces, the following coefficients are obtained:

$$
\begin{aligned}
a_{F_2} &= -k\frac{\Delta x_{f_2}}{\delta y_{f_2}} \\
a_{F_3} &= -k\frac{\Delta y_{f_3}}{\delta x_{f_3}} \\
a_{F_4} &= -k\frac{\Delta x_{f_4}}{\delta y_{f_4}}
\end{aligned}
\tag{4.13}
$$

which when substituted into Eq. (4.6) yields

$$
\begin{aligned}
-\sum_{f\sim nb(C)}(k\nabla T)_f\cdot\mathbf{S}_f &= \sum_{F\sim NB(C)} a_F(T_F - T_C) \\
&= -(a_{F_1}+a_{F_2}+a_{F_3}+a_{F_4})T_C + a_{F_1}T_{F_1} + a_{F_2}T_{F_2} + a_{F_3}T_{F_3} + a_{F_4}T_{F_4} \\
&= \dot{q}_C V_C
\end{aligned}
\tag{4.14}
$$

or more compactly

$$
a_C T_C + \sum_{F\sim NB(C)} a_F T_F = b_C \tag{4.15}
$$

where

$$
\begin{aligned}
a_C &= -\sum_{F\sim NB(C)} a_F = -(a_{F_1}+a_{F_2}+a_{F_3}+a_{F_4}) \\
b_C &= \dot{q}_C V_C
\end{aligned}
\tag{4.16}
$$

Equations similar to Eq. (4.15) may be derived for all cells in the domain, yielding a set of algebraic equations, which can be solved using a variety of direct or iterative methods. Focusing on element C in Fig. (4.13), Eq. (4.15) implies a relation

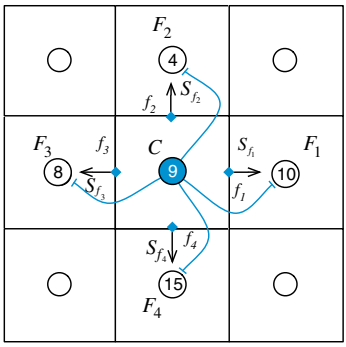


Fig. 4.13 Element notation

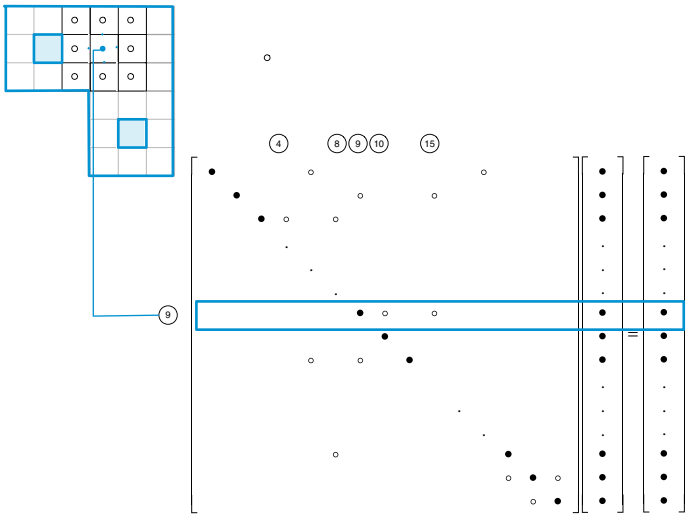


Fig. 4.14 System of equations

between T_C and the temperatures at its four neighbors namely T_{F_1}, T_{F_2}, T_{F_3}, and T_{F_4}, which in global assembly would be T_9, and T_{10}, T_4, T_8 and T_{15}.

Similar equations are also derived for boundary elements and their collection yields the set of equations illustrated in Fig. (4.14), which can be represented in matrix form as given in Eq. (4.2), where **A** is the matrix of coefficients, [*T*] the solution vector, and **b** is a vector composed of terms that cannot be included in **A**. The methodology to solve Eq. (4.2) is presented in the next section.

Finally it should be stated that the properties of the finite volume method as related to accuracy, robustness, and other characteristics will be reviewed in later chapters. This includes examining in more details the finite volume discretization of the diffusion term, which was presented above for a rectangular Cartesian grid.

4.1.5 Step IV: Solution of the Discretized Equations

The discretization of the differential equation results in a set of discrete algebraic equations, which must be solved to obtain the discrete values of *T*. The coefficients of these equations may be independent of *T* (i.e., linear) or dependent on *T* (i.e. non-linear). The techniques to solve this algebraic system of equations are independent of the discretization method, and represent the various trajectories that can be followed to obtain a solution. For the linear algebraic sets encountered in this book, the uniqueness of the solution is guaranteed. Therefore if the adopted solution method gives a solution, it will be the desired solution. All solution methods (i.e., all paths to solution) which arrive at a solution will give the same solution for the same set of discrete equations.

The solution methods for solving systems of algebraic equations may be broadly classified as direct or iterative and are briefly reviewed below.

4.1.5.1 Direct Methods

In a direct method the solution to the system of equations [e.g., Eq. (4.2)] is obtained by applying a relatively complex algorithm, in comparison with an iterative method, only once to obtain the solution for a given set of coefficients. An example of a direct method is matrix inversion whereby the solution is obtained as

$$[T] = \mathbf{A}^{-1}\mathbf{b} \tag{4.17}$$

Therefore a solution for $[T]$ is guaranteed if $\mathbf{A}^{-1}$ can be found. However, the operation count for the inversion of an $N \times N$ matrix is $O(N^3)$, which is computationally expensive. Consequently, inversion is almost never employed in practical problems. More efficient methods for linear systems are available. For the discretization methods of interest here, $\mathbf{A}$ is sparse, and for structured meshes it is banded. For certain types of equations (e.g., pure diffusion), the matrix is symmetric. Matrix manipulation can take into account the special structure of $\mathbf{A}$ in devising efficient solution techniques. Such methods will be reviewed in Chap. 10.

In general, direct methods are rarely used in computational fluid dynamics because of their large computational and storage requirements. Most industrial CFD problems today involve hundreds of thousands of cells, with 5–10 unknowns per cell even for simple problems. Thus the matrix $\mathbf{A}$ is usually very large, and most direct methods become impractical for these large problems. Furthermore, the matrix $\mathbf{A}$ is usually non-linear, so that the direct method must be embedded within an iterative loop to update nonlinearities in $\mathbf{A}$. Thus, the direct method is applied over and over again, making it all the more time-consuming.

4.1.5.2 Iterative Methods

Iterative methods follow a guess-and-correct procedure to gradually refine the estimated solution by repeatedly solving the discrete system of equations. Let us consider an extremely simple Gauss-Seidel iterative method. The overall solution loop for this method may be written as follows:

(a) Guess the discrete values of T at all grid elements in the domain.
(b) Visit each grid element in turn. Update T using

$$T_C = \frac{-\sum_{F \sim NB(C)} a_F T_F + b_C}{a_C} \tag{4.18}$$

The neighboring values are required for the update of T_C. These are assumed known at prevailing values. Thus, grid elements which have already been visited will have updated values of T and those that have not will have old values.

(c) Sweep the domain until all grid elements are covered. This completes one iteration.

(d) Check if an appropriate convergence criterion is met. The requirement, for example, could be that the maximum change in the grid-point values of T be less than 1 %. If the criterion is met, stop. Else, go back to step b and repeat.

The iteration procedure described here is not guaranteed to converge to a solution for arbitrary combinations of a_C and a_{NB}. Convergence of the process is guaranteed for linear problems if the *Scarborough criterion* is satisfied. The Scarborough criterion requires that a_C and a_{NB} should satisfy

$$\frac{-\sum_{F\sim NB(C)} a_F}{a_C} \begin{cases} \leq 1 & \text{for all grid points} \\ < 1 & \text{for at least one point} \end{cases} \tag{4.19}$$

Matrices which satisfy the Scarborough criterion have *diagonal dominance*.

The Gauss-Seidel scheme can be implemented with very little storage. All that is required is storage for the discrete values of T at the grid elements. The coefficients can be computed on the fly if desired, since the entire coefficient matrix for the domain is not required when updating the value of T at any grid point. Also, the iterative nature of the scheme makes it particularly suitable for non-linear problems. If the coefficients depend on T, they may be updated using prevailing values of T as iterations proceed. Nevertheless, the Gauss-Seidel scheme is rarely used in practice for solving the systems encountered in CFD. The rate of convergence of the scheme decreases to unacceptably low levels if the system of equations is large. In Chap. 10, an algebraic *multigrid method* will be used to accelerate the rate of convergence of iterative schemes and improve their performance.

4.1.6 Other Types of Fields

In addition to the element field introduced above, other fields are defined for different purposes. Two such fields include the face field and the vertex field that are briefly described below.

The **face field** consists of the array of values defined at the centre of the faces. As shown in Fig. 4.15, it defines a number of arrays for the interior faces and the various patch faces. The face field is used, for example, to define the face mass fluxes for use when solving advective and flow problems.

The **vertex field** schematically depicted in Fig. 4.16 stores variables at the vertices; these again are grouped into interior vertices and patch vertices. Vertex fields are usually used for post processing, and in some cases for gradient computation.

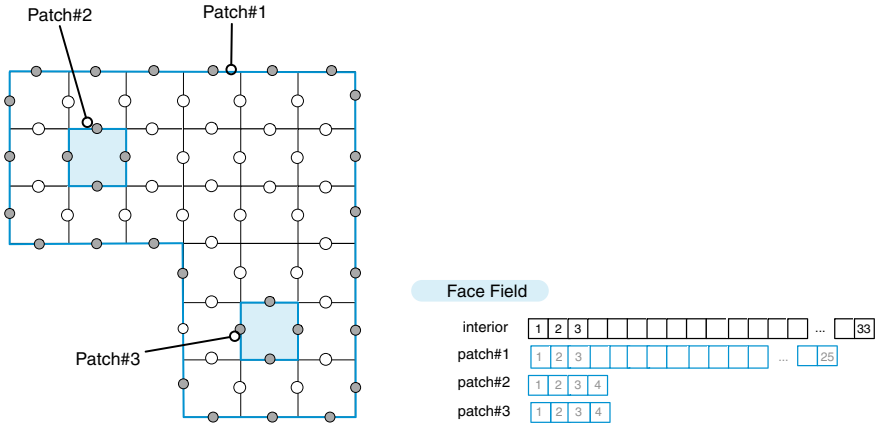


Fig. 4.15 Face field

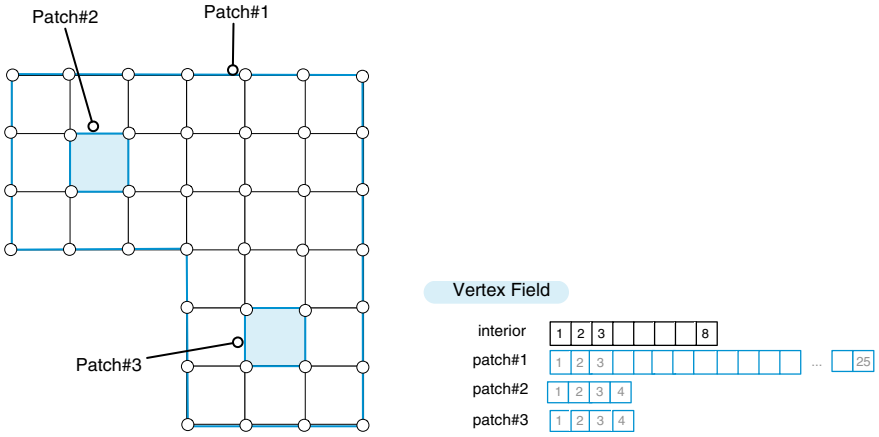


Fig. 4.16 Vertex field

4.2 Closure

This chapter overviewed the discretization process, underlining on the way the basic ingredients needed for the development of a CFD code. The coming chapters will take each of these ingredients and dissect it, while developing the "uFVM" and learning about the industrial open source CFD library, denoted by OpenFOAM®. The next two chapters will further expand on the idea of finite volume mesh and finite volume discretization.