Chapter 6
The Finite Volume Mesh

Abstract A key ingredient in the implementation of the finite volume method is setting up the geometrical support framework for the problem at hand. This process starts with mesh generation, which replaces the continuous domain by a discrete one formed of a contiguous set of non overlapping elements or cells delimited by a set of faces, and the defining of the physical boundaries through the marking of the boundary faces. It continues with the computation of relevant geometric information for the various components of the computational mesh, and is completed by capturing the topology of these components, i.e., how they are related and located one with respect to the other. Thus the result of the domain discretization step is not only the set of non-overlapping elements and other related geometric entities and the generated information about their geometric properties, but also the topological information about their arrangement and relations. It is this combined information that defines the finite volume mesh. The objective of this chapter is to clarify the topological and geometric requirements of the finite volume mesh.

6.1 Domain Discretization

The discretization of the physical domain, or mesh generation, produces a computational mesh (Fig. 6.1) on which the governing equations are subsequently solved. The methods and techniques used for domain discretization have changed drastically over the last few decades [1, 2], and, nowadays, have become mostly automated [3–6]. Before reviewing the types of elements commonly used in a computational mesh, the characteristics and attributes that the mesh system should possess in order to be employed in the context of the finite volume method are first described. These attributes will be presented in the context of computing the gradient of a variable ϕ on both a structured and an unstructured triangular mesh.

In general a geometric domain may be discretized using either a structured or an unstructured grid system. In a structured mesh, three dimensional elements are defined by their local indices (i, j, k). A structured grid system has many coding and

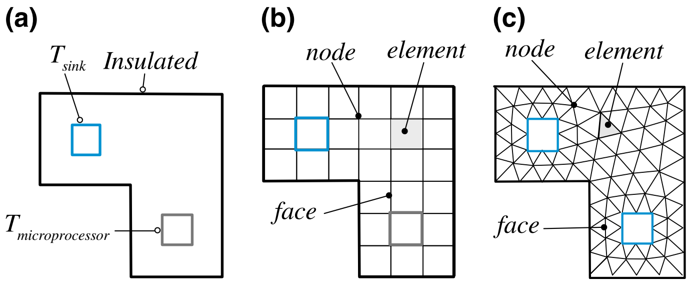

Fig. 6.1 **a** Domain of interest, **b** domain discretized using a uniform grid system, and **c** domain discretized using an unstructured grid system with triangular elements

performance advantages but suffers from a limited geometric flexibility. Additional flexibility in the generation of structured meshes can be achieved by employing multiple blocks to define the geometry, with a structured mesh generated for each block either independently from other blocks or jointly.

Another way to make the mesh generation more flexible is to avoid the use of structured grids with their implicit topological information, and to adopt an unstructured mesh with explicit topological information based on connectivity tables and geometric entity numbering.

Structured grids remained the staple of numerical simulation for a long time and it is only in the past two decades that unstructured grids became more popular [7]. Starting in early 1970s, interest in automatic mesh generation escalated as problem size increased and manual mesh generation became too time consuming [8]. The first methods were semi-automatic with an operator manually placing points in the computational domain and then, in a second step, using a computer to generate the mesh. Nowadays the whole process is fully automated with both points and elements generated automatically.

Most modern CFD codes have the ability to use unstructured grids in addition to a variety of hybrid multiblock grids [9]. OpenFOAM® [10] uses unstructured grids but can also use conforming and non-conforming multiblock grids [11]. The finite volume method will be presented here in the context of an unstructured grid system. However as the unstructured finite volume mesh requirements are defined, its characteristics will be compared to those of a structured grid system.

6.2 The Finite Volume Mesh

The discussion for the requirements of the finite volume mesh will be contextualized in terms of a simple problem, namely the computation of the gradient of an element field. The gradient will first be computed on a structured grid and then over an unstructured grid; the differences will help clarifying a number of issues.

6.2.1 Mesh Support for Gradient Computation

Many techniques can be used to compute the gradient of an element field, and these will form the subject of Chap. 9. The method adopted in this chapter is based on the Green-Gauss theorem [12, 13]. It is relatively straightforward and can be used for a variety of topologies and grids (structured/unstructured, orthogonal/nonorthogonal, etc.). The starting point is defining the average gradient over a finite volume element of centroid C and volume V_C as

$$\overline{\nabla\phi}_C = \frac{1}{V_C}\int\limits_{V_C} \nabla\phi\, dV \tag{6.1}$$

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

$$\overline{\nabla\phi}_C = \frac{1}{V_C}\int\limits_{\partial V_C} \phi\, d\mathbf{S} \tag{6.2}$$

where $d\mathbf{S}$ is the **outward** pointing surface vector. In the presence of discrete faces, Eq. (6.2) can be written as

$$\overline{\nabla\phi}_C V_C = \sum_{\partial V_C}\int\limits_{face} \phi\, d\mathbf{S} \tag{6.3}$$

Next the integral over a cell face is approximated using the mid-point integration rule to become equal to the interpolated value of the field at the face centroid multiplied by the face area, resulting in

$$\overline{\nabla\phi}_C = \frac{1}{V_C}\sum_{f=nb(C)} \overline{\phi_f}\mathrm{S}_f \tag{6.4}$$

By reviewing Eq. 6.4 and Fig. 6.2, it is clear that to compute the average of the gradient over the control element C, information about the face area and direction ($\mathbf{S}_f$) is required, as well as information about the neighboring elements and the ϕ values at the element centroids $\left(\phi_C, \phi_{F_k}\right)$. This information is needed to compute the value of ϕ at the interface (ϕ_f), which will have to be interpolated in some fashion.

A profile for the variation of the dependent variable ϕ between nodal values is assumed, which basically introduces an approximation in the evaluation of the gradient. In all cases the value of ϕ_f will have to be computed at each face centroid. Assuming a linear profile for the variation of ϕ between the elements C and F straddling the interface f, an approximate value for ϕ_f, denoted by $\overline{\phi_f}$, can be computed as

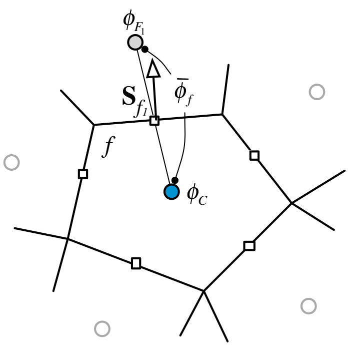

Fig. 6.2 Gradient computation

$$\overline{\phi}_f = g_F \phi_F + g_C \phi_C \tag{6.5}$$

One way to calculate the weight factors g_F and g_C is given by

$$g_F = \frac{V_C}{V_C + V_F} \qquad g_C = \frac{V_F}{V_C + V_F} = 1 - g_F \tag{6.6}$$

Other interpolation practices may be used some of which will be explained later in the chapter.

Example 1
Compute the gradient for the two fields given in Table 6.1 *over the two-dimensional cell shown in* Fig. 6.3 *using the surface vector values given in the table.*

Table 6.1 Geometric data for Example 1

	Sf	V	Field (1)	Field (2)
C		37.8	1	6
1	(−2.4, −3.24)		1	10
2	(2.4, −3.48)		1	9
3	(4.1, −6.7)		1	5
4	(2.2, 3.7)		1	3
5	(−2.64, 2.9)		1	4
6	(−3.66, 6.82)		1	8

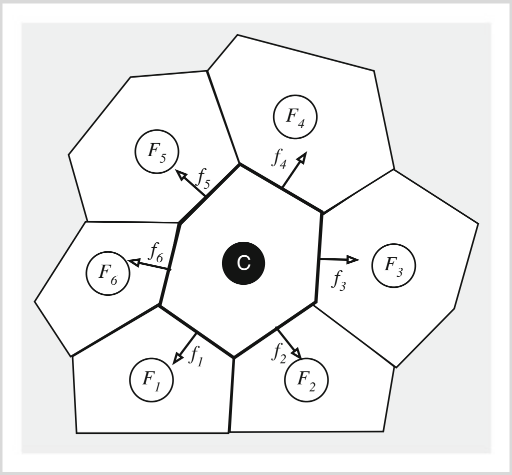

Fig. 6.3 A two dimensional cell for Example 1

Solution

For case 1, the field is constant with a value of 1, so the value of the gradient is expected to be 0.

$$
\begin{aligned}
\overline{\nabla\phi_C} &= \frac{1}{V_C}\sum_{f=nb(C)}\phi_f\mathbf{S}_f \\
&= \frac{1}{V_C}\left(\phi_{f_1}\mathbf{S}_{f_1} + \phi_{f_2}\mathbf{S}_{f_2} + \phi_{f_3}\mathbf{S}_{f_3} + \phi_{f_4}\mathbf{S}_{f_4} + \phi_{f_5}\mathbf{S}_{f_5} + \phi_{f_6}\mathbf{S}_{f_6}\right) \\
\overline{\nabla\phi_C} &= \frac{1}{37.8}\begin{pmatrix} 1(-2.4\mathbf{i} - 3.24\mathbf{j}) + 1(2.4\mathbf{i} - 3.48\mathbf{j}) + 1(4.1\mathbf{i} - 6.7\mathbf{j}) \\ + 1(2.2\mathbf{i} + 3.7\mathbf{j}) + 1(-2.64\mathbf{i} + 2.9\mathbf{j}) + 1(-3.66\mathbf{i} + 6.82\mathbf{j}) \end{pmatrix} \\
&= (0\mathbf{i} + 0\mathbf{j})
\end{aligned}
$$

This is actually a property of the surfaces of closed elements, provided they all are pointing outward (or inward) they always sum to zero.

For case 2 the gradient is computed as

$$
\begin{aligned}
\overline{\nabla\phi_C} &= \frac{1}{V_C}\sum_{f=nb(C)}\phi_f\mathbf{S}_f \\
&= \frac{1}{V_C}\left(\phi_{f_1}\mathbf{S}_{f_1} + \phi_{f_2}\mathbf{S}_{f_2} + \phi_{f_3}\mathbf{S}_{f_3} + \phi_{f_4}\mathbf{S}_{f_4} + \phi_{f_5}\mathbf{S}_{f_5} + \phi_{f_6}\mathbf{S}_{f_6}\right) \\
&= \frac{1}{37.8}\begin{pmatrix} 10(-2.4\mathbf{i} - 3.24\mathbf{j}) + 9(2.4\mathbf{i} - 3.48\mathbf{j}) + 5(4.1\mathbf{i} - 6.7\mathbf{j}) \\ + 3(2.2\mathbf{i} + 3.7\mathbf{j}) + 4(-2.64\mathbf{i} + 2.9\mathbf{j}) + 8(-3.66\mathbf{i} + 6.82\mathbf{j}) \end{pmatrix} \\
&= -15.14\mathbf{i} - 19.96\mathbf{j}
\end{aligned}
$$

6.3 Structured Grids

For a regular structured grid, every interior cell in the domain is connected to the same number of neighboring cells. These neighboring cells (Fig. 6.4) can be identified using the indices i, j, and (k) in the x, y, and (z) coordinate direction, respectively, and can be directly accessed by incrementing or decrementing the respective indices. This allows for lower memory usage since topological information is embedded in the mesh structure through the indexing system. This also leads to greater efficiency in coding, cache utilization, and vectorization. Structured grids were widely used in the development of the Finite Volume and Finite Difference methods.

In a structured grid, one can associate with each computational cell an ordered set of indices (i, j, k), where each index varies over a fixed range, independently of the values of the other indices, and where neighboring cells have associated indices that differ by plus or minus one. Thus, if there are Ni, Nj, and Nk elements in the i, j, and k index direction, respectively, then the total number of elements in the domain is $Ni * Nj * Nk$. In three-dimensional spaces, elements are hexagons with 6 faces and 8 vertices, with each interior element having 6 neighbors. In two-dimensions, elements are quadrilaterals with 4 faces and 4 vertices, with each interior element having 4 neighbors.

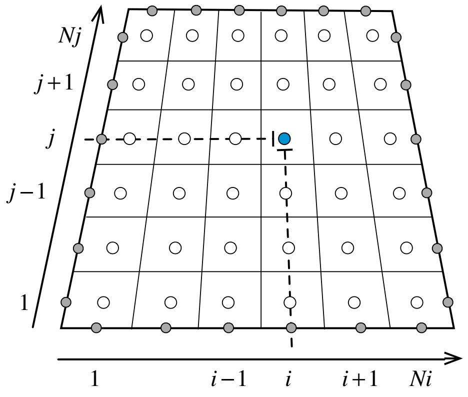

Fig. 6.4 Local indices and topology

6.3.1 Topological Information

Global indices are generally used for building the full system of equations over the computational domain, while local indices are employed to define the local stencil for an element, information that is useful during the discretization process. In structured grid systems local indices are used interchangeably as global indices

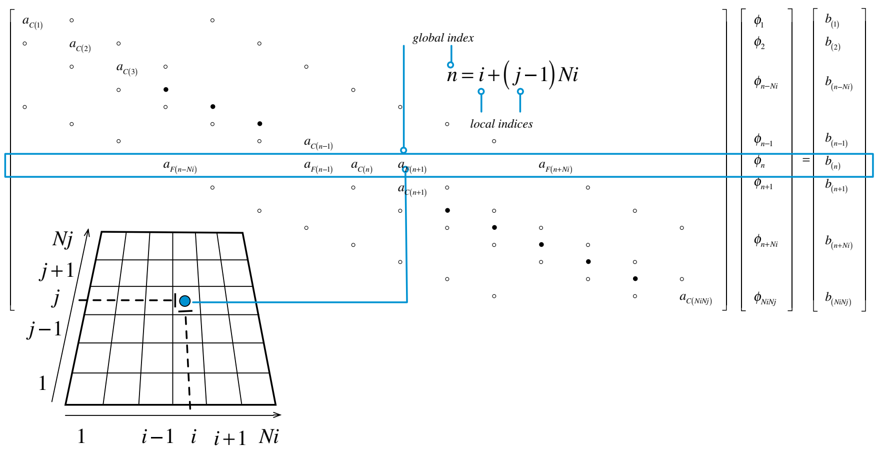

Fig. 6.5 Local versus global indices

because they can be readily translated to global indices and vice versa, as illustrated in Fig. 6.5. In two dimensional spaces the relation between local indices (i, j) and the global index (n) is given by

$$n = i + (j-1)Ni \qquad 1 \le i \le Ni \qquad 1 \le j \le Nj \tag{6.7}$$

and the corresponding global indices for the neighbors of cell (i, j) are obtained as

$$\begin{array}{lcllcl} (i,j) & \to & n & & & \\ (i+1,j) & \to & n+1 & (i-1,j) & \to & n-1 \\ (i,j+1) & \to & n+Ni & (i,j-1) & \to & n-Ni \end{array} \tag{6.8}$$

On the other hand, in three-dimensional spaces the relation is

$$n = i + (j-1)Ni + (k-1)Ni * Nj \qquad 1 \le i \le Ni \qquad 1 \le j \le Nj \qquad 1 \le k \le Nk \tag{6.9}$$

and the corresponding global indices for the neighbors of cell (i, j, k) are given by

$$\begin{array}{lcllcl} (i,j,k) & \to & n & & & \\ (i+1,j,k) & \to & n+1 & (i-1,j,k) & \to & n-1 \\ (i,j+1,k) & \to & n+Ni & (i,j-1,k) & \to & n-Ni \\ (i,j,k+1) & \to & n+Ni*Nj & (i,j,k-1) & \to & n-Ni*Nj \end{array} \tag{6.10}$$

This greatly simplifies the access to the coefficients and the solution of the system of equations, since the coefficients constructed over the local stencil of a cell are basically used in the general system of equations with no need for a translational step between local and global indices as the mapping between them is readily available. This also applies to the geometric fields and the various conservation fields that are being resolved.

Example 2
In a 5 × 7 *structured grid find the global indices of the neighbors of the element C defined by the local index* (3, 4).

Solution
For the defined mesh $Ni = 5$ and $Nj = 7$, the global index of element $C(3, 4)$ is thus $3 + (4 - 1) * 5 = 18$, the neighbors of elements C in terms of their local and global indices are $(2, 4) \rightarrow 17$, $(4, 4) \rightarrow 19$, $(3, 3) \rightarrow 13$, and $(3, 5) \rightarrow 23$.

6.3.2 Geometric Information

As shown in Fig. 6.6, accessing the local geometric information around an element is quite simple. For element (i, j) the surrounding stored faces are $\mathbf{S}1(i, j)$, $\mathbf{S}2(i, j)$, $\mathbf{S}1(i + 1, j)$, and $\mathbf{S}2(i, j + 1)$. Since for any element the faces have to be pointing outward (see Fig. 6.6), then

$$\begin{aligned} \mathbf{S}_{i-1/2,j} &= -\mathbf{S}1(i,j) \\ \mathbf{S}_{i,j-1/2} &= -\mathbf{S}2(i,j) \end{aligned} \tag{6.11}$$

while for other faces, the following applies:

$$\begin{aligned} \mathbf{S}_{i+1/2,j} &= \mathbf{S}1(i+1,j) \\ \mathbf{S}_{i,j+1/2} &= \mathbf{S}2(i,j+1) \end{aligned} \tag{6.12}$$

The element field in a structured two or three dimensional mesh is also defined as an array of size $[Nx][Ny]$ or $[Nx][Ny][Nz]$, respectively. Thus accessing the element value and its neighbors is equally simple. The element field of a multi-dimensional system may also be defined using a one dimensional array, which in two

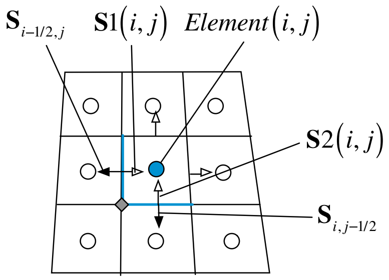

Fig. 6.6 Geometric information

dimensions will be of size $[Ni * Nj]$ and in three dimensions of size $[Ni * Nj * Nk]$. Representing a multi-dimensional field by a one-dimensional array, with values stored using the global indexing system, saves a lot of computer memory when a multi grid system is adopted.

6.3.3 Accessing the Element Field

Accessing the field in a structured grid is as simple as using the indices of the element. Therefore, in a two-dimensional space, $\phi(i,j)$ or $\phi_{i,j}$ is the value of field ϕ at element (i, j). As shown in Fig. 6.7a the values of ϕ at the neighboring cells to (i, j) are, respectively, $\phi_{i+1,j}$, $\phi_{i-1,j}$, $\phi_{i,j+1}$, and $\phi_{i,j-1}$.

As mentioned above computing the gradient using Eq. (6.4) requires calculating the value of ϕ at each face of the finite volume (for our pseudo elements the front and back faces have the same value and thus will not be included in the computation). Thus in addition to the value of ϕ at point (i, j), the values of ϕ at the neighboring points $(i + 1, j)$, $(i - 1, j)$, $(i, j + 1)$, and $(i, j - 1)$ are also needed. In a structured grid this information is readily available and the value at the face is computed by simple interpolation between the values of ϕ at the centroids of the two volumes sharing the face. Using Eq. (6.5), the interpolated value at face $(i + 1/2, j)$ in terms of local indices can be written as

$$\overline{\phi}_{i+1/2,j} = g_{i+1/2,j}\phi_{i+1,j} + \left(1 - g_{i+1/2,j}\right)\phi_{i,j} \tag{6.13}$$

Details of the linear interpolation will be presented later in the chapter. Referring to Fig. 6.7a, the gradient at element (i, j) can be computed using local indexing as

$$\begin{aligned}\overline{\nabla\phi_{i,j}} &= \frac{1}{V_{ij}}\left(\overline{\phi}_{i+1/2,j}\mathbf{S}_{i+1/2,j} + \overline{\phi}_{i-1/2,j}\mathbf{S}_{i-1/2,j} + \overline{\phi}_{i,j+1/2}\mathbf{S}_{i,j+1/2} + \overline{\phi}_{i,j-1/2}\mathbf{S}_{i,j-1/2}\right)\\ &= \frac{1}{V_{ij}}\left(\overline{\phi}_{i+1/2,j}\mathbf{S1}_{i+1,j} - \overline{\phi}_{i-1/2,j}\mathbf{S1}_{i,j} + \overline{\phi}_{i,j+1/2}\mathbf{S2}_{i,j+1} - \overline{\phi}_{i,j-1/2}\mathbf{S2}_{i,j}\right)\end{aligned} \tag{6.14}$$

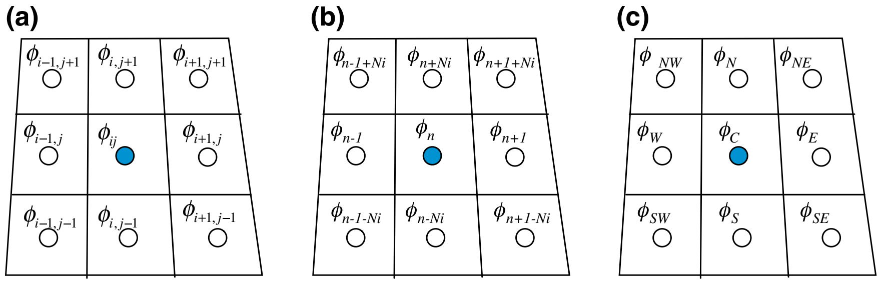

Fig. 6.7 Local versus discretization versus global indices. **a** Local indexing, **b** global indexing, and **c** discretization indexing

while using a global indexing system, Fig. 6.7b, it becomes

$$\overline{\nabla\phi_n} = \frac{1}{V_n}\left(\overline{\phi}_{n+1/2}\mathbf{S}_{n+1/2} + \overline{\phi}_{n-1/2}\mathbf{S}_{n-1/2} + \overline{\phi}_{n+Ni/2}\mathbf{S}_{n+Ni/2} + \overline{\phi}_{n-Ni/2}\mathbf{S}_{n-Ni/2}\right) \tag{6.15}$$

It should be mentioned that **S** is the outward vector normal to the surface at the control volume face. Except at the domain boundaries, control volume faces are shared by two elements. Therefore the outward direction for one element will represent the inward direction for the other element. Thus to avoid duplicating surface vectors at interfaces, only one vector is computed and stored at an interface; the one in the direction of increasing *i* or *j*. The embedded features in a structured grid system allows for the correct direction to be chosen with no need to store any additional information. For any element (*i*, *j*), the surface with an index greater than *i* or *j* is positive while the surface with an index lower than *i* or *j* is multiplied by a negative sign. This explain the negative signs in Eqs. (6.11) and (6.14).

6.3.3.1 Discretization Indexing

In addition to local and global indexing, another type known as discretization indexing is sometimes used, where the fields and geometric quantities are defined in terms of their position or neighboring values. Referring to Fig. 6.7c, the gradient at element (*i*, *j*) can be computed using discretization indexing as

$$\overline{\nabla\phi_C} = \frac{1}{V_C}\left(\overline{\phi}_e\mathbf{S}_e + \overline{\phi}_w\mathbf{S}_w + \overline{\phi}_n\mathbf{S}_n + \overline{\phi}_s\mathbf{S}_s\right) \tag{6.16}$$

Thus the algorithm for computing the gradient could be written as

```
>Loop over elements (i,j)
     > initialize element gradient to zero grad(i,j) = 0
     > loop over the element faces
          > compute the flux_f =  phi_f*S_f
          > add/subtract flux_f to the element gradient
          depending on the orientation of S_f (pointing
          out of/into element)
     > divide the sum of the fluxes stored in the
     gradient by the volume of the element to yield the
     element gradient
```

6.4 Unstructured Grids

Unstructured grids offer more flexibility in meshing a domain both in terms of the element types that can be used and in terms of where the elements can be concentrated. This flexibility, however, comes at the cost of additional complexity.

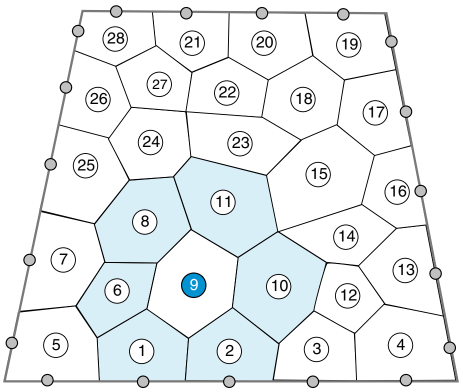

Fig. 6.8 Unstructured mesh global indexing

In an unstructured mesh system, elements are numbered sequentially, as are faces, nodes, and other geometric quantities. This means that there is no direct way to link various entities together based on their indices alone. Thus local connectivity has to be defined explicitly starting with determining the geometric quantities for a particular element. In Fig. 6.8, for example, the neighbors of element 9 cannot be directly derived from its index. Similarly the bounding faces of element 9, or for that matter their nodes, cannot be guessed or derived from its index in the same way this can be done in a structured grid.

Therefore detailed topological information about neighboring elements, faces, and nodes is needed to complement the global indexing of the grid.

6.4.1 Topological Information (Connectivities)

As shown in Fig. 6.9, topological information is developed by explicitly constructing the local, Fig. 6.9a, and global, Fig. 6.9b, indices that define the geometric component connectivities (element to element, element to faces, faces to elements, element to nodes, etc.). To this end, the data structure of elements, faces, and nodes now include information about neighboring connections in terms of local and global indices.

The gradient computation algorithm can now be expressed in terms of the discretization indices shown in Fig. 6.9a as

$$\overline{\nabla\phi_C} = \frac{1}{V_C}\left(\overline{\phi}_{f_1}\mathbf{S}_{f_1} + \overline{\phi}_{f_2}\mathbf{S}_{f_2} + \overline{\phi}_{f_3}\mathbf{S}_{f_3} + \overline{\phi}_{f_4}\mathbf{S}_{f_4} + \overline{\phi}_{f_5}\mathbf{S}_{f_5} + \overline{\phi}_{f_6}\mathbf{S}_{f_6}\right) \tag{6.17}$$

or in terms of the local numbers of the element faces shown in Fig. 6.9a as

$$\overline{\nabla\phi_{(0)}} = \frac{1}{V_{(0)}}\left(\overline{\phi}_{(1)}\mathbf{S}_{(1)} + \overline{\phi}_{(2)}\mathbf{S}_{(2)} + \overline{\phi}_{(3)}\mathbf{S}_{(3)} + \overline{\phi}_{(4)}\mathbf{S}_{(4)} + \overline{\phi}_{(5)}\mathbf{S}_{(5)} + \overline{\phi}_{(6)}\mathbf{S}_{(6)}\right) \tag{6.18}$$

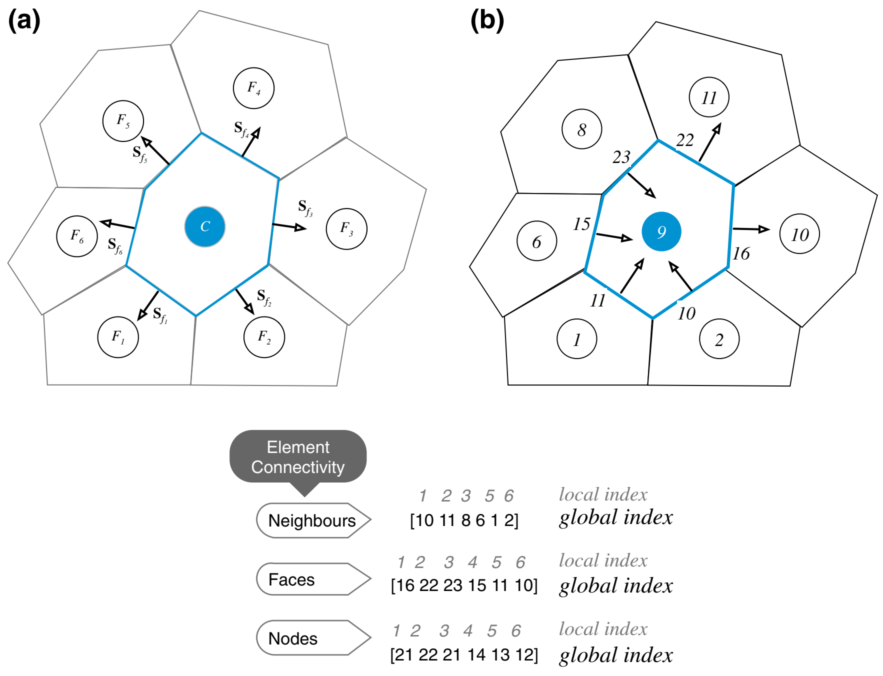

Fig. 6.9 Element connectivity and face orientation using **a** local indices and **b** global indices

where the "()" indicates the local index of the face. The gradient relation can be also written in terms of global indices representing the stored values at the element faces shown in Fig. 6.9b as

$$\overline{\nabla\phi_9} = \frac{1}{V_9}\left(\overline{\phi}_{16}\mathbf{S}_{16} + \overline{\phi}_{22}\mathbf{S}_{22} - \overline{\phi}_{23}\mathbf{S}_{23} - \overline{\phi}_{15}\mathbf{S}_{15} - \overline{\phi}_{11}\mathbf{S}_{11} - \overline{\phi}_{10}\mathbf{S}_{10}\right) \tag{6.19}$$

where the negative signs for terms of faces 23, 15, 11 and 10 are to be noted. While the local surface vector **S** is always assumed to be in the outward direction, this is not necessarily true, as only one normal vector is stored at any one face. A look at Fig. 6.9 shows that these specific stored surface vectors are actually pointing inward of element 9, thus the negative sign. Unlike structured grid systems where the correct direction can easily be obtained, in an unstructured grid the direction of the normal to the surface should somehow be stored. This will be detailed in the following section. In order to account for the vector direction, a sign function is used and the equation for the gradient is modified as

$$\overline{\nabla\phi_k} = \frac{1}{V_k}\left(-\sum_{n\leftarrow\langle f\sim nb(k)\rangle<k}\overline{\phi}_n\mathbf{S}_n + \sum_{n\leftarrow\langle f\sim nb(k)\rangle>k}\overline{\phi}_n\mathbf{S}_n\right) \tag{6.20}$$

For faces, information about the straddling elements is what defines the topology of the face. Furthermore the orientation of the face can be defined in a standard fashion by indexing the elements in a specified order. In this case the normal vector at the interface between two elements is oriented from element 1 to element 2, which are also denoted in OpenFOAM® by the *owner* and *neighbor* elements, respectively, as shown in Fig. 6.10.

Therefore if the interface is considered with element 2, then it should be multiplied by a negative sign. Thus considering the faces that bound element 9, the connectivity information is defined as shown in Fig. 6.11.

The computation of the gradient can be done for every element. However as the flux $\overline{\phi}_f \mathbf{S}_f$ at every interface is the same for the straddling elements with the

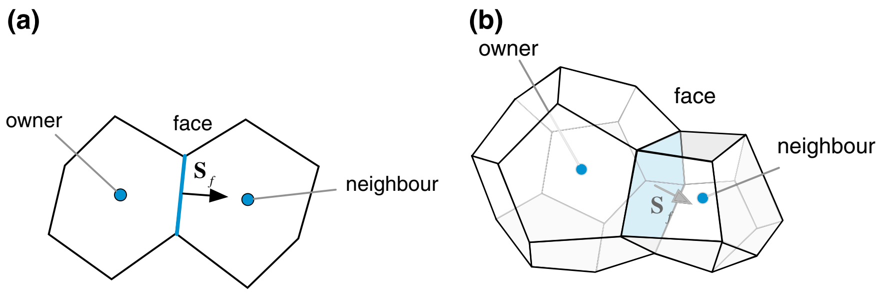

Fig. 6.10 Owners, neighbors, and faces for **a** 2D and **b** 3D elements

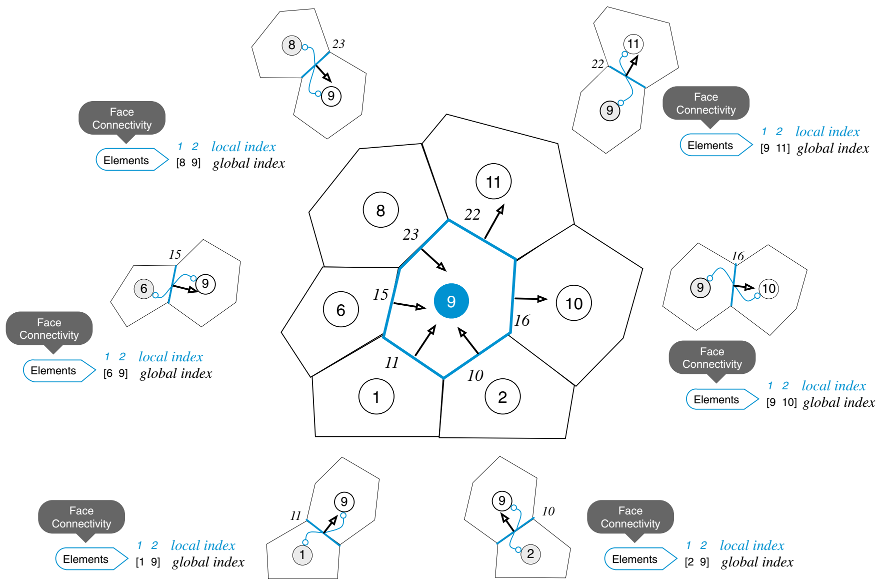

Fig. 6.11 An example of face, element, and node connectivities for unstructured grids

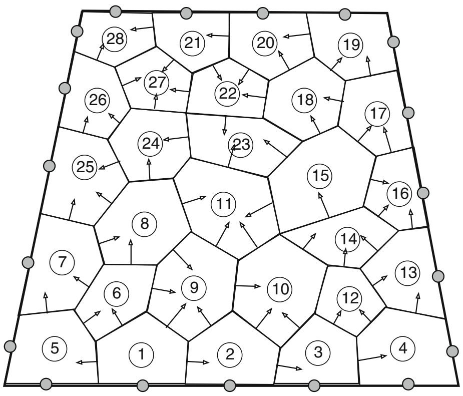

Fig. 6.12 An unstructured mesh system

difference being its sign, the computation of the gradient could proceed in a more efficient manner by computing the gradient over the entire domain (e.g., the entire domain shown in Fig. 6.12) rather than element by element. This is done by looping over all the faces in the computational domain and directly updating the value of the gradient for the elements straddling the interface by incrementing and decrementing the calculated flux from the gradient of element 1 and element 2, respectively.

Therefore the algorithm on an unstructured grid for calculating a gradient field becomes

Algorithm for computing the gradient on an unstructured grid system

```
1. Declare gradient array and initialize it to zero
2. Loop over interior faces
     > compute flux_f = phi_f*Sf
     > add flux_f to gradient of owner element and -
     flux_f to gradient of neighbor element
3.       Loop over boundary faces
     > compute flux_f = phi_f Sf
     > add flux_f to gradient of owner element
4.       Loop over elements
     > divide gradient by volume of element
```

This basically yields the gradient at each element in the computational domain as per Eq. (6.20). The same algorithm could be used with a structured grid to reduce the computational cost.

Example 3
Write the connectivity arrays for elements 1 *and* 5, *and faces* 1, 7, 11 *and* 23 *for the mesh shown in* Figs. 6.13 *and* 6.14.

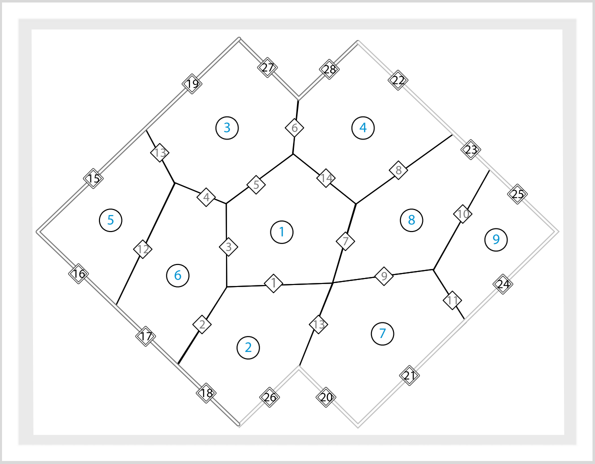

Fig. 6.13 An unstructured mesh for Example 2

Solution

For elements, the neighbors are stored in the increasing index number of shared faces, and interior faces are stored in increasing index number, followed by boundary faces again stored in increasing index number.

element 5
neighbors 6 3
faces 12 13 15 16

element 1
neighbors 2 6 3 8 4
faces 1 3 5 7 14

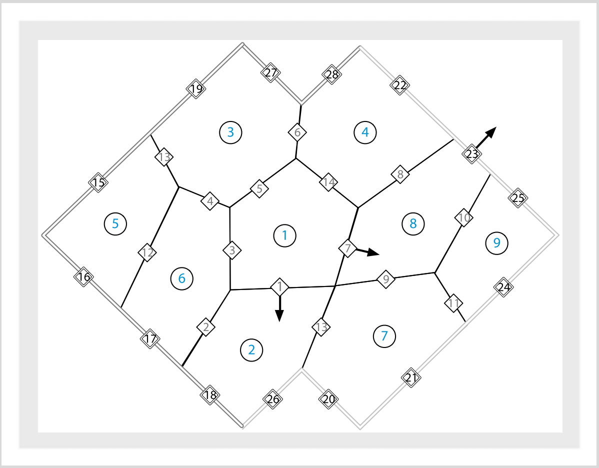

Fig. 6.14 Surface vector direction at faces *1*, *7*, and *23* for Example 2

For a face the owner is the element of lower index and the neighbor is the element of higher index. A Boundary face has only an owner. The direction of the surface vector associated with any face is oriented from owner to neighbor.

face 1
owner 1
neighbor 2

face 7
owner 1
neighbor 8

face 11
owner 7
neighbor 9

face 23 (a boundary face)
owner 8
neighbor -1 (a boundary face has no neighbor)

6.5 Geometric Quantities

In addition to topological data, the finite volume mesh incorporates information about its geometric entities, such as the volume of elements, the area of faces, the centroids of elements and faces, the alignment of faces with the vectors joining the centroids of the owner and neighbor elements (Fig. 6.15), etc. The calculations of some of these geometric quantities will be presented next. The type of elements that

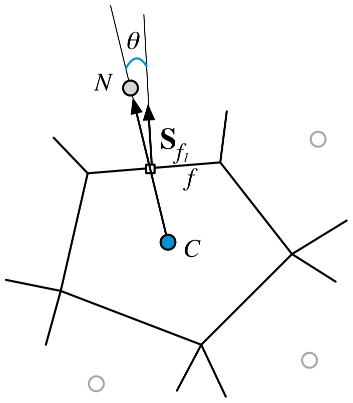

Fig. 6.15 Angle between surface vector and vector joining the centroids of the owner and neighbor elements

can be used in generating the mesh are first described, followed by the techniques employed for computing the geometric information.

6.5.1 Element Types

An element in the finite volume mesh is basically a polyhedron in a three-dimensional mesh (Fig. 6.16) or a polygon in a two-dimensional mesh (Fig. 6.17). The most widely used three-dimensional shapes, as displayed in Fig. 6.16, are tetrahedrons, hexahedrons, prisms, and in some cases general polyhedrons.

The type of faces for these three-dimensional elements, which also represent the type of two-dimensional elements (Fig. 6.17), vary greatly, with the ones that are the most widely used being quadrilaterals, triangles and pentagons, though general polygons have also been adopted in some applications.

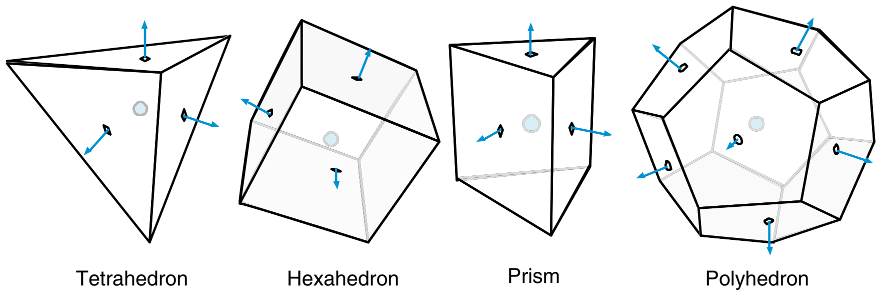

Fig. 6.16 Three-dimensional element types

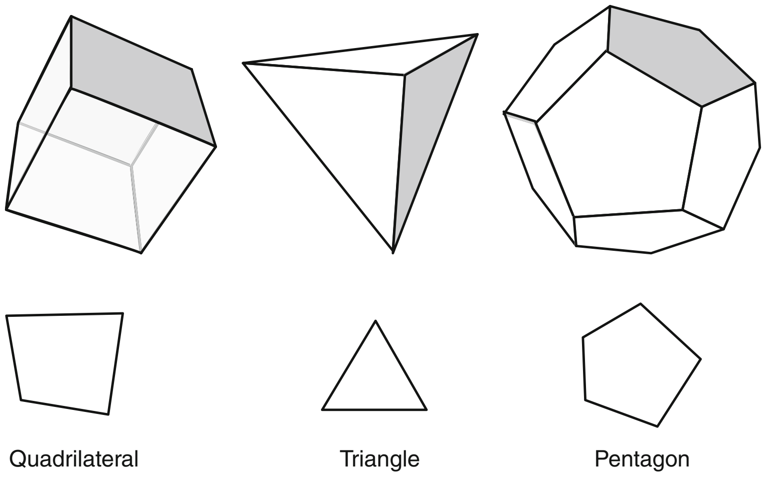

Fig. 6.17 Three-dimensional face types or two-dimensional element types

The computation of geometric factors for such elements and faces will now be detailed. It is worth noting that when working with a two-dimensional mesh the volume of the elements are considered to be the area of the two dimensional elements multiplied by a unit dimension in the off plane direction. Thus the techniques to compute the volume of elements of two-dimensional meshes are exactly those used in computing the surface area of faces of three-dimensional meshes. Other variables arising during discretization, which are solely dependent on geometric quantities, will be presented when needed.

6.5.2 Computing Surface Area and Centroid of Faces

The general shape of an element face in a three-dimensional finite volume mesh is a polygon, though triangular and quadrilateral faces are the most widely used. The computation of the surface vector and centroid follows the same procedure for all types of polygons. Basically a point is constructed within the polygon based on the average of all the points that define the polygon. This point is the geometric centre of the polygon $\mathbf{x}_G = (x_G, y_G, z_G)$, which coincides with the centroid of the polygon $\mathbf{x}_{CE} = (x_{CE}, y_{CE}, z_{CE})$ only for some very special shapes, which include triangles. Therefore the geometric centre of k points forming a polygon is computed as

$$\mathbf{x}_G = \frac{1}{k}\sum_{i=1}^{k}\mathbf{x}_i \qquad (6.21)$$

Using the geometric centre (Fig. 6.18) a number of triangles are formed with the centre as the apex for each side of the polygon. For each of the triangles the centroid (since for triangles $\mathbf{x}_G$ and $\mathbf{x}_{CE}$ coincide) and area are readily computed. The sum of their areas will give the total area of the polygon. For computing the centroid the area-weighted centroid (or geometric centre) of each sub-triangle are summed over the polygon and divided by the polygon area, yielding the centroid of the polygon. Mathematically this is computed as

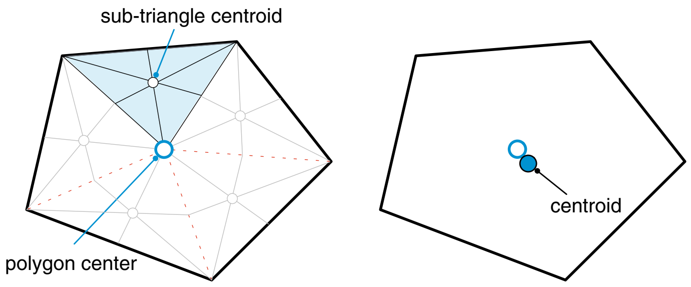

Fig. 6.18 The geometric centre and centroid of a polygon

$$
\begin{aligned}
S_f &= \sum_{t\sim Sub-triangles(C)} S_t \\
(\mathbf{x}_{CE})_f &= \frac{\sum\limits_{t\sim Sub-triangles(C)} (\mathbf{x}_{CE})_t * S_t}{S_f}
\end{aligned} \tag{6.22}
$$

6.5.2.1 Surface of a Triangle

The area of a triangle is computed using vector product. The magnitude of the vector product of two vectors represents the area of the parallelogram formed by the two vectors. Thus the area of the triangle is half the magnitude of the vector product of the two vectors. Denoting the position vectors of the three vertices 1, 2, and 3 of the triangle shown in Fig. 6.19 by $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$, respectively, the surface vector of the triangle can be computed as

$$
\mathbf{S} = \frac{1}{2}(\mathbf{r}_2 - \mathbf{r}_1) \times (\mathbf{r}_3 - \mathbf{r}_1) = \frac{1}{2}\begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = S_x\mathbf{i} + S_y\mathbf{j} + S_z\mathbf{k} \tag{6.23}
$$

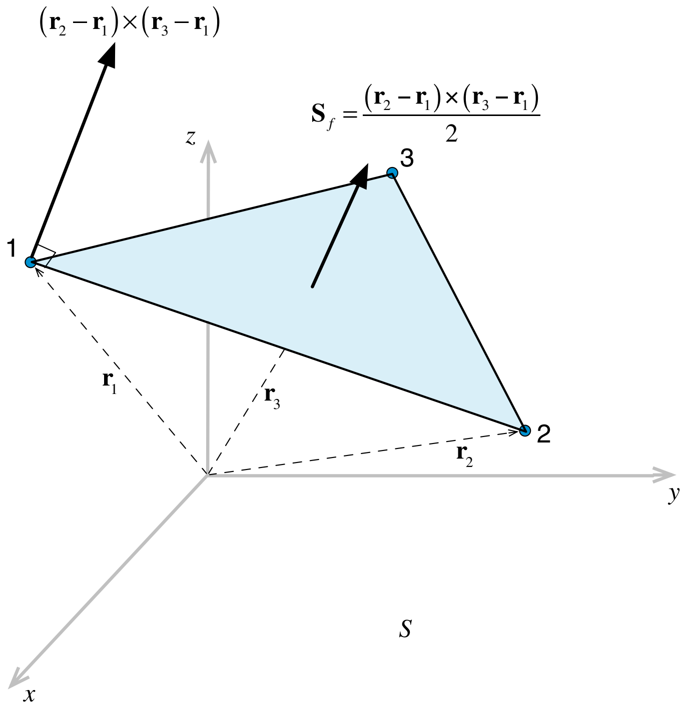
$(\mathbf{r}_2-\mathbf{r}_1)\times(\mathbf{r}_3-\mathbf{r}_1)$
$\mathbf{S}_f = \frac{(\mathbf{r}_2-\mathbf{r}_1)\times(\mathbf{r}_3-\mathbf{r}_1)}{2}$
z
3
1
$\mathbf{r}_1$
$\mathbf{r}_3$
2
$\mathbf{r}_2$
y
S
x

Fig. 6.19 Surface vector and area magnitude of a triangle in a three dimensional space

and the magnitude of the area is given by

$$S = \sqrt{S_x^2 + S_y^2 + S_z^2} \tag{6.24}$$

To know whether the surface vector is pointing outward, its dot product with the position vector that joins the centroid of the element *CE* to the centroid of the surface *ce* is computed. If the sign of the dot product is positive then the surface vector is pointing outward, otherwise it is pointing inward. The same approach may be used to discern the orientation of the surface vector in two dimensions.

For a two dimensional grid the surface area represents the volume of the control cell with a unit depth in the off-plane direction. Therefore the volume of a triangular cell in a two dimensional grid is calculated using

$$\begin{aligned} V &= \frac{1}{2}|(\mathbf{r}_2 - \mathbf{r}_1) \times (\mathbf{r}_3 - \mathbf{r}_1)| = \frac{1}{2}[(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)] \\ &= \frac{1}{2}[x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)] \end{aligned} \tag{6.25}$$

Note that the signed volume (or area) will be positive if the vertices 1, 2 and 3 are oriented counterclockwise around the triangle, otherwise it will be negative. Taking the absolute value of the right hand side of Eq. (6.25) will always result in the correct volume value.

Example 4

Compute the centroid and area of the polygon shown in Fig. 6.20 *whose coordinates are displayed in* Table 6.2.

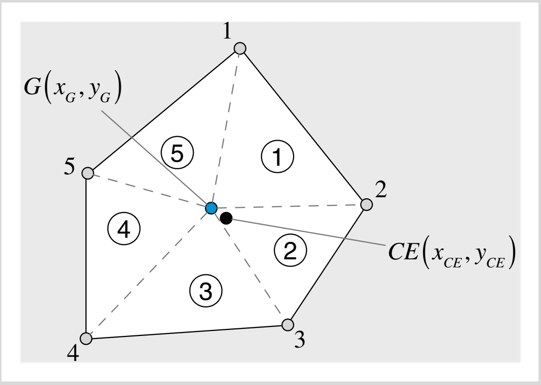

Fig. 6.20 A polygonal element

Table 6.2 Coordinates of the polygonal element for Example 3

Node	1	2	3	4	5
x	1	2.4	2	0.4	0
y	6.4	4.0	0.2	0	4.0

Solution

For this polygon $k = 5$. Thus its geometric centre $G(x_G, y_G)$ is located at

$$x_G = \frac{1}{5}(1 + 2.4 + 2 + 0.4 + 0) = 1.16$$
$$y_G = \frac{1}{5}(6.4 + 4 + 0.2 + 0 + 4) = 2.92$$

The polygon is decomposed into 5 triangles of apex $G(x_G, y_G)$. The centroid of triangle 1 is located at

$$x_{G1} = \frac{1}{3}(1.16 + 1 + 2.4) = 1.52$$
$$y_{G1} = \frac{1}{3}(2.92 + 6.4 + 4) = 4.44$$

In a similar way the centroids of other triangles are found and are presented in Table 6.3.

Table 6.3 Coordinates of the triangles' centroids

Triangle	1	2	3	4	5
x-centroid	1.52	1.85333	1.18666	0.52	0.72
y-centroid	4.44	2.37333	1.04	2.30666	4.44

The areas of the triangles are found using Eq. (6.25) and for triangle 1 is given by

$$\begin{aligned} S_1 &= \frac{1}{2}[x_G(y_2 - y_1) + x_2(y_1 - y_G) + x_1(y_G - y_2)] \\ &= \frac{1}{2}[1.16(4 - 6.4) + 2.4(6.4 - 2.92) + 1(2.92 - 4)] = 2.244 \end{aligned}$$

In a similar way the areas for the other triangles are found and are presented in Table 6.4.

Table 6.4 Areas of the various triangles

Triangle	1	2	3	4	5
Area	2.244	2.14	2.62	2.104	1.932

The area of the polygon is found to be

$$S_t = \sum_{i=1}^{5} S_i = 2.244 + 2.14 + 2.62 + 2.104 + 1.932 = 11.04$$

The coordinates of its centroid can be obtained as

$$x_C = \frac{1}{S_t}\sum_{i=1}^{5} S_i x_{Ci} = 1.174925$$

$$y_C = \frac{1}{S_t}\sum_{i=1}^{5} S_i y_{Ci} = 2.825940$$

The difference between the geometric centre and centroid is clear.

6.5.2.2 Volume and Centroid of Elements

The general procedure followed to compute the volume and centroid of a general polyhedron is conceptually simple. The process starts by computing the location of the geometric centre of the polyhedron element and decomposing it into a number of polygonal pyramids. As shown in Fig. 6.21, each polygonal pyramid is formed of the geometric centre as the apex and a polygonal face of the element as the base, with its side faces being triangles.

For a polygonal pyramid, the volume and centroid are readily computed. The volume is calculated as $(1/3) \times \mathit{Base} \times \mathit{Height}$. The base is basically one of the surfaces of the element, while the sub-element pyramid centroid, measured from the centroid of the base, is situated at $1/4$ of the line joining the centroid of the base to the apex of the pyramid. The volume of the polyhedron element is the sum of the volumes of the polygonal pyramids. As for the centroid it is computed as the

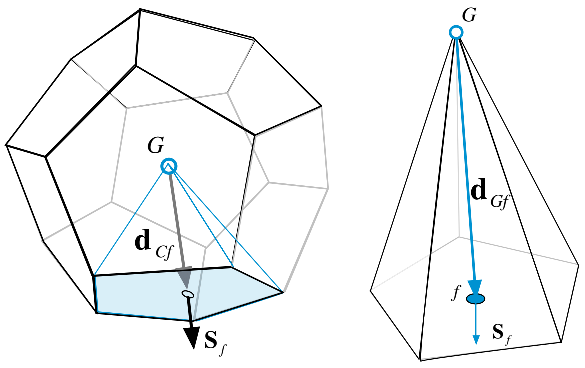

Fig. 6.21 A sub-element pyramid

volume-weighted average of the centroids of the pyramids. Mathematically this is computed from the following relations:

$$\begin{aligned}
\mathbf{x}_G &= \frac{1}{k}\sum_{i=1}^{k}\mathbf{x}_i \\
(\mathbf{x}_{CE})_{pyramid} &= 0.75(\mathbf{x}_{CE})_f + 0.25(\mathbf{x}_G)_{pyramid} \\
V_{pyramid} &= \frac{\mathbf{d}_{Gf}\cdot\mathbf{S}_f}{3} \\
V_C &= \sum_{\sim Sub-pyramids(C)} V_{pyramid} \\
(\mathbf{x}_{CE})_C = \mathbf{x}_C &= \frac{\sum\limits_{\sim Sub-pyramids(C)} (\mathbf{x}_{CE})_{pyramid} V_{pyramid}}{V_C}
\end{aligned} \tag{6.26}$$

Since the centroid and volume of a polygonal pyramid are easily computed, this procedure allows accurate computations of both the volume and centroid of an element.

6.5.2.3 Face Weighting Factor

Consider the one dimensional finite volume mesh system shown in Fig. 6.22. The values of ϕ are known at the control volume centroids C and F, and are to be used to compute the value of ϕ at the interface f.

A simple linear interpolation will result in the following formula:

$$\phi_f = g_f\phi_F + (1 - g_f)\phi_C \tag{6.27}$$

where

$$g_f = \frac{d_{Cf}}{d_{Cf} + d_{fF}} \tag{6.28}$$

The simplicity of this formula does not extend into multi-dimensional situations as in two or three dimensions the circumstances become a bit more complicated. In this case there is not a unique option for the definition of the geometric weighting factors.

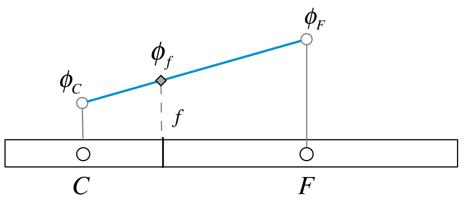

Fig. 6.22 One dimensional mesh system

One choice would be to base the weighting factor on the respective volumes, such that

$$g_f = \frac{V_C}{V_C + V_F} \tag{6.29}$$

This however yields wrong results in certain cases, such as in the configuration shown in Fig. 6.23.

Another difficulty arises when the points C, f, and F are not collinear as depicted in Fig. 6.24a.

A better alternative for such cases, as displayed in Fig. 6.24b, is to base the interpolation on the normal distances to the face, i.e., Cf' and Ff''. Thus the interpolation factor is computed as

$$g_f = \frac{\mathbf{d}_{Cf} \cdot \mathbf{e}_f}{\mathbf{d}_{Cf} \cdot \mathbf{e}_f + \mathbf{d}_{fF} \cdot \mathbf{e}_f} \tag{6.30}$$

where $\mathbf{e}_f$ is the surface unit vector given by

$$\mathbf{e}_f = \frac{\mathbf{S}_f}{\|\mathbf{S}_f\|} \tag{6.31}$$

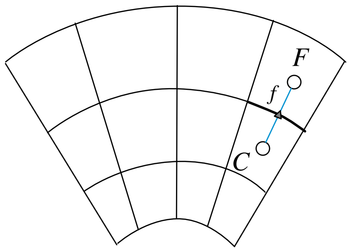

Fig. 6.23 Axisymmetric grid system

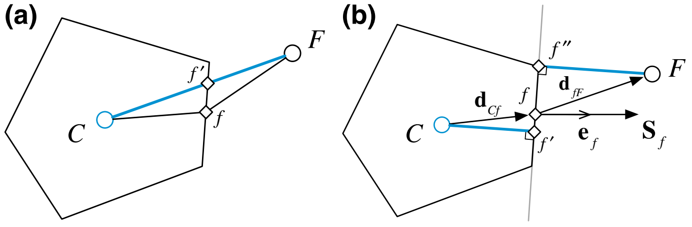

Fig. 6.24 Two dimensional control volume with the points C, f, and F being non collinear

Example 5

Compute the weighing factor using Eqs. (6.26) *and* (6.27) *for the two triangular elements shown in* Fig. 6.25 (Table 6.5).

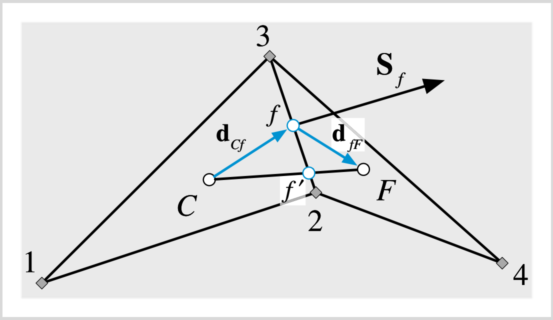

Fig. 6.25 Two neighboring polygonal elements

Table 6.5 Coordinates of the triangular elements for Example 4

Node	1	2	3	4
x	0	1.2	1	2
y	0	0.4	1	0.1

Solution

To calculate the interpolation factor using Eq. (6.26) the volumes of the two elements are needed and are computed as

$$\begin{aligned} V_C &= \frac{1}{2}[x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)] \\ &= \frac{1}{2}[0 + 1.2(1 - 0) + 1(0 - 0.4)] = 0.4 \\ V_F &= \frac{1}{2}[x_2(y_4 - y_3) + x_4(y_3 - y_2) + x_3(y_2 - y_4)] \\ &= \frac{1}{2}[1.2(0.1 - 1) + 2(1 - 0.4) + 1(0.4 - 0.1)] = 0.42 \\ g_f &= \frac{V_C}{V_C + V_F} = \frac{0.4}{0.4 + 0.42} = 0.4878 \end{aligned}$$

The calculation of g_f based on Eq. (6.27) is more involved. The centroids of the two control volumes are required and are calculated as

$$x_C = \frac{1}{3}(0 + 1 + 1.2) = 0.7333 \qquad y_C = \frac{1}{3}(0 + 1 + 0.4) = 0.4666$$

$$x_F = \frac{1}{3}(1.2 + 1 + 2) = 1.4 \qquad y_F = \frac{1}{3}(0.4 + 1 + 0.1) = 0.5$$

The face centroid is also required and is found to be

$$x_f = \frac{1}{2}(1 + 1.2) = 1.1 \qquad y_f = \frac{1}{2}(1 + 0.4) = 0.7$$

The surface vector is calculated as

$$\mathbf{S}_f = (y_3 - y_2)\mathbf{i} - (x_3 - x_2)\mathbf{j} = 0.6\mathbf{i} + 0.2\mathbf{j}$$

The unit vector normal to the face becomes

$$\mathbf{e}_f = \frac{\mathbf{S}_f}{S_f} = \frac{0.6\mathbf{i} + 0.2\mathbf{j}}{\sqrt{0.6^2 + 0.2^2}} = 0.949\mathbf{i} + 0.316\mathbf{j}$$

The distance from the centroids of the control volumes to the face centroid are given by

$$\begin{aligned}\mathbf{d}_{Cf} &= \left(x_f - x_C\right)\mathbf{i} + \left(y_f - y_C\right)\mathbf{j} = 0.3667\mathbf{i} + 0.2334\mathbf{j}\\ \mathbf{d}_{fF} &= \left(x_F - x_f\right)\mathbf{i} + \left(y_F - y_f\right)\mathbf{j} = 0.3\mathbf{i} - 0.2\mathbf{j}\end{aligned}$$

The interpolation factor using Eq. (6.27) is found as

$$g_f = \frac{\mathbf{d}_{Cf} \cdot \mathbf{e}_f}{\mathbf{d}_{Cf} \cdot \mathbf{e}_f + \mathbf{d}_{fF} \cdot \mathbf{e}_f} = \frac{(0.3667\mathbf{i} + 0.2334\mathbf{j}) \cdot (0.949\mathbf{i} + 0.316\mathbf{j})}{(0.6667\mathbf{i} + 0.0334\mathbf{j}) \cdot (0.949\mathbf{i} + 0.316\mathbf{j})} = 0.6556$$

the difference in values is obvious.

6.6 Computational Pointers

6.6.1 uFVM

In ufvm, the processing of all geometric and topological data is performed in one routine denoted by cfdProcessOpenFoamMesh, which is executed right after reading the OpenFOAM® mesh with cfdReadOpenFoamMesh. Reading the OpenFOAM® mesh in its native form, requires reading the various files defining an OpenFOAM® mesh, namely and in that order, *points*, *faces*, *owners*, *neighbours*, and *boundaries* files.

In cfdProcessOpenFoamMesh the face geometry is initially processed (centroid, area and surface vector), as shown in Listing 6.1.

```
%
% Process basic Face Geometry
%
numberOfFaces = theMesh.numberOfFaces;
for iFace=1:numberOfFaces
    iNodes = theMesh.faces(iFace).iNodes;
    numberOfiNodes = length(iNodes);
    %
    % Compute a rough centre of the face
    %
    centre = [0 0 0]';
    for iNode=iNodes
       centre = centre + theMesh.nodes(iNode).centroid;
    end
    centre = centre/numberOfiNodes;

    centroid = [0 0 0]';
    Sf = [0 0 0]';
    area = 0;
    %
    % using the center compute the area and centroid
    % of virtual triangles based on the centre and the
    % face nodes
    %
    for iTriangle=1:numberOfiNodes
        point1 = centre;
        point2 = theMesh.nodes(iNodes(iTriangle)).centroid;
        if(iTriangle<numberOfiNodes)
  point3 = theMesh.nodes(iNodes(iTriangle+1)).centroid;

        else
            point3 = theMesh.nodes(iNodes(1)).centroid;
        end
        local_centroid = (point1+point2+point3)/3;
        local_Sf  = 0.5*cross(point2-point1,point3-point1);
        local_area = cfdMagnitude(local_Sf);

       centroid = centroid + local_area*local_centroid;
        Sf = Sf + local_Sf;
        area = area + local_area;
    end
    centroid = centroid/area;

    %
    theMesh.faces(iFace).centroid = centroid;
    theMesh.faces(iFace).Sf = Sf;
    theMesh.faces(iFace).area = area;
end
```

Listing 6.1 Processing of basic face geometry

This is followed by processing the basic element geometry (Listing 6.2). The element volume is computed by subdividing it into non-overlapping pyramids, whose geometric characteristics can be easily computed.

```
%
%  compute volume and centroid of each element
%
numberOfElements = theMesh.numberOfElements;
for iElement=1:numberOfElements

    iFaces = theMesh.elements(iElement).iFaces;
    %
    % Compute a rough centre of the element
    %
    centre = [0 0 0]';
    for iFace=1:length(iFaces)
      centre = centre + theMesh.faces(iFace).centroid;
    end
    centroid = [0 0 0]';
    Sf = [0 0 0]';
    centre = centre/length(iFaces);
    % using the centre, compute the area and centroid
    % of virtual triangles based on the centre and the
    % face nodes
    %
    localVolumeCentroidSum = [0 0 0]';
    localVolumeSum = 0;
    for iFace=1:length(iFaces)
        localFace = theMesh.faces(iFaces(iFace));
        localFaceSign = theMesh.elements(iElement).faceSign(iFace);
        Sf = localFace.Sf*localFaceSign;
        Cf = localFace.centroid - centre;
        % calculate face-pyramid volume
        localVolume = Sf'*Cf/3;
        % Calculate face-pyramid centre

        localCentroid  =  0.75*localFace.centroid  +
0.25*centre;

        %Accumulate volume-weighted face-pyramid centre
         localVolumeCentroidSum = localVolumeCentroidSum +
localCentroid*localVolume;

          % Accumulate face-pyramid volume
        localVolumeSum = localVolumeSum + localVolume;
    end
    centroid = localVolumeCentroidSum/localVolumeSum;
    volume = localVolumeSum;
    %
    theMesh.elements(iElement).volume = volume;
    theMesh.elements(iElement).centroid = centroid;
end
```

Listing 6.2 Processing of basic element geometry

6.6.2 *OpenFOAM®*

As OpenFOAM® [10] uses an unstructured grid platform, all its geometric entities, such as elements, volumes, areas, and centroids, as well as face weighting factors, have to be evaluated and stored. This section provides an overview of the geometric relations needed in the evaluation of the geometric quantities used in OpenFOAM®.

6.6.2.1 Area and Centroid of Faces

In order to evaluate areas and centers for a generic polygon face, the face is decomposed into a series of triangular faces. The overall polygon face metrics are calculated by summing up the properties of each triangular portion. Thus Eqs. (6.21)–(6.23) have to be applied for each computational cell.

OpenFOAM® constructs centers of faces and calculates their areas in the file "$FOAM_SRC/OpenFOAM/meshes/primitiveMesh/primitiveMeshFaceCentres-AndAreas.C" in which the following function is defined (Listing 6.3):

```
void Foam::primitiveMesh::makeFaceCentresAndAreas
(
    const pointField& p,
    vectorField& fCtrs,
    vectorField& fAreas
) const
…
```

Listing 6.3 Function used for defining centers and areas of faces

The function has three arguments with the first, defined as *const,* representing data read from files, while the second and third arguments designate returned lists of objects of dimensions equal to the number of faces in the domain, containing the centers (fCtrs) and areas (fAreas) of the polygonal faces, respectively.

The *pointField* data is a list of all mesh vertices with each vertex defined using three spatial coordinates. The second data needed is the face definition. In OpenFOAM® this is provided in faceList (Listing 6.4), which is a list of identities of the points defining the faces of the mesh.

```
const faceList& fs = faces();

forAll(fs, facei)
{
    const labelList& f = fs[facei];
    label nPoints = f.size();
…
```

Listing 6.4 List of the identities of points defining faces

Then a *for* loop is performed for each face. Inside the loop, the number of points describing the face and their identities are first read in order to access the coordinates of the corresponding vertices. When a face is defined by only three vertices, then direct calculations of its centroid location and area are performed, as shown in Listing 6.5.

```
…
// If the face is a triangle, do a direct    calculation
for efficiency
// and to avoid round-off error-related problems
if (nPoints == 3)
{
     fCtrs[facei]  =  (1.0/3.0)*(p[f[0]]  +  p[f[1]]  +
p[f[2]]);
     fAreas[facei] = 0.5*((p[f[1]] - p[f[0]])^(p[f[2]] -
p[f[0]]));
}
else
{
…
```

Listing 6.5 Equations used in calculating the geometric center and area of a triangular face

In this case the face center *fCtrs[facei]* is evaluated directly using Eq. (6.21) with $k = 3$, while the face area is calculated using Eq. (6.23) where the symbol "^" represents vector product.

For a generic polygonal shape, the face is first decomposed into triangles. For that purpose, OpenFOAM® defines estimated centers for faces based on the average value of vertices defining the face, as displayed in Listing 6.6.

```
vector sumN = vector::zero;
scalar sumA = 0.0;
vector sumAc = vector::zero;

point fCentre = p[f[0]];
for (label pi = 1; pi < nPoints; pi++)
{
    fCentre += p[f[pi]];
}

fCentre /= nPoints;
```

Listing 6.6 Compute estimated centers for faces

Listing 6.7 indicates that a loop over all faces is performed for calculating the geometric centers and areas of the triangles into which these faces are decomposed.

```
for (label pi = 0; pi < nPoints; pi++)
{
    const point& nextPoint = p[f[(pi + 1) % nPoints]];

    vector c = p[f[pi]] + nextPoint + fCentre;
    vector  n  =  (nextPoint  -  p[f[pi]])^(fCentre  -  p[f[pi]]);
    scalar a = mag(n);

    sumN += n;
    sumA += a;
    sumAc += a*c;
}
```

Listing 6.7 Decomposing the polygonal faces into triangles and calculating the geometric centers and areas of these triangles

This is done by finding the face center "*c*" of each triangle using Eq. (6.21) (the factor $1/3$ is applied later), the face normal vector "**n**" and its magnitude "*a*" that defines the triangle area as stated by Eq. (6.23) (again the factor $1/2$ will be applied later). All metrics of the decomposed triangles are then summed up and normalized in accordance with Eq. (6.22), where now the factors $1/3$ and $1/2$ are applied, as depicted in Listing 6.8.

```
// This is to deal with zero-area faces. Mark very small
faces
// to be detected in e.g., processorPolyPatch.
if (sumA < ROOTVSMALL)
{
    fCtrs[facei] = fCentre;
    fAreas[facei] = vector::zero;
}
else
{
    fCtrs[facei] = (1.0/3.0)*sumAc/sumA;
    fAreas[facei] = 0.5*sumN;
}
```

Listing 6.8 Calculating the centroids and areas of faces

In case of a degenerate face, i.e., when "sumA < a very small number", a rescue value is set for the face.

6.6.2.2 Volume and Centroid of Elements

After computing the normals, areas, and centers of faces, it is possible to evaluate the metrics of cells. Similar to faces, the basic idea for a polyhedral cell is to decompose it into a sum of tetrahedra elements. The operations that calculate volumes and centroids of elements are then defined in the file “$FOAM_SRC/OpenFOAM/meshes/primitiveMesh/primitiveMeshCellCentresAndVols.C” using the function shown in Listing 6.9.

```
void Foam::primitiveMesh::makeCellCentresAndVols
(
    const vectorField& fCtrs,
    const vectorField& fAreas,
    vectorField& cellCtrs,
    scalarField& cellVols
) const
{
…
```

Listing 6.9 Function used to calculate volumes and centers of elements

This function uses four arguments where the first two represent, respectively, centers and areas of faces. The remaining two arguments return objects containing centers and volumes of cells. As shown in Listing 6.10, the “for” loops are defined twice. This redundancy is related to the LDU addressing used in OpenFOAM®, which will be introduced and described in the Chap. 7.

```
…
    // Clear the fields for accumulation
    cellCtrs = vector::zero;
    cellVols = 0.0;

    const labelList& own = faceOwner();
    const labelList& nei = faceNeighbour();

    vectorField cEst(nCells(), vector::zero);
    labelField nCellFaces(nCells(), 0);

    forAll(own, facei)
    {
        cEst[own[facei]] += fCtrs[facei];
        nCellFaces[own[facei]] += 1;
    }

    forAll(nei, facei)
    {
        cEst[nei[facei]] += fCtrs[facei];
        nCellFaces[nei[facei]] += 1;
    }
    forAll(cEst, celli)
    {
        cEst[celli] /= nCellFaces[celli];
    }
```

Listing 6.10 Compute estimated cell centers

In order to calculate the cells centroids and volumes, the first step is the evaluation of $\mathbf{x}_G$, the geometric centers of cells denoted in Listing 6.10 by cEst, using the first relation in Eq. (6.25). Once the $\mathbf{x}_G$ values are obtained, calculations of the pyramids' volumes and cell centroids proceed by applying Eq. (6.25) using the code displayed in Listing 6.11.

```
    forAll(own, facei)
    {
        // Calculate 3*face-pyramid volume
        scalar pyr3Vol =
                     max(fAreas[facei] & (fCtrs[facei] -
cEst[own[facei]]), VSMALL);

        // Calculate face-pyramid centre
        vector pc = (3.0/4.0)*fCtrs[facei] +
(1.0/4.0)*cEst[own[facei]];

        // Accumulate volume-weighted face-pyramid centre
        cellCtrs[own[facei]] += pyr3Vol*pc;

        // Accumulate face-pyramid volume
        cellVols[own[facei]] += pyr3Vol;
    }

    forAll(nei, facei)
    {
        // Calculate 3*face-pyramid volume
        scalar pyr3Vol =
                 max(fAreas[facei] & (cEst[nei[facei]] -
fCtrs[facei]), VSMALL);

        // Calculate face-pyramid centre
        vector pc = (3.0/4.0)*fCtrs[facei] +
(1.0/4.0)*cEst[nei[facei]];

        // Accumulate volume-weighted face-pyramid centre
        cellCtrs[nei[facei]] += pyr3Vol*pc;

        // Accumulate face-pyramid volume
        cellVols[nei[facei]] += pyr3Vol;
    }
```

Listing 6.11 Calculating cell centers and volumes

In the above code, fCtrs corresponds to $\mathbf{x}_{CE}$ while "cEst—fCtrs" corresponds to the distance vector $\mathbf{d}_{Gf}$ displayed in Fig. 6.21. The final values are obtained, as shown in Listing 6.12, by dividing cell centroids by cell volumes and then dividing cell volumes by 3.

```
    cellCtrs /= cellVols;
    cellVols *= (1.0/3.0);
```

Listing 6.12 The final values of cell centroids and volumes

The mesh data structure for uFVM and OpenFOAM® will be described in detail in Chap. 7.

6.7 Closure

The geometric data defining a finite volume mesh were presented in this chapter. It was stressed that the finite volume mesh is not simply the set of non-overlapping elements and nodes. It also includes the set of all geometric quantities with information about their topologies. The collection of all this represents the infrastructure needed by the equation discretization method adopted in this book, namely the Finite Volume Method (FVM).

6.8 Exercises

Exercise 1
Compare the equations presented in the book to the ones used in OpenFOAM® for computing the interpolation weights at the faces, the owner-Neighbor element distances, and the non orthogonal coefficient. These can be found in OpenFOAM® using Doxygen [14] in the functions makeWeights(), makeDeltaCoeffs(), and makeNonOrthDeltaCoeffs().

Exercise 2
Write a program that reads an OpenFOAM® mesh and checks that for each element the sum of the surface vectors is zero.

Exercise 3
Start by reading a mesh into uFVM and then investigate the mesh structure (use **`m = cfdGetMesh`** to get access to the mesh data, and then investigate the structure of an element, a face, and a vertex).

References

1. Thompson JF, Warsi Z, Mastin C (1985) Numerical grid generation. Elsevier Science Publishers, New York
2. Cheng S, Dey T, Shewchuk JR (2012) Delaunay mesh generation. CRC Press, Boca Raton
3. Thompson JF, Soni BK, Weatherill NP (eds) (1999) Handbook of grid generation, Chapter 17. CRC Press, Boca Raton
4. George PL (1991) Automatic mesh generation. Wiley, New York
5. Frey P, George PL (2010) Mesh generation. Wiley, New York
6. Bern M, Plassmann P (2000) Mesh generation. Handbook of computational geometry. Elsevier Science, North Holland
7. Mavriplis DJ (1996) Mesh generation and adaptivity for complex geometries and flows. In: Peyret R (ed) Handbook of computational fluid mechanics. Academic Press, London
8. Thompson JF, Soni BK, Weatherill NP (1998) Handbook of grid generation. CRC Press, Boca Raton
9. Chappell JA, Shaw JA, Leatham M (1996) The generation of hybrid grids incorporating prismatic regions for viscous flow calculations. In: Soni BK, Thompson JF, Hauser J,

Eiseman, PR (eds) Numerical grid generation in computational field simulations. Mississippi State University, MS, pp 537–546
10. OpenFOAM (2015) Version 2.3.x. http://www.openfoam.org
11. Mavriplis DJ (1997) Unstructured grid techniques. Ann Rev Fluid Mech 29:473–514
12. Lerner RG, Trigg GL (1994) Encyclopaedia of physics. VHC, New York
13. Byron F, Fuller R (1992) Mathematics of classical and quantum physics. Dover Publications, Mineola
14. OpenFOAM Doxygen (2015) Version 2.3.x. http://www.openfoam.org/docs/cpp/