Chapter 3

Numerical Techniques

This chapter describes the numerical techniques, which have been used in the scope of this work. All numerical developments have been integrated into an object oriented library called *MOUSE* . This library shall be technically described in chapter 7. The techniques described in the present chapter form the numerical basis of that library. Most of the work has been related to computational fluid mechanics, using finite volume discretizations. These will therefore get special attention in the following descriptions. The goal of the project was to develop a toolbox for numerical computations. This was already kept in mind, while deciding for the numerical techniques to be used. Nevertheless it has been avoided as much as possible to implement any numerical method in a low-level part of the library. It has been tried to keep the methods as flexible as possible. Special physical problems need special numerics of course. It is, however, possible to share a lot of things between different applications. It does not matter if you want to compute elasticity problems, fluid dynamics or electrical fields, you still need a computational grid. If it is intended to be geometrically flexible, this is a nontrivial thing. Many lines of source code

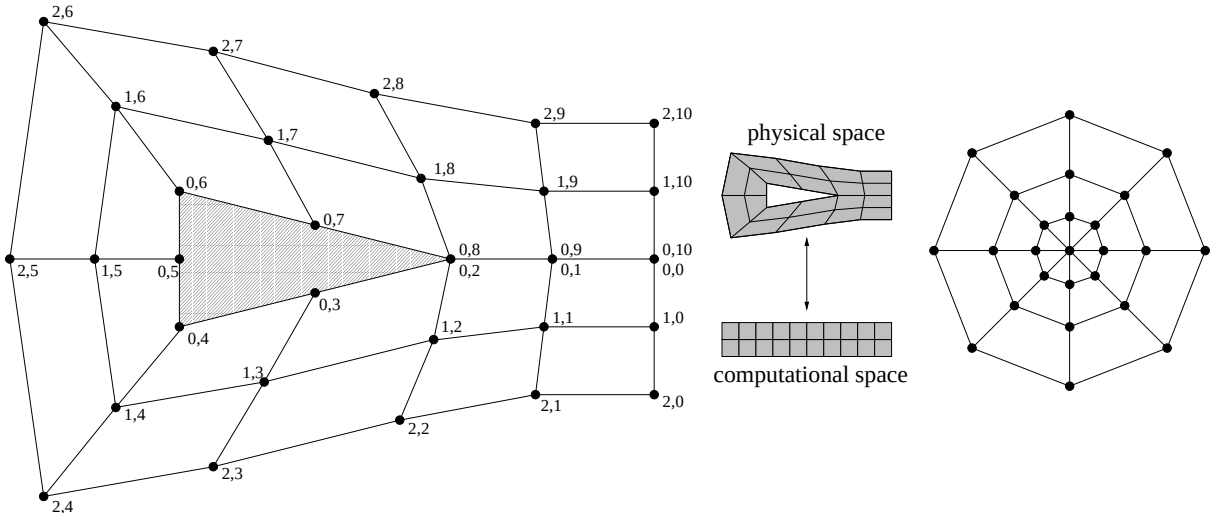

Figure 3.1: Structured grids.

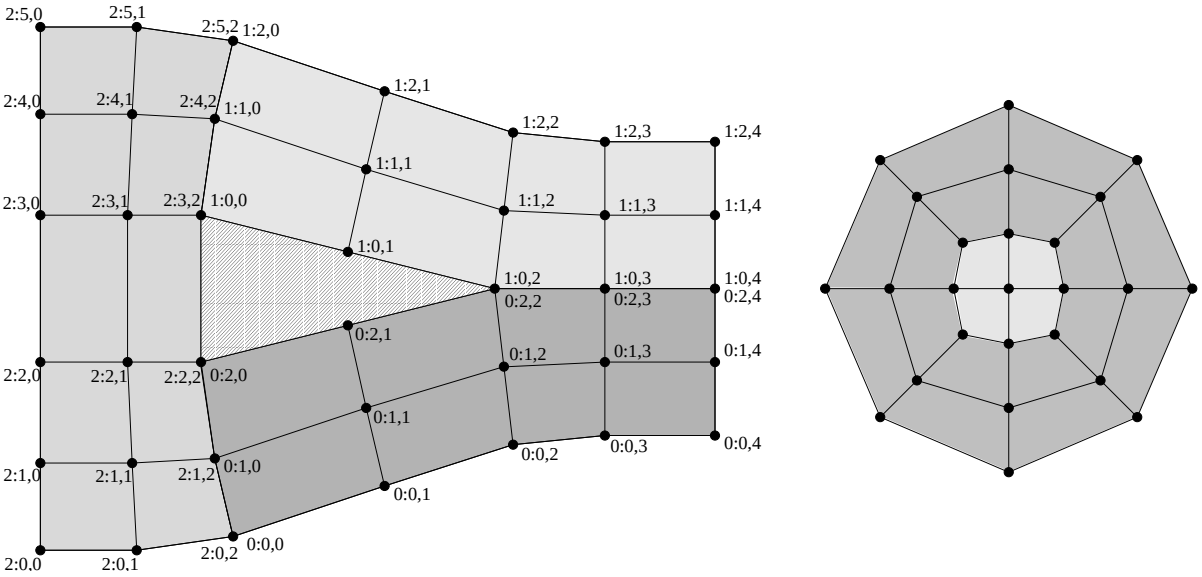

Figure 3.2: Block structured grids.

will go into the handling of grids. Thus it seems obvious that it should be shared for as many applications as possible. The grid is only the most obvious example of this kind. There are many other techniques for which this could be said as well. It will hopefully become clearer in the following text.

3.1 Grid Structure

Apart from a few mesh free discretizations (see for example [2]) every numerical method, aiming at the solution of partial differential equations, needs a grid. Mesh-free discretizations offer some very interesting approaches, but for the vast majority of numerical methods, a grid is needed. There are many different types of grids. The following paragraphs try to give a very brief outline of what is used today. The different grid types shall be compared. A very important criterion for the choice of a method is the ability to use anisotropic refinement. Anisotropic refinement is crucial, especially in three dimensions. For an isotropic grid the following applies for the number of necessary nodes:

$$N_{\text{nds}} \sim \frac{1}{l^{dim}} \tag{3.1}$$

N_{nds} is the number of nodes required, l the geometrical length which shall be resolved and dim the spatial dimension. If the length which has to be resolved halves, for instance, the number of nodes will be eight times as big in the case of a three dimensional grid. This will bring a computer to its limits very quickly. For anisotropic grids, if the elements are properly aligned, the complexity becomes linear. Thus the following is valid for the number of nodes:

$$N_{\text{nds}} \sim \frac{1}{l} \tag{3.2}$$

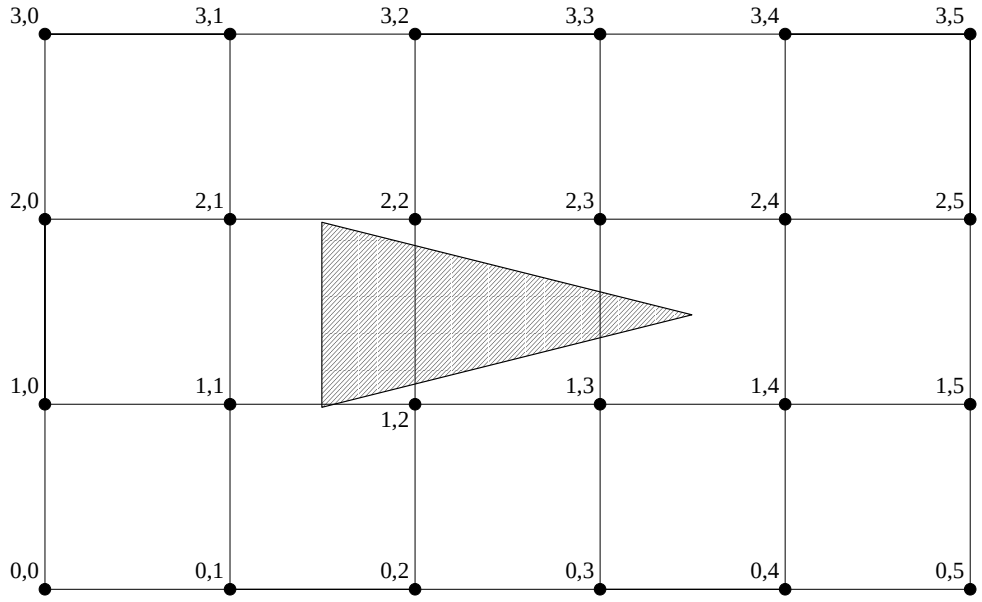

Figure 3.3: A cartesian grid.

3.1.1 Structured Grids

Figure 3.1 shows an example of a structured grid. Structured grids offer a natural orientation of their nodes. Every node can be uniquely identified by a pair of indices in two spatial dimensions, or a triple of indices in three dimensions. Furthermore it is possible to define a transformation between a computational space and the physical space. To realize a finite-difference method, it is possible to compute numerical approximations for derivatives in the computational space and then transform them to the physical space. A structured grid consists entirely of quads in 2D or hexahedra in 3D. Information about neighboring nodes and elements is naturally available. Thus no further data structures are needed to access it. The mesh around the triangle in figure 3.1 shows a so called "C-mesh". It is called that way because its shape reminds remotely to the letter C. The indices of the nodes are shown in the figure. Please note that some nodes exist twice. This is a fact that has to be considered by the numerical algorithm that is used on the grid. Structured grids have some drawbacks, however. Singularities might arise, depending on the geometry which shall be meshed. These singularities in the grid will hence require a special treatment in the numerical algorithm. A simple circular region, for example, cannot be meshed using a structured grid (see figure 3.1). The central node of the circular mesh is such a singular node. It can be imagined, that the computation of derivatives, using the finite difference method, is very difficult for such nodes. For complicated geometries it might become very difficult to generate a structured grid. Imagine an entire aircraft should be meshed. This will be simply impossible to do with a single structured grid. Furthermore the quality of the generated elements can become quite bad, depending on the geometrical constraints. The left part of the mesh around the triangle shows fairly distorted elements. Such elements might have a negative influence on the behavior of numerical methods.

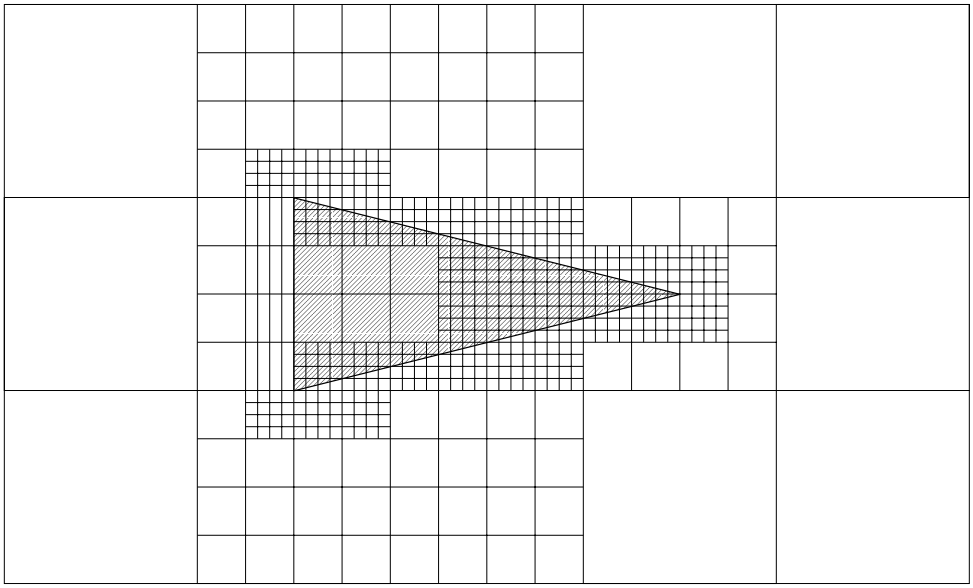

Figure 3.4: An *AMR*-like refined cartesian grid.

3.1.2 Block Structured Grids

An example for block structured grids can be found in figure 3.2. Block structured grids are an approach to overcome a few of the drawbacks from structured grids. A look at the circular mesh in figure 3.2 shows, that by using more than one grid it is sometimes possible to avoid singular nodes. Now, however, the regions where two or more blocks touch each other need special treatment. There are, of course, many possibilities to realize these block interfaces. In figure 3.2 the block number is shown, together with the indices of the nodes. Practical applications showed, that is is possible to create block structured grids for very complex geometries. Experiences also show that the effort, which has to be put into mesh generation can be very high sometimes.

3.1.3 Cartesian Grids

Figure 3.3 shows an example of a cartesian grid. Cartesian grids represent the most simple grid type and they are a subset of the structured grids. A cartesian grid is completely orthogonal and its grid lines are aligned with the Cartesian coordinates. Probably the most significant advantage of Cartesian grids is the simple mesh-generation. Another big advantage is, higher order approximations are a lot easier to realize. Generally spoken discretizations on Cartesian grids are fairly straightforward. To describe complex boundaries on Cartesian grids can be very difficult, because it cannot be assured that the physical boundary is formed by edges of the mesh. Usually the boundary lies somewhere in between two nodes. It is possible to realize the boundary conditions using some sort of interpolation technique. Depending on the type of boundary condition this can become a very delicate task. Another possibility can be used in the case of a finite volume scheme. Assuming a cell-centered scheme, it is possible to use partial cells in regions where a

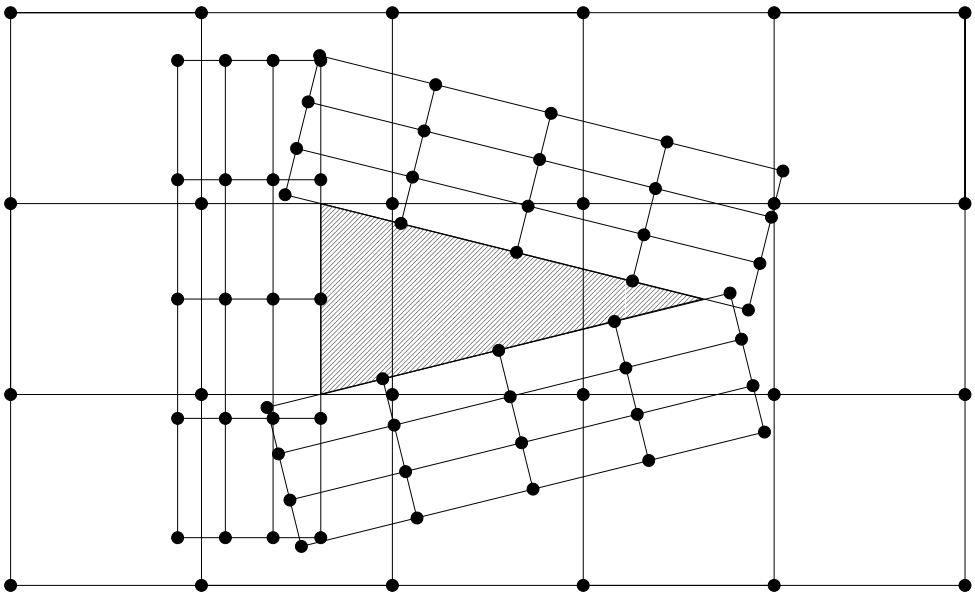

Figure 3.5: A chimera grid.

Cartesian cell is cut by the physical boundary. Some interesting refinement strategies exist for Cartesian or structured grids (see figure 3.4) Anisotropy is only possible in the already existing directions of the grid. If a Cartesian mesh is used as basic grid, this will hence only be the Cartesian directions. If, for instance, an oblique shock wave has to be resolved, only isotropic refinement can be used. For many applications, however, where anisotropy is not required, Cartesian grids might be an easy to use alternative to the other more complicated approaches.

3.1.4 Chimera (Overlaying) Grids

So called chimera grids are an interesting approach to combine the advantages of structured grids in terms of addressability and natural ordering with a high geometric flexibility. Chimera, or overset grids, consist of different structured patches. The exchange between the different grids is realized by interpolation. It is not required that the grids match on their interfaces (see figure 3.5).

3.1.5 Triangular Grids

Triangular grids consist entirely of triangles or tetrahedra in the case of a three dimensional mesh. These grids offer no natural orientation. They are formed by a set of nodes and a set of rules how to combine them to elements. See figure 3.6 for an example of a triangular grid. The numbering of the nodes, as well as the number triples that form the triangles are shown in the example grid. Triangular grids are mainly used together with finite volume or finite element schemes. Finite difference discretizations are not impossible, but they loose their simplicity and thus their major advantage of unstructured grids. Geometrically triangular grids are the most flexible mesh type. The process of mesh generation can be

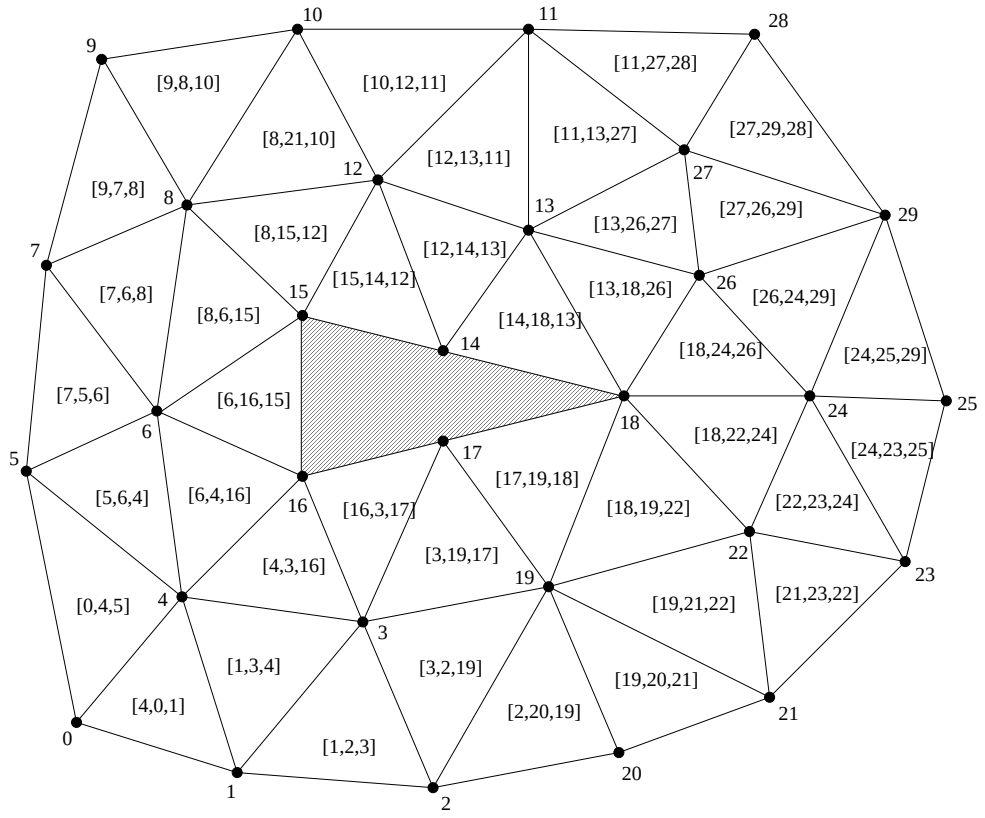

Figure 3.6: A triangular grid.

more easily automatized then for most other grid types. Still it is possible to accurately model complex boundaries. Furthermore they offer the possibility to be anisotropically refined in certain regions [3]. Unfortunately anisotropic triangular grids show significant problems with some discretizations for viscous terms (friction terms for computational fluid dynamics). Resolving boundary layers in three dimensional flow, using very flat tetrahedra is almost impossible for certain discretizations [1].

3.1.6 Unstructured Hybrid Grids

Hybrid grids are flexible in terms of the element types used. A simple example is shown in figure 3.7. Similar to purely triangular grids, they offer no natural orientation. A hybrid grid can consist of different element types. The most commonly used are

- triangles
- quads

for two dimensional problems and

- tetrahedra
- pyramids

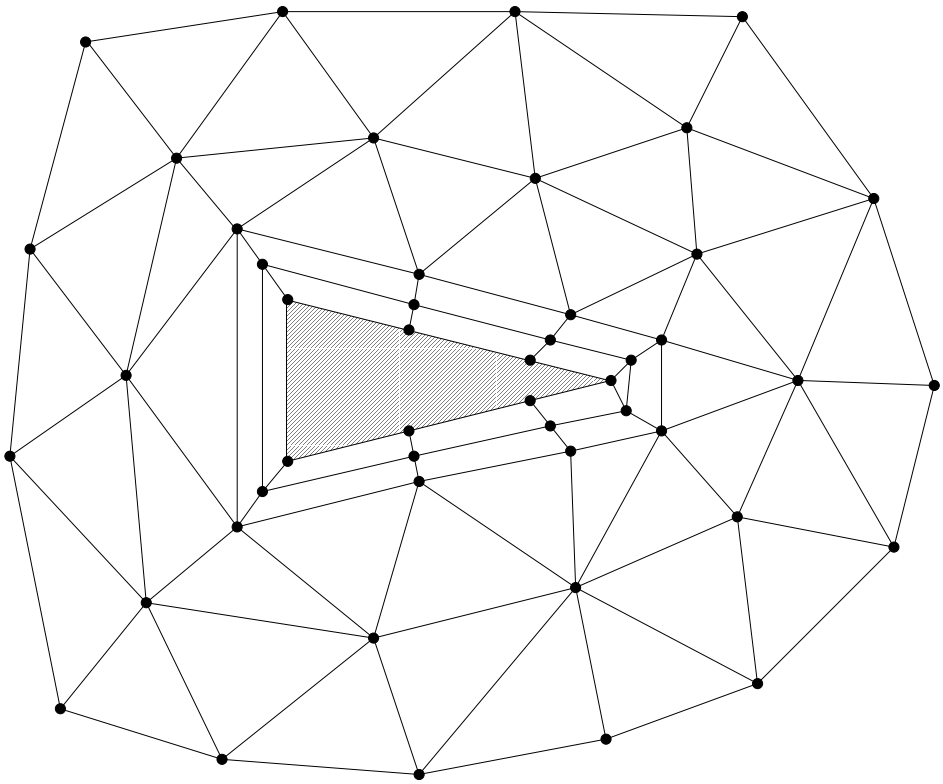

Figure 3.7: A multi element unstructured grid.

- prisms (wedges)
- hexahedra

for three dimensional grids. Unstructured hybrid techniques can be used to describe structured or block structured grids as well. Thus they are probably one of the most flexible grid types available. The meshes from figures 3.1 and 3.2 could easily be easily handled by a solver using the hybrid approach. Of course a Cartesian mesh can also be described. For Cartesian grids, however, there are a number of advantages which cannot be easily used by a solver which has been designed for unstructured hybrid grids. There are some approaches, though, which might make these advantages accessible too. One of the main advantages of hybrid grids is, that anisotropic features can be resolved, using bilinear elements, which are aligned in the direction of the anisotropy. This is a very important advantage, especially for large three dimensional problems.

3.2 Finite Volume Discretization

Here a brief outline of the finite volume method shall be given. Furthermore it has been tried to find a general and flexible description which is suitable for a generic library, as it will be described in chapter 7. Generally it can be said that finite volume methods

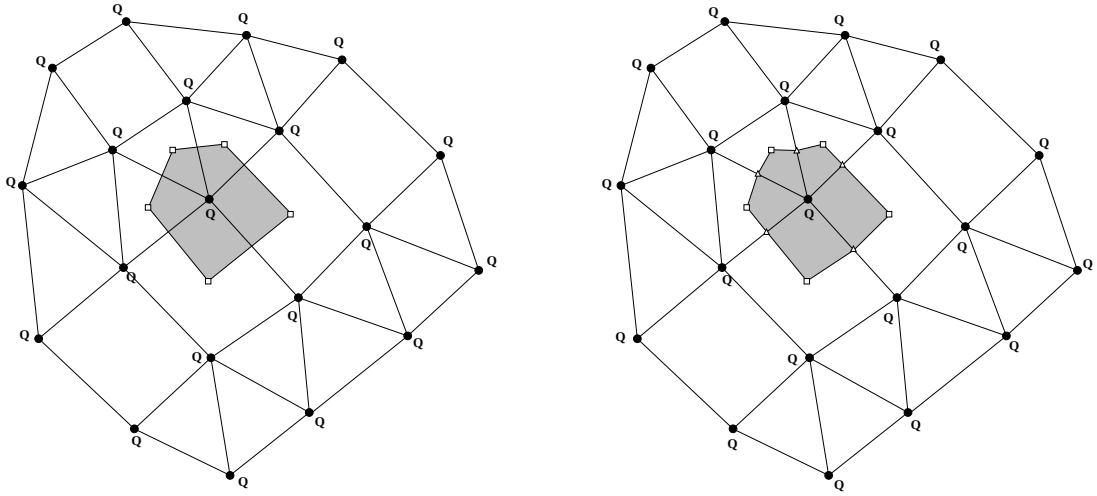

Figure 3.8: Node-centered control volumes.

subdivide the physical space into a finite number of volumes. This can be done in various ways, of which most are based on computational grids as they have been introduced in the last paragraph. The integral equation (2.1) will then be formulated for all finite volumes. Furthermore it is assumed that the variables of state are piecewise constant for the discrete control volumes. Thus an integral over a discrete control volume can be formulated as a simple product:

$$\left[\int_V f(\mathbf{x})\,\mathrm{d}V\right]_i = f(\mathbf{x}_i)\,(\Delta V)_i\,. \tag{3.3}$$

Any conservation law (see (2.2)) can be written in this form:

$$\int_V \frac{\partial \mathbf{q}}{\partial t}\,\mathrm{d}V \quad = \quad \int_V \mathrm{res}(\mathbf{q})\,\mathrm{d}V\,. \tag{3.4}$$

The spatial operator res combines all fluxes and source terms. A finite volume discretization of (3.4) becomes:

$$\int_V \left[\frac{\partial \mathbf{q}}{\partial t}\right]_i \quad = \quad \mathrm{Res}(\mathbf{Q})_i\,. \tag{3.5}$$

The discrete spatial operator $\mathrm{Res}(\mathbf{Q})_i$ already represents integrated values (opposed to res from (3.4)) and it combines the discretized fluxes (surface integrals) and source-terms (volume integrals). Please note that the expression $[\frac{\partial \mathbf{q}}{\partial t}]_i$ denotes a discretized first derivative in time. The actual time discretization may vary. Discretized surface integrals, which occur in $\mathrm{Res}(\mathbf{Q})_i$, will find special mention in paragraph 3.2.2.

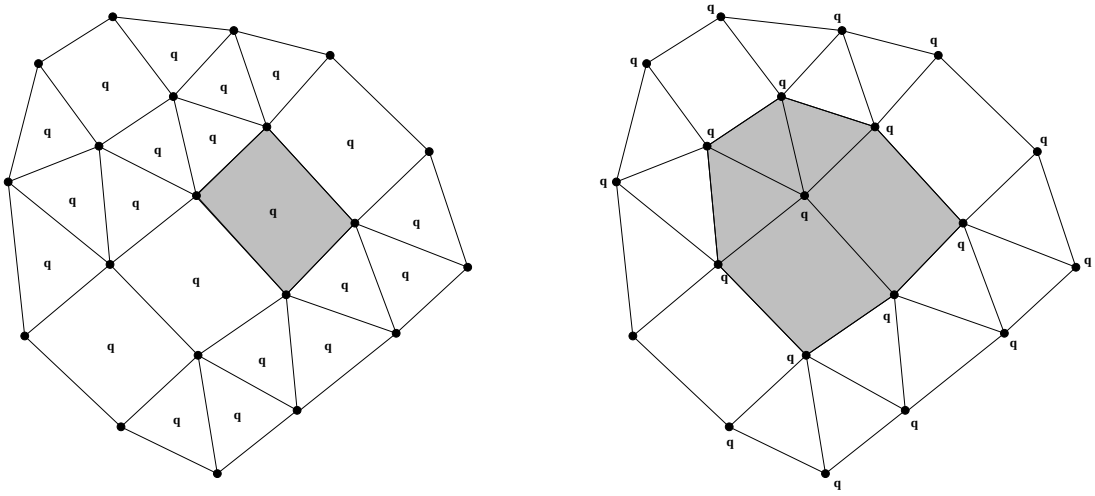

Figure 3.9: Cell-centered (left) and cell-vertex (right) control-volumes.

3.2.1 Different Control Volume Constructions

Discrete control volumes of a mesh can be created using different strategies. Three basic methods exist:

- cell-centered
- node-centered
- cell-vertex

These different control volume types shall be very briefly explained here.

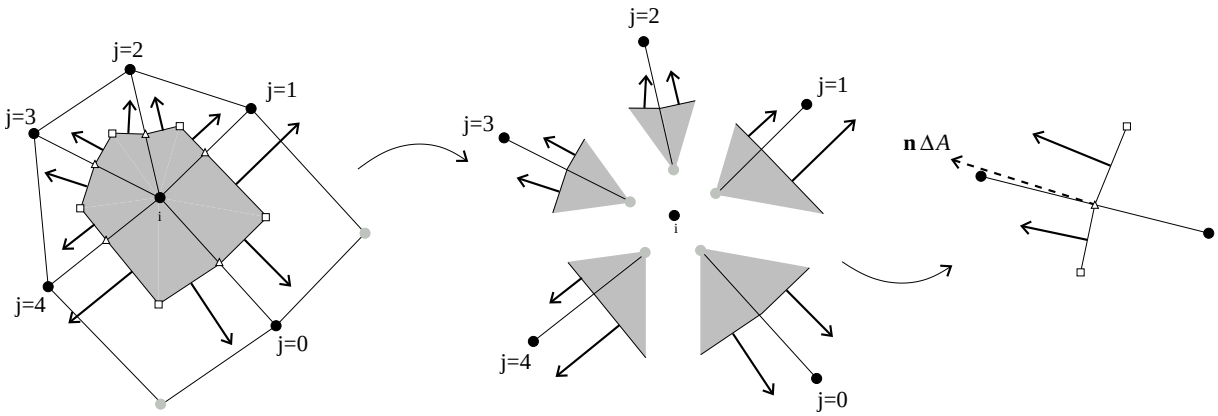

Figure 3.10: A 2D control volume.

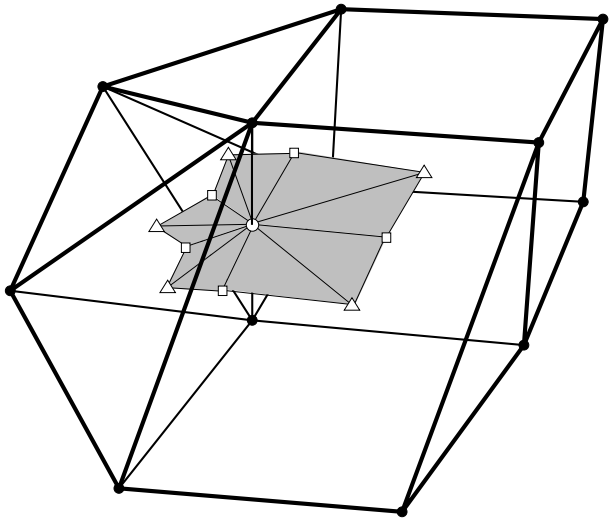

Figure 3.11: A 3D computational molecule.

Cell-Centered Control Volumes

Cell-centered control-volumes are probably the most obvious approach. The control-volumes are directly formed by the elements (cells) of the grid. The variables of state will hence reside in the cells and discretized fluxes will be exchanged across the bounding edges of the elements. One of the main advantages of a cell-centered arrangement is that the computational boundary is identical with the cell boundaries of those cells next to it. A disadvantage, however, is that boundary conditions for the variables themselves, rather than fluxes, are difficult to formulate. This is due to the fact that no discrete variables are available on the boundary itself.

Node-Centered Control Volumes

For node-centered schemes, the variables are stored on the nodes of the mesh. Different strategies exist to construct control volumes around the nodes of an unstructured grid. Figure 3.8 shows two ways of constructing such a volume around a node. Figure 3.8/a shows a very obvious method. By connecting the middle points of the surrounding elements a discrete control volume can be obtained. To define the middle point of an element, once again, several different definitions exist. The barycenter can be used, or the middle point can be defined using the arithmetic mean value of all node coordinates. A different approach is to connect the middle points of the elements with the middle points of the edges, as it can be seen in figure 3.8/b. The later proved to be a fairly accurate and robust approach.

Cell-Vertex Control Volumes

Cell-vertex arrangements try to combine the advantages of cell-centered and node-centered methods. As it is done in node-centered algorithms, the variables are stored on the nodes.

Thus it is possible to prescribe boundary conditions for the variables directly, since nodes exist on the computational boundary. The control-volume for a vertex is formed by all adjacent elements to a it. Fluxes are then evaluated across the edges, which are not internal to the control-volume.

General Description

Any conformal mesh, using any control volume arrangement, can be described using a graph structure. Every control volume is linked to a number of neighboring control volumes. These neighboring volumes do not need to be direct neighbors on the element level. For cell vertex arrangements, for example, this is not the case. Figure 3.10 shows a detailed view of the node centered strategy, which has been used in the scope of this work. As said at the beginning of this chapter, it has always been tried to keep the developments as flexible as possible. Although the control volume type from figure 3.10 has been used, different approaches would be possible. Many developments could be reused for different control volume arrangements as well. It would, for instance, be possible to use the flux formulations for compressible gas dynamics, which shall be described later, with other spatial constructions.

3.2.2 Discretized Surface Integrals

A surface integral will be discretized as a sum of discrete contributions:

$$\left[\oint_{\partial V} f(\mathbf{q})\mathbf{n}\mathrm{d}A\right]_i = \sum_{j=1}^{N_{\mathrm{ngh},i}} \left[\int_{(\Delta A)_{(i,j)}} f(\mathbf{q})\,\mathbf{n}\,\mathrm{d}A\right]_{(i,j)} . \tag{3.6}$$

$f(\mathbf{q})$ is the expression which shall be integrated. Many methods assume that the variables of state $\mathbf{q}$ are constant for a discrete surface segment. If, furthermore, it is assumed that the normal vector $\mathbf{n}_{(i,j)}$ is constant for $(\Delta A)_{(i,j)}$, the integral can be computed as follows:

$$\left[\oint_{\partial V} f(\mathbf{q})\mathbf{n}\mathrm{d}A\right]_i = \sum_{j=1}^{N_{\mathrm{ngh},i}} \left(\left[f(\mathbf{q})\right]_{(i,j)}\ \mathbf{n}_{(i,j)}\,\Delta A_{(i,j)}\right). \tag{3.7}$$

Crucial for a good discretization is the formulation of the discrete function $\left[f(\mathbf{q})\right]_{(i,j)}$. Construction of conservative schemes is easy, using this formulation. A surface segment always represents a link between two control volumes and thus the computed contribution from (3.7) can be used for both control volumes, positive for one and negative for the other volume.
Figure 3.10 shows a typical arrangement, as it has been used in the scope of this work. The smallest computational unit is an edge, and thus it will be called computational molecule. A surface integral contains contributions from several computational molecules. Figure 3.11 illustrates how such a molecule is constructed in a three dimensional computation. The small triangles denote the centers of the elements (two tetrahedra, a pyramid, a prism

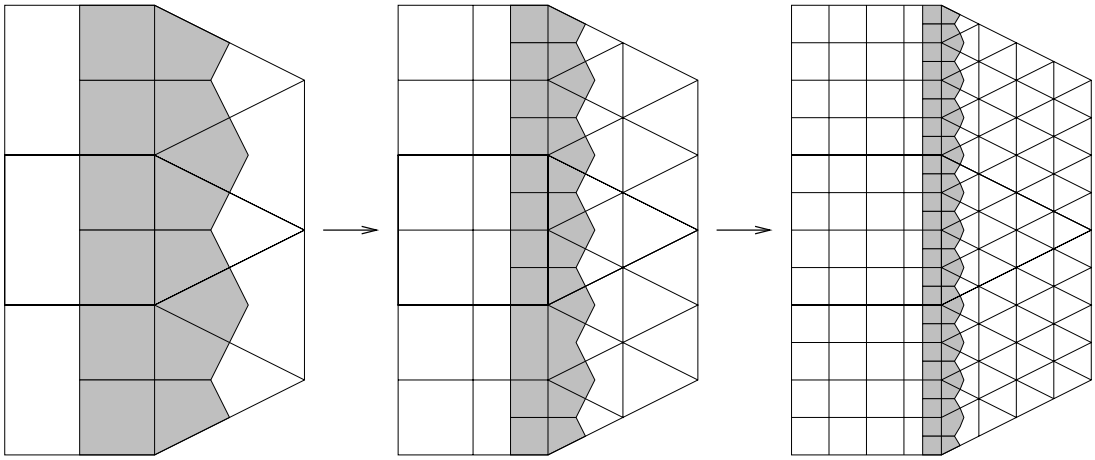

Figure 3.12: Erroneous control-volumes.

and a hexahedron in this example). Small squares mark the centers of the adjacent faces to the edge which forms the computational molecule. The whole faceted surface will be combined and stored as a single normal vector.

Accuracy

For simplex grids it can be shown that the computational molecules, which can be seen in figure 3.10 and 3.11 lead to consistent discretizations. Simplex grids means, grids consisting entirely of triangles for two dimensional grids, or tetrahedra for three dimensional problems. Appendix C shows a simple demonstration of the consistency for triangular grids. For arbitrary hybrid or curvilinear structured grids the consistency can not be shown on this level. It is, however, possible to demonstrate the consistency of orthogonal grids, using either quads, prisms or hexahedra as elements. On a Cartesian grid a

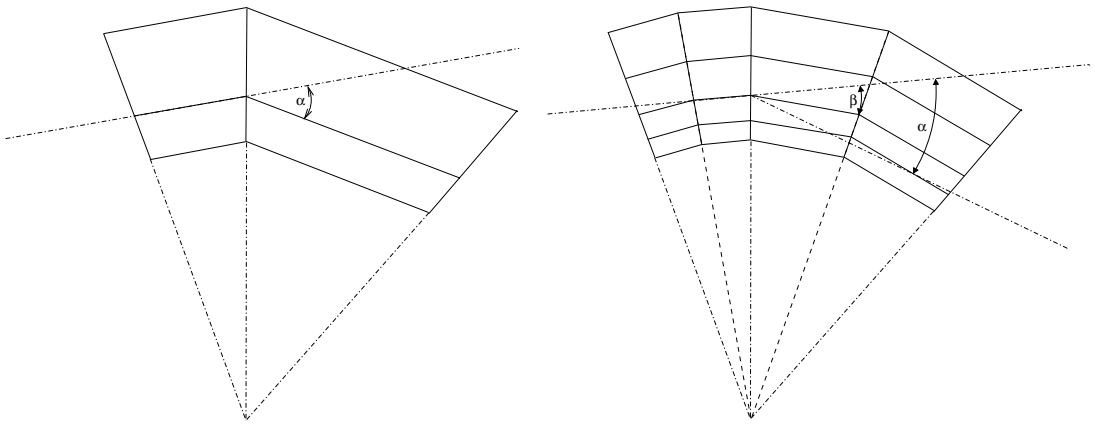

Figure 3.13: Orthogonal grids.

finite volume discretization is identical with a finite difference discretization and it is of course consistent. Equidistant Cartesian grids, as well as equilateral triangulated grids, are ideal to achieve second order accurate discretizations. Problems arise if the grid lines are not orthogonal to each other. An additional error is introduced, which is a function of the local angle (see figure 3.13). If the mesh resolution will be increased, however, this angle will decrease. In figure 3.13 it can be seen what happens, if the local step length is reduced by a factor of two. The angle between the edges will also reduce by the same factor (compare α and β in figure 3.13). There are still configurations for which this is not true. If a sharp geometrical feature (e.g. a trailing edge of an airfoil) has to be resolved, the angle will remain the same no matter how fine it is resolved. But there is only a finite number of those features and the total error will decrease, due to the decreasing fraction of problematic cells. In regions where orthogonal meshes are connected to triangulated ones, vertices exist that share elements of different types. Vertices of this type introduce additional errors, which will unfortunately not vanish with a decreasing step size. These regions, however, do not increase if the total number of nodes for a mesh is increased. Basically the connection can always be done with one layer of vertices. The error, introduced into a simulation by such a connection layer, is dependent on the control volume size of the erroneous nodes. If the resolution is improved, the fraction of erroneous control volumes reduces and thus the total error is reduced as well (see figure 3.12). This is exactly the same argumentation as for the sharp geometric features above. Practical computations show almost no problems when using hybrid grids. Results are comparable or better than what can be obtained with triangulated grids. In particular, boundary layers can be anisotropically resolved using hybrid grids. Anisotropically triangulated grids introduce large errors for second derivatives (e.g. in viscous or diffusive flux formulations).

Upwind Discretizations

For hyperbolic or parabolic equations it is often desired to construct schemes that take into account the direction of the local characteristics of the equation to be solved. Consider the following equation in one spatial dimension:

$$\frac{\partial \mathbf{q}}{\partial t} + u\frac{\partial \mathbf{q}}{\partial x} = 0 \quad , \quad \frac{\partial x}{\partial t} = u \tag{3.8}$$

Assuming that a set of equidistant discrete points is given, the following finite difference scheme can be used.

$$\frac{\mathbf{q}_i^{n+1} - \mathbf{q}_i^n}{\Delta t} + u\Delta_x = 0 \quad , \quad \Delta x = \begin{cases} \frac{1}{h}\left(\mathbf{q}_i^n - \mathbf{q}_{i-1}^n\right) & \text{for} \quad u <= 0 \\ \\ \frac{1}{h}\left(\mathbf{q}_{i+1}^n - \mathbf{q}_i^n\right) & \text{for} \quad u > 0 \end{cases} \tag{3.9}$$

With i as the point index, the time-step Δt and the spatial step h, this describes a very simple first order accurate (space and time) scheme. Please note that for a structured and one-dimensional grid the nodes $i-1$ and $i+1$ are the left and right neighbors of a node i. It is called an upwind scheme, since it takes into account the characteristic u. Equation

(3.8) can be used to describe a one-dimensional scalar convection problem. u would then be the local velocity. Thus differencing would always be done in the "upstream" direction, hence the name upwind-scheme. Opposed to a scheme using a central difference, this will be stable without artificial damping. To further illustrate the motivation for upwind discretizations, (3.9) shall be written in the following form:

$$\frac{\mathbf{q}_i^{n+1} - \mathbf{q}_i^n}{\Delta t} + u \left(\frac{\mathbf{q}_{i+1}^n - \mathbf{q}_{i-1}^n}{2h} \right) - \frac{h}{2}|u| \left(\frac{\mathbf{q}_{i+1}^n - 2\mathbf{q}_i^n + \mathbf{q}_{i+1}^n}{h^2} \right) = 0 \tag{3.10}$$

Please note that (3.9) and (3.10) describe exactly the same discretization. (3.10) can be interpreted as a central discretization, plus an artificial damping term, which is dependent on the characteristic u and the spatial step-width h.
For multi-dimensional problems on unstructured grids, however, the construction of upwind schemes is not always as obvious as demonstrated here. As already stated in the previous paragraph, the computation of discrete surface integrals relies on the formulation on finite surface segments, which connect two control volumes. To compute the term $f(\mathbf{q})$ from (3.7), it is necessary to reconstruct $\mathbf{q}$ on the interface. To enable upwind discretizations, one-sided reconstructions have to be computed. For any interface, connecting two finite volumes, $f(\mathbf{q})$ becomes a function of $\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}$, $\tilde{\mathbf{q}}^{\rightarrow}_{(i,j)}$ and $\tilde{\mathbf{q}}^{\leftrightarrow}_{(i,j)}$. These symbols represent the following:

- $\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}$ is a value of $\mathbf{q}$, which has been reconstructed on the interface (i,j), where information coming from the volume i dominates the reconstruction. An extrapolation is a good example of such a onesided construction.

- $\tilde{\mathbf{q}}^{\rightarrow}_{(i,j)}$ represents the same coming from the other side of the interface (i.e. coming from the volume $\{i,j\}$).

- $\tilde{\mathbf{q}}^{\leftrightarrow}_{(i,j)}$ denotes a central reconstruction (e.g. a simple average value).

Details on how such reconstructions can be formulated will be given in the next paragraph. (3.7) hence becomes:

$$\left[\oint_{\partial V} f(\mathbf{q}) \mathbf{n} \mathrm{d}A \right]_i = \sum_{j=0}^{N_{\mathrm{ngh},i}} \left([f](\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}, \tilde{\mathbf{q}}^{\rightarrow}_{(i,j)}, \tilde{\mathbf{q}}^{\leftrightarrow}_{(i,j)}) \, \mathbf{n}_{(i,j)} \, (\Delta A)_{(i,j)} \right) \tag{3.11}$$

with $[f](\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}, \tilde{\mathbf{q}}^{\rightarrow}_{(i,j)}, \tilde{\mathbf{q}}^{\leftrightarrow}_{(i,j)})$ being the numerical formulation on the interface (i,j). As another example a multi-dimensional convection equation, with a constant convection speed $\mathbf{v}$, shall be considered. The integral form of such an equation can be written as:

$$\int_V \frac{\partial \mathbf{q}}{\partial t} \mathrm{d}V \quad + \quad \oint_{\partial V} (\mathbf{v} \cdot \mathbf{q}) \, \mathbf{n} \, \mathrm{d}A \quad = \quad 0 \tag{3.12}$$

A simple finite volume discretization of (3.12) would be the following:

$$\frac{\mathbf{q}_i^{n+1} - \mathbf{q}_i^n}{\Delta t}(\Delta V)_i + \sum_{j=1}^{N_{\mathrm{ngh},i}} \mathbf{h}_{(i,j)} = 0\,,$$

$$\mathbf{h}_{(i,j)} = \begin{cases} \tilde{\mathbf{q}}^{\rightarrow}_{(i,j)}(\mathbf{v}\,\mathbf{n})_{(i,j)}(\Delta A)_{(i,j)} & \text{for} \quad (\mathbf{v}\,\mathbf{n})_{i,j} <= 0 \\ \tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}(\mathbf{v}\,\mathbf{n})_{(i,j)}(\Delta A)_{(i,j)} & \text{for} \quad (\mathbf{v}\,\mathbf{n})_{i,j} > 0 \end{cases}. \tag{3.13}$$

If $\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}$ and $\tilde{\mathbf{q}}^{\rightarrow}_{(i,j)}$ are properly reconstructed, this describes an upwind discretization, similar to the one-dimensional example. For a simple first-order scheme, a representation showing the artificial damping, can be obtained. With the following simple reconstruction

$$\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)} = \mathbf{q}_i\,, \quad \tilde{\mathbf{q}}^{\rightarrow}_{(i,j)} = \mathbf{q}_{\{i,j\}}, \tag{3.14}$$

(3.13) can now be written as

$$\frac{(\Delta\mathbf{q})_i}{\Delta t}(\Delta V)_i + \sum_{j=1}^{N_{\mathrm{ngh},i}} \left(\frac{\mathbf{q}_{\{i,j\}} + \mathbf{q}_i}{2}(\mathbf{v}\,\mathbf{n})_{(i,j)} - \frac{\mathbf{q}_{\{i,j\}} - \mathbf{q}_i}{2}|\mathbf{v}\,\mathbf{n}|_{(i,j)} \right)(\Delta A)_{(i,j)} = 0 \tag{3.15}$$

$$\Leftrightarrow \frac{(\Delta\mathbf{q})_i}{\Delta t}(\Delta V)_i + \sum_{j=1}^{N_{\mathrm{ngh},i}} \left(\frac{\mathbf{q}_{\{i,j\}} + \mathbf{q}_i}{2}(\mathbf{v}\,\mathbf{n})_{(i,j)}(\Delta A)_{(i,j)} \right) \tag{3.16}$$

$$- \sum_{j=1}^{N_{\mathrm{ngh},i}} \left(\frac{\mathbf{q}_{\{i,j\}} - \mathbf{q}_i}{2}|\mathbf{v}\,\mathbf{n}|_{(i,j)}(\Delta A)_{(i,j)} \right) = 0 \tag{3.17}$$

$$\Leftrightarrow \frac{(\Delta\mathbf{q})_i}{\Delta t}(\Delta V)_i + \sum_{j=1}^{N_{\mathrm{ngh},i}} [\mathbf{h}]_{\mathrm{ctr},i,j} - \left[\nabla^2\mathbf{q}\right]_i = 0. \tag{3.18}$$

It can be seen, that the flux integral can be decomposed into a central flux plus an artificial damping term. This is very similar to the one-dimensional example before. The Laplace-like operator $[\nabla^2\mathbf{q}]_i$, however, is not a consistent discretization for the Laplace operator. Only for equidistant grids, this will become a second-order accurate Laplace discretization. It does, however, behave like a Laplacian damper.

3.2.3 Discretized Gradients

Different strategies exist to compute gradients on unstructured grids and two methods shall be described briefly in this paragraph.

Based on the Divergence Theorem

A very obvious method to compute gradients, using a finite volume approach, is to evaluate the following equation:

$$\int_V (\nabla\,\phi\,\mathrm{d}V) = \oint_{\partial V} \phi\,\mathbf{n}\,\mathrm{d}A\,. \tag{3.19}$$

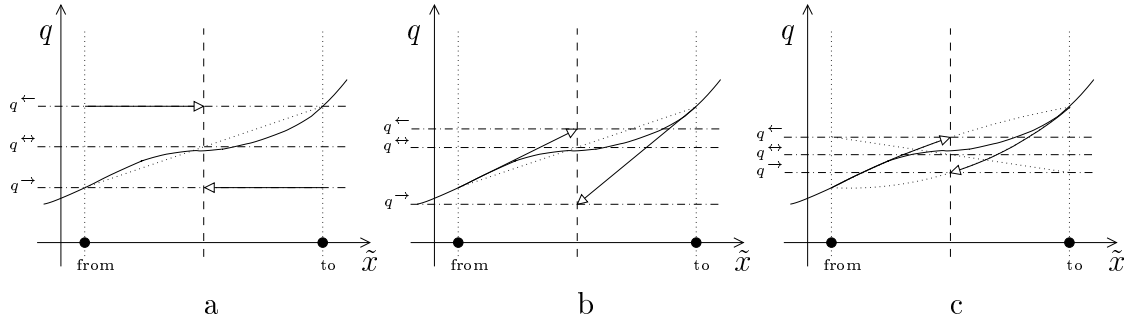

Figure 3.14: Different variable reconstructions.

(3.19) can be derived, using Gauss' divergence theorem (see appendix B.1 for details). To evaluate the gradient, using (3.19) the values have to be linearly reconstructed on the middle of the edges. In this case a central discretization, meaning simple average values, is appropriate. On orthogonal, or perfectly triangulated grids, this leads to a second order accurate scheme. This method has, however, one big disadvantage. For boundary nodes a good reconstruction of the values is rather complicated. Simple averaging on an edge will lead to severely wrong gradients.

Based on the Least Squares Approach

Another possibility to compute gradients is to locally approach a discrete function by a polynomial function. Usually this would be a linear approach. To compute the discrete gradient on a node, all its neighboring nodes would be considered. It will be fit, by minimizing the sum of the error squares (difference between actual and fitted values). The main advantage of this method is its robustness. For a two dimensional problem it is sufficient to have two linearly independent edges (connections from node i to its neighbors). In three dimensions, three connections are needed of course. This is always fulfilled, unless the mesh is degenerated. The later case would not allow any computation and thus is no problem. For a higher than first order polynomial approach, however, more neighbors would be needed. See appendix B.2 for a more detailed description of the linear least squares approach to compute gradients.

3.2.4 Variable Reconstruction

As it has been outlined in paragraph 3.2.2 a reconstruction of the variables on cell interfaces is necessary to compute discretized surface integrals. The algorithm, which has been used in the scope of this work, requires a reconstruction prior to any flux evaluation. In fact the reconstruction mechanism is completely independent from the flux evaluation. Values for $\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}, \tilde{\mathbf{q}}^{\rightarrow}_{(i,j)}$ and $\tilde{\mathbf{q}}^{\leftrightarrow}_{(i,j)}$, on which flux routines can later rely on, have to be provided. A very simple approach is to use the values, available on the nodes, directly for the left and right-sided values $\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}$ and $\tilde{\mathbf{q}}^{\rightarrow}_{(i,j)}$. A value $\tilde{\mathbf{q}}^{\leftrightarrow}_{(i,j)}$ can be obtained by simply averaging the values on the vertices. To obtain a discretization of a formal second order, an

extrapolation of the variables, using their centrally computed gradients, can be used. By speaking of a formal second order it is meant, that the second order can only be achieved on a uniform (e.g. Cartesian or equilaterally triangular) grid. All reconstruction schemes, which have been used in the scope of this work, rely on the reconstruction along an edge. This means no gradients orthogonal to the edge will be considered for variable extrapolations. Figure 3.14 illustrates three different reconstruction variants. 3.14/a shows a simple first-order approach, whereas 3.14/b represents a linear extrapolation of $\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}$ and $\tilde{\mathbf{q}}^{\rightarrow}_{(i,j)}$ on the cell-interface. Values for $\tilde{\mathbf{q}}^{\leftrightarrow}_{(i,j)}$ are, in both cases, computed by an arithmetic averaging. Figure 3.14/c uses a one-sided parabolic approach to compute $\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}$ and $\tilde{\mathbf{q}}^{\rightarrow}_{(i,j)}$. For $\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}$, for example, values for $\mathbf{q}$ and $\nabla\mathbf{q}$ on the from-side, as well as values for $\mathbf{q}$ on the to-side will be used to construct a parabola. This parabola is then used to compute the values on the interface. For this reconstruction formulation the central values $\tilde{\mathbf{q}}^{\leftrightarrow}_{(i,j)}$ are computed by averaging $\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}$ and $\tilde{\mathbf{q}}^{\rightarrow}_{(i,j)}$ instead of averaging the nodal values directly. This approach is able represent a second order polynomial function exactly and thus leads to a formal third order. Due to the lack of transverse influence, however, this extrapolation does not lead to a higher order approximation for multi-dimensional problems. It does, however, improve the solution quality for certain problems (see also chapter 8).
If discontinuous solutions occur, it might become necessary to locally restrict the variable reconstruction to the simple first order approach. This is necessary to ensure the stability of upwind schemes. As an example for such a local reduction to first-order the limiter by Van Albada [4] shall be briefly explained here. Let $\tilde{\mathbf{q}}_{\mathrm{I},(i,j)}$ be a first-order reconstruction and $\tilde{\mathbf{q}}_{\mathrm{II},(i,j)}$ a second-order reconstruction. To compute a limited reconstruction the directional derivatives (in direction of the connecting edge) are needed on both adjacent nodes:

$$\mathbf{q}'_i = \frac{1}{|\mathbf{x}_{\{i,j\}} - \mathbf{x}_i|}(\nabla\mathbf{q})^T_i(\mathbf{x}_{\{\mathbf{i,j}\}} - \mathbf{x}_\mathbf{i}), \quad \mathbf{q}'_{\{i,j\}} = \frac{1}{|\mathbf{x}_{\{i,j\}} - \mathbf{x}_i|}(\nabla\mathbf{q})^T_{\{i,j\}}(\mathbf{x}_{\{\mathbf{i,j}\}} - \mathbf{x}_\mathbf{i}). \quad (3.20)$$

A locally reduced (to first order) reconstruction can now be computed for every component of the variable vector:

$$\tilde{q}_{\mathrm{ltd},(i,j),k} = \left(1 - VA\left(q'_{i,k}, q'_{\{i,j\},k}\right)\right)\tilde{q}_{\mathrm{I},(i,j),k} + VA\left(q'_{i,k}, q'_{\{i,j\},k}\right)\tilde{q}_{\mathrm{II},(i,j),k}. \quad (3.21)$$

3.3 Time Integration

This intends to briefly describe the time integration techniques, used in the scope of this work. Explicit techniques, as well as various convergence acceleration techniques shall be described. The use of multi-grid on unstructured grids can be found in a separate chapter. Mainly equations with only first derivatives of the variables in time have been treated. In chapter 7, however, an example for an equation with second derivatives can be found. Any discretized equation with first derivatives in time can be written in the following form:

$$\left[\int_V \frac{\partial\mathbf{q}}{\partial t}\mathrm{d}V\right]_i = \mathrm{Res}(\mathbf{Q})_i. \quad (3.22)$$

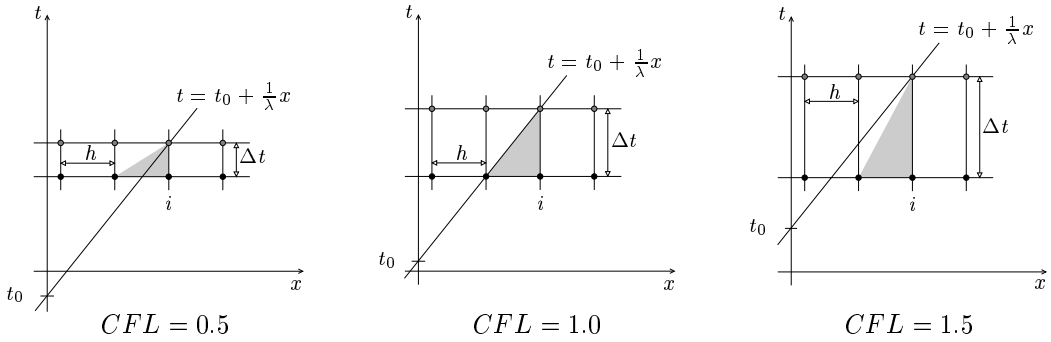

Figure 3.15: Space-time diagrams for different CFL-numbers.

The left part of (3.22) contains the full time discretization, whereas the right hand side describes the spatial discretization plus all possible source terms.

3.3.1 Explicit Time Stepping

Explicit schemes compute a new time level, using only existing values of an old time level. If the upper index n denotes the time level of a computation, $\mathbf{Q}^{n+1}$ can be computed as a function of $\mathbf{Q}^n$ and the discrete time step Δt:

$$\mathbf{Q}^{n+1} = \mathbf{Q}^{n+1}(\mathbf{Q}^n, \Delta t)\,. \tag{3.23}$$

Probably the main disadvantage of explicit time-discretizations is the limited stability. The time-step width Δt is limited, depending on the spatial step-sizes and the eigenvalues of the system to be solved. See the following paragraph about a one step scheme for an example. Explicit schemes do, however, have the significant advantage of their simplicity. Thus the implementation of new physical problems or new spatial discretizations, becomes a lot easier compared to implicit schemes. Another advantage over implicit schemes is the fairly low memory requirement for explicit computations. As memory becomes cheaper, however, this advantage does not play as big a role as it used to.

Simple One Step Scheme

A simple first order accurate approach is the following:

$$\left[\int_V \frac{\partial \mathbf{q}}{\partial t}\mathrm{d}V\right]_i = \frac{(\Delta V)_i}{\Delta t}\left(\mathbf{q}_i^{n+1} - \mathbf{q}_i^n\right) \tag{3.24}$$

Thus the value on a new time level can be computed using this update formula.

$$\mathbf{q}_i^{n+1} = \mathbf{q}_i^n + \frac{\Delta t}{(\Delta V)_i}\mathrm{Res}(\mathbf{q}_i) \tag{3.25}$$

Consider the simple convection problem (3.8), which has been used to introduce the motivation for upwind schemes. The characteristic of (3.8) is:

$$\frac{\partial x}{\partial t} = u \tag{3.26}$$

If the discretization (3.9), is used, the time-step Δt is limited to:

$$\Delta t_{\max} \leq \frac{h}{|u|} \tag{3.27}$$

The ratio between the actual time-step Δt and the upper limit, based on the characteristics of the system to be solved, is called the *Courant* or *CFL (Courant-Friedrichs-Levy)* number. It can be defined as follows:

$$CFL = \frac{\lambda \Delta t}{h} \tag{3.28}$$

with h as the spatial step-size and λ as the characteristic of the equation (u in case of (3.8)). In case of a first order accurate upwind scheme in space and a one step scheme in time, $CFL \leq 1$ has to be fulfilled. One step of such a simple scheme will transport information from one node to its neighbors (depending on the direction of the characteristic), but not any further than this. Figure 3.15 illustrates this behavior. For (3.8) all points on the straight line, that can be seen in figure 3.15, have the same state of $\mathbf{q}$. It can be seen that for $CFL = 0.5$ the characteristic curve $t = t_0 + \frac{1}{\lambda}x$ is well within the numerical range of influence, whereas for $CFL = 1$ it is on the limit of the influence range. For $CFL > 1$ it is outside of this area and thus the scheme becomes unstable. For multi dimensional problems on unstructured grids it is not always easy to fulfill the CFL-condition exactly. In praxis most often an implementation is used, which tries to stay on the safe side. It does locally take into account the maximal characteristic and the minimal edge length.

Runge-Kutta Time Stepping

The Runge-Kutta scheme is a multi-stage method, which has been originally developed to solve ordinary differential equations. It provides a better time-accuracy as well as better stability properties. A new time level $n + 1$ will be reached via different intermediate states.

$$\mathbf{Q}^n = \mathbf{Q}^{n,0} \rightarrow \mathbf{Q}^{n,1} \rightarrow \mathbf{Q}^{n,2} \rightarrow \ldots \mathbf{Q}^{n,N_{rk}} = \mathbf{Q}^{n+1}$$

The maximal time-accuracy of a Runge-Kutta scheme is $O(\Delta t^{N_{\mathrm{rk}}-1})$, with N_{rk} as the number of sub-steps. A variant of the Runge Kutta scheme, which is widely used for CFD problems [5], shall be described here. The following sequence is used to compute a

new time-level.

$$\begin{aligned}
\mathbf{q}_i^{n,1} &= \mathbf{q}_i^{n,0} + \frac{\alpha_1 \Delta t}{(\Delta V)_i} \mathrm{Res}\left(\mathbf{Q}^{n,0}\right) \\
\mathbf{q}_i^{n,2} &= \mathbf{q}_i^{n,0} + \frac{\alpha_2 \Delta t}{(\Delta V)_i} \mathrm{Res}\left(\mathbf{Q}^{n,1}\right) \\
\mathbf{q}_i^{n,3} &= \mathbf{q}_i^{n,0} + \frac{\alpha_3 \Delta t}{(\Delta V)_i} \mathrm{Res}\left(\mathbf{Q}^{n,2}\right) \\
&\vdots \qquad\qquad\qquad \vdots \\
\mathbf{q}^{n+1} \;=\; \mathbf{q}_i^{n,N_{\mathrm{rk}}} &= \mathbf{q}_i^{n,0} + \frac{\alpha_{N_{\mathrm{rk}}} \Delta t}{(\Delta V)_i} \mathrm{Res}\left(\mathbf{Q}^{n,N_{\mathrm{rk}}-1}\right)
\end{aligned} \tag{3.29}$$

$\alpha_1, \alpha_2, \ldots, \alpha_{N_{\mathrm{rk}}}$ are the Runge-Kutta coefficients. A standard three step scheme with the following coefficients has been widely used in this work.

$$\alpha_1 = \frac{1}{4}\,, \quad \alpha_2 = \frac{1}{2}\,, \quad \alpha_3 = 1$$

3.3.2 Convergence Acceleration

If a stationary solution is sought, the time-accuracy becomes completely unimportant. Various techniques exist to accelerate the convergence of a time-dependent solver towards a steady solution. The most important, probably, is the multi-grid method. As already stated above this will find mention in another chapter. There are, however, a few fairly simple but efficient methods to accelerate a solution process.

Local Time Steps

Explicit methods have severe restrictions on the size of a time-step Δt. These depend on the characteristics of the problem, as well as on the local step-size of the mesh. For an unsteady simulation, a globally minimal Δt has to be used, in order to keep the scheme consistent in time. Certain regions of a domain, however, might impose stricter restrictions than others. In case of steady state problems it proved to be very efficient to use the maximal possible Δt, based on a local stability restriction. Thus the time-step will not be constant in space, which results in a complete loss of the time accuracy.

Residual Smoothing

To improve the stability of (3.25) or (3.29), it is possible to smooth the discrete residual $\mathrm{Res}(\mathbf{Q})$. The following discrete equation can be used to compute a smoothed residual:

$$\overline{\mathbf{r}}_i - \epsilon \left[\nabla^2 \overline{\mathbf{r}}\right]_i = \mathbf{r}_i \quad , \quad \mathbf{r}_i \;=\; \mathrm{Res}(\mathbf{Q})_i\,. \tag{3.30}$$

The Laplace operator shall be discretized in a simplified manner:

$$\left[\nabla^2 \overline{\mathbf{r}}\right]_i = \left(\sum_{j=1}^{N_{\mathrm{ngh},i}} \overline{\mathbf{r}}_{\{i,j\}} \right) - N_{\mathrm{ngh},i} \cdot \overline{\mathbf{r}}_i \tag{3.31}$$

A solution for the residual-smoothing equation can be found using a point-iterative method (e.g. Gauss-Seidel, Jacobi). With (3.30) and (3.31) the following update-formula can be obtained (k indicates the iteration level):

$$\overline{\mathbf{r}}_i^{k+1} = \frac{1}{1 - \epsilon N_{\mathrm{ngh},i}} \cdot \left(\mathbf{r}_i + \epsilon \left(\sum_{j=1}^{N_{\mathrm{ngh},i}} \overline{\mathbf{r}}_{\{i,j\}}^{k} \right) \right) . \tag{3.32}$$

Usually this iterative procedure will be executed for a fixed number of steps, instead of iterating it until a certain level of convergence has been reached. The smoothed residual is then used to perform an explicit time-step. Together with a multi-grid scheme, it proved to be very useful.

Pre-Conditioning

Consider the following system of N_{eq} partial differential equations:

$$\frac{\partial \mathbf{q}}{\partial t} + \mathbf{A} \frac{\partial \mathbf{q}}{\partial x} = 0 . \tag{3.33}$$

If $\mathbf{A}$ has the eigenvalues $\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_{N_{\mathrm{eq}}}$, the time-step will be limited due to the maximal absolute value. If the condition number

$$\mathrm{cond}(\mathbf{A}) = \frac{|\lambda|_{\max}}{|\lambda|_{\min}} \qquad \text{with} \quad \begin{cases} |\lambda|_{\max} = \max(|\lambda_1|, |\lambda_2|, |\lambda_3|, \ldots, |\lambda_{N_{\mathrm{eq}}}|) \\ |\lambda|_{\min} = \min(|\lambda_1|, |\lambda_2|, |\lambda_3|, \ldots, |\lambda_{N_{\mathrm{eq}}}|) \end{cases} . \tag{3.34}$$

becomes very large, the different waves will travel with very different speeds. A good example for such a system are the Euler equations at very low Mach-numbers. The system of the one-dimensional Euler-equations, written in a non-conservative form, will lead to the following matrix:

$$\mathbf{A} = \begin{pmatrix} u & \rho & 0 \\ 0 & u & \rho^{-1} \\ 0 & \rho a^2 & u \end{pmatrix} , \quad \text{with } \mathbf{q} = \begin{pmatrix} \rho \\ u \\ p \end{pmatrix} \tag{3.35}$$

with u as the local convection-speed, ρ as the density and a as the speed of sound. The eigenvalues of (3.35) are the following:

$$\lambda_1 = u, \quad \lambda_2 = u + a, \quad \lambda_3 = u - a \tag{3.36}$$

For low Mach numbers the sound waves (λ_2, λ_3), can be several orders of magnitude faster than the convection. Thus the time-step of an explicit scheme will be limited due to the sound waves. The time, required to come to a steady solution, strongly depends on the convection as well. Thus the system of the one-dimensional Euler equations becomes very stiff for low Mach numbers. This leads to extremely long computational times for low Mach number computations. One way to overcome this problem will be briefly described now. It is only intended to show the principle. For a detailed description of

pre-conditioning methods for low Mach-number compressible flow, please refer to [6]. Assume a matrix $\mathbf{P}$ exists, so that the product $\mathbf{P} \cdot \mathbf{A}$ has a better condition number than $\mathbf{A}$. If now the vector of the time derivatives will be multiplied by $\mathbf{P}^{-1}$ the following equation can be obtained.

$$\mathbf{P}^{-1} \frac{\partial \mathbf{q}}{\partial t} + \quad \mathbf{A} \frac{\partial \mathbf{q}}{\partial x} \;=\; 0 \tag{3.37}$$

$$\Leftrightarrow \qquad \frac{\partial \mathbf{q}}{\partial t} + \mathbf{P}\,\mathbf{A} \frac{\partial \mathbf{q}}{\partial x} \;=\; 0 \tag{3.38}$$

If a steady state solution is sought, solving (3.37) or (3.38) respectively, would mean that a solution of (3.33) has been found. Please note that intermediate states do not represent real physical solutions. A scheme, which iterates on (3.38) is not time accurate and thus not applicable to time-dependent problems. An example for a pre-conditioning matrix for (3.35) is the following simple matrix:

$$\mathbf{P} = \begin{pmatrix} \frac{1}{M_\infty} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad , \quad M_\infty = \frac{u_\infty}{a_\infty} \,. \tag{3.39}$$

The eigenvalues of (3.39) are:

$$\lambda_1 = \frac{a_\infty u}{u_\infty}, \quad \lambda_2 = u + a, \quad \lambda_3 = u - a \,. \tag{3.40}$$

For the free-stream state $(\rho_\infty, u_\infty, p_\infty)$ the condition number becomes:

$$\mathrm{cond}(\mathbf{P}) = \frac{|u_\infty + a_\infty|}{|u_\infty - a_\infty|} = \frac{1 + M_\infty}{1 - M_\infty} \,, \tag{3.41}$$

and it tends to one, which is the optimum, if the Mach-number tends to zero. An efficient pre-conditioning matrix for the system of the multi-dimensional Euler-equations, written in conservative variables, has been proposed by Turkel [7]. The pre-conditioning which has been used in the scope of this work is similar to the one proposed by Turkel. See [6] for a detailed description of this pre-conditioner.

3.3.3 Generic Numerical Algorithms

Many numerical aspects, which have been mentioned so far, are common to a variety of problems and equations. The main objective of this work has been to provide a flexible toolbox for a variety of different applications. Hence the implementation of new physical problems should be as fast and as easy as possible, without having to worry about too many details. In order to achieve this, many numerical features and aspects can be moved into a general library kernel, instead of having to code this for every new problem. At present the following topics are treated in the general kernel:

- mesh handling (i.e. construction of control volumes and interface vectors)

- variable reconstruction for central, upwind or mixed discretizations
- limited variable reconstruction for discontinuous problems
- gradient computation using least squares, divergence theorem or a combination of both
- various time advancing schemes

3.4 Flux Formulations for Compressible Flow

The flux formulations, which have been used for compressible flow simulations, shall be briefly described here. Both (viscous and inviscid) parts of the *Navier-Stokes* flux are treated, using a finite volume discretization. Opposed to this, some research groups use a mixed finite volume/finite element discretization, where the viscous part of the equations will be treated, using finite elements [8]. This is mainly done because of the inaccurate discretizations of second order derivatives on anisotropic triangles or tetrahedra [1]. In the scope of this work hybrid grids have been used to overcome this problem.

3.4.1 Inviscid Fluxes

For the inviscid part of the *Navier-Stokes Equations* a modified version of Roe's flux difference splitting has been used. This scheme has been introduced by Viozat et. al. in 1996 [6] and is referred to as the Roe-Turkel scheme, since it uses ideas of Turkel's pre-conditioning method for Low Mach number flow. The original Roe scheme and the modification shall be very briefly outlined in the following two paragraphs.

The Roe Flux Difference Splitting

A very popular upwind scheme for the compressible Euler equations has been proposed by Roe in 1981 [9]. It is based on the idea from (3.10). Other than the simple system of scalar convection equations (3.12), which has only one characteristic, the one dimensional Euler equations have three different characteristics. These characteristics may have different signs and they cannot be assigned to one equation each. In one dimension the Euler equations read:

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}(\mathbf{q})}{\partial x} = 0 \quad , \quad \mathbf{q} = \begin{pmatrix} \rho \\ \rho u \\ \rho E \end{pmatrix} , \mathbf{f}(\mathbf{q}) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ u(\rho E + p) \end{pmatrix} . \tag{3.42}$$

Using the Jacobi matrix $\partial \mathbf{f}/\partial \mathbf{q}$ (3.42) can be written as:

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial x} = 0 \, . \tag{3.43}$$

$\partial \mathbf{f}/\partial \mathbf{q}$ can be seen analogly to the convection speed u from (3.8) and it has the eigenvalues $\lambda_1 = u, \lambda_2 = u + a, \lambda_3 = u - a$, with a as the speed of sound. The central difference

from (3.10) can be easily discretized. A problem will be faced, however, if the upwind damping term has to be formulated. To formulate this, (3.43) can be transformed into its characteristic form. Using the matrix of the eigenvectors $\mathbf{R}$ the characteristic form is:

$$\frac{\partial \mathbf{W}}{\partial t} + \mathbf{\Lambda}\frac{\partial \mathbf{W}}{\partial x} = 0 \quad , \quad \mathrm{d}\mathbf{W} = \mathbf{R}^{-1}\,\mathrm{d}\mathbf{q} \quad , \quad \mathbf{\Lambda} = \begin{pmatrix} u & 0 & 0 \\ 0 & u+a & 0 \\ 0 & 0 & u-a \end{pmatrix} \tag{3.44}$$

The damping term can be formulated, using this characteristic form, and then be transformed back. Hence it becomes:

$$\frac{h}{2}|\mathbf{\Lambda}|\left(\frac{\mathbf{W}_{i+1} - 2\,\mathbf{W}_i + \mathbf{W}_{i-1}}{h^2}\right) \quad , \quad |\mathbf{\Lambda}| = \begin{pmatrix} |u| & 0 & 0 \\ 0 & |u+a| & 0 \\ 0 & 0 & |u-a| \end{pmatrix} . \tag{3.45}$$

If this has to be formulated on an unstructured grid, using the finite volume approach, the Jacobi matrix of the complete inviscid flux vector (see (2.18)) on a cell interface $\mathbf{n}\Delta A$ has to be computed. This has to include the geometry as well. For a three dimensional problem the following flux can be formulated in Cartesian coordinates.

$$\mathbf{H}(\mathbf{q})\mathbf{n}\Delta A = \begin{pmatrix} \rho u \cdot n_x & + & \rho v \cdot n_y & + & \rho w \cdot n_z \\ (\rho u^2 + p) \cdot n_x & + & \rho uv \cdot n_y & + & \rho uw \cdot n_z \\ \rho vu \cdot n_x & + & (\rho v^2 + p) \cdot n_y & + & \rho vw \cdot n_z \\ \rho uv \cdot n_x & + & \rho vw \cdot n_y & + & (\rho v^2 + p) \cdot n_z \\ u(\rho E + p) \cdot n_x & + & u(\rho E + p) \cdot n_y & + & u(\rho E + p) \cdot n_z \end{pmatrix} \Delta A\,. \tag{3.46}$$

The Roe-flux on the interface i, j becomes the following:

$$[\mathbf{H}\,\mathbf{n}\,\Delta A]_{(i,j)} = \frac{1}{2}\,\mathbf{H}(\tilde{\mathbf{q}}^{\leftrightarrow}_{(i,j)})\,\mathbf{n}\,(\Delta A)_{(i,j)} - \frac{1}{2}\,|\mathbf{A}|(\bar{\mathbf{q}})\,(\tilde{\mathbf{q}}^{\rightarrow}_{(i,j)} - \tilde{\mathbf{q}}^{\leftarrow}_{(i,j)})\,. \tag{3.47}$$

The matrix $|\mathbf{A}|$ is called Roe-matrix and the values $\bar{\mathbf{q}}$ Roe averages. The Roe averages represent central values on the interface. For details on how to compute the Roe averages, please refer to [9]. The Roe-matrix can be computed using the absolute values of the eigenvalues from the Jacobi matrix:

$$|\mathbf{A}| = \mathbf{R}\,|\mathbf{\Lambda}|\,\mathbf{R}^{-1}, \quad |\mathbf{\Lambda}|\left(\frac{\partial\,[\mathbf{H}\,\mathbf{n}\,\Delta A]_{(i,j)}}{\partial \mathbf{q}}\right) = \begin{pmatrix} |\lambda_1| & 0 & 0 & 0 & 0 \\ 0 & |\lambda_2| & 0 & 0 & 0 \\ 0 & 0 & |\lambda_3| & 0 & 0 \\ 0 & 0 & 0 & |\lambda_4| & 0 \\ 0 & 0 & 0 & 0 & |\lambda_5| \end{pmatrix} . \tag{3.48}$$

$\mathbf{R}$ represents the corresponding matrix of the eigenvectors. The Roe-averages are computed in order to fulfill the following condition:

$$\frac{\partial\,[\mathbf{H}\,\mathbf{n}\,\Delta A]_{(i,j)}}{\partial \mathbf{q}}\,(\tilde{\mathbf{q}}^{\rightarrow}_{(i,j)} - \tilde{\mathbf{q}}^{\leftarrow}_{(i,j)}) = \mathbf{H}(\tilde{\mathbf{q}}^{\rightarrow}_{(i,j)})\,(\mathbf{n}\,\Delta A)_{(i,j)} - \mathbf{H}(\tilde{\mathbf{q}}^{\leftarrow}_{(i,j)})\,(\mathbf{n}\,\Delta A)_{(i,j)} \tag{3.49}$$

Modification of Roe's Scheme for Low Mach Numbers (Roe-Turkel)

The Roe scheme, together with a pre-conditioner as proposed by Turkel, can principally be used for low Mach-number computations. It can, however, be observed that the artificial damping term becomes excessively large if the Mach number tends to zero. In 1997 it has been shown by Viozat [6], that the truncation error of the Roe scheme is of the order $O(h/M)$. h denoted the spatial step-size and M the Mach-number. To overcome this problem, Viozat et al. proposed to modify the stabilization term of the Roe scheme. To briefly illustrate the idea, the one dimensional Euler equations in differential form shall be considered:

$$\frac{\partial \mathbf{q}}{\partial t} + \mathbf{A}\frac{\partial \mathbf{q}}{\partial x} = 0\,. \tag{3.50}$$

$\mathbf{A}$ represents the Jacobi matrix of the Euler flux. A finite difference discretization can be written as:

$$\frac{\mathbf{q}_i^{n+1} - \mathbf{q}_i^n}{\Delta t} + \mathbf{A}\left(\frac{\mathbf{q}_{i+1}^n - \mathbf{q}_{i-1}^n}{2h}\right) - \frac{h}{2}|\mathbf{A}|\left(\frac{\mathbf{q}_{i+1}^n - 2\mathbf{q}_i^n + \mathbf{q}_{i-1}^n}{h^2}\right) = 0\,. \tag{3.51}$$

$|\mathbf{A}|$ represents the Roe matrix. If (3.51) shall be pre-conditioned to improve the convergence it becomes:

$$\frac{\mathbf{q}_i^{n+1} - \mathbf{q}_i^n}{\Delta t} + \mathbf{PA}\left(\frac{\mathbf{q}_{i+1}^n - \mathbf{q}_{i-1}^n}{2h}\right) - \frac{h}{2}|\mathbf{PA}|\left(\frac{\mathbf{q}_{i+1}^n - 2\mathbf{q}_i^n + \mathbf{q}_{i-1}^n}{h^2}\right) = 0\,, \tag{3.52}$$

with $\mathbf{P}$ as the pre-conditioning matrix. (3.52) can be multiplied by $\mathbf{P}^{-1}$ to obtain the following form:

$$\mathbf{P}^{-1}\frac{\mathbf{q}_i^n - \mathbf{q}_i^{n+1}}{\Delta t} + \mathbf{A}\left(\frac{\mathbf{q}_{i+1}^n - \mathbf{q}_{i-1}^n}{2h}\right) - \frac{h}{2}\mathbf{P}^{-1}|\mathbf{PA}|\left(\frac{\mathbf{q}_{i+1}^n - 2\mathbf{q}_i^n + \mathbf{q}_{i-1}^n}{h^2}\right) = 0\,. \tag{3.53}$$

Viozat showed, that the truncation error can be improved to be of the order $O(h)$, if $\mathbf{P}^{-1}|\mathbf{PA}|$ is used as stabilization term, instead of $|\mathbf{A}|$. For a detailed description of the Roe-Turkel scheme and the corresponding consistency proofs for low Mach-number flow, please see [6].

3.4.2 Viscous Fluxes

In order to compute the viscous flux vector (3.16) the spatial derivatives of the primitive variables $\mathbf{v}$ and T are needed. As stated above, the derivatives of the conservative variables are directly available from the library kernel. The vector of conservative variables $\mathbf{q}$ and the velocity vector $\mathbf{v}$ are given by:

$$\mathbf{q} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{pmatrix} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix} \quad , \quad \mathbf{v} = \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \frac{1}{q_1}\begin{pmatrix} q_2 \\ q_3 \\ q_4 \end{pmatrix}\,. \tag{3.54}$$

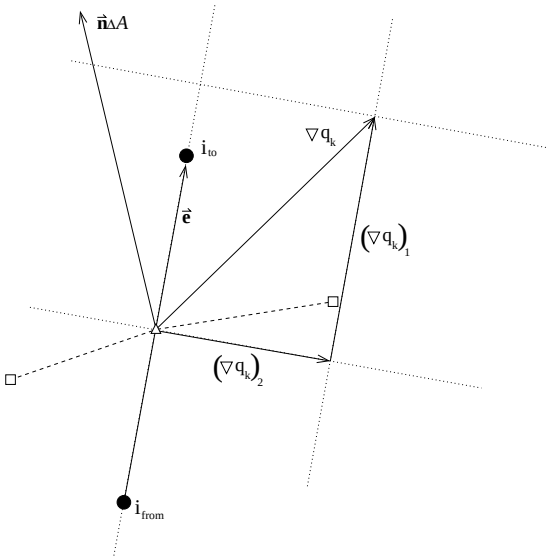

Figure 3.16: Derivatives for viscous fluxes.

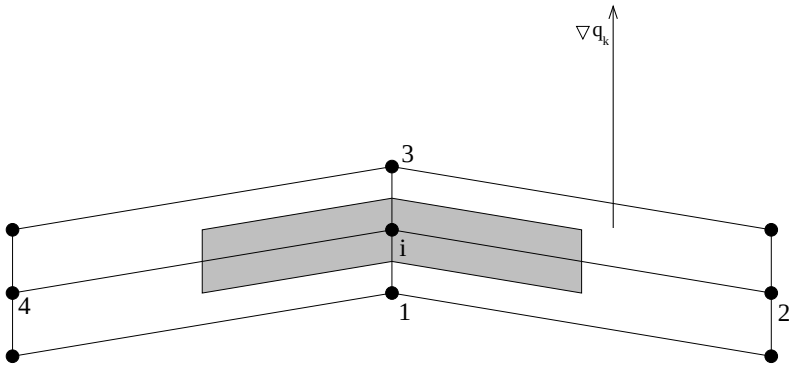

Figure 3.17: Erroneous gradient for viscous fluxes.

Using the available gradients for the conservative variables the following expression can be derived for the velocity gradient:

$$\nabla \mathbf{v} = \begin{pmatrix} \frac{1}{q_1}\frac{\partial q_2}{\partial x} - \frac{q_2}{q_1^2}\frac{\partial q_1}{\partial x} & \frac{1}{q_1}\frac{\partial q_2}{\partial y} - \frac{q_2}{q_1^2}\frac{\partial q_1}{\partial y} & \frac{1}{q_1}\frac{\partial q_2}{\partial z} - \frac{q_2}{q_1^2}\frac{\partial q_1}{\partial z} \\ \frac{1}{q_1}\frac{\partial q_3}{\partial x} - \frac{q_3}{q_1^2}\frac{\partial q_1}{\partial x} & \frac{1}{q_1}\frac{\partial q_3}{\partial y} - \frac{q_3}{q_1^2}\frac{\partial q_1}{\partial y} & \frac{1}{q_1}\frac{\partial q_3}{\partial z} - \frac{q_3}{q_1^2}\frac{\partial q_1}{\partial z} \\ \frac{1}{q_1}\frac{\partial q_4}{\partial x} - \frac{q_4}{q_1^2}\frac{\partial q_1}{\partial x} & \frac{1}{q_1}\frac{\partial q_4}{\partial y} - \frac{q_4}{q_1^2}\frac{\partial q_1}{\partial y} & \frac{1}{q_1}\frac{\partial q_4}{\partial z} - \frac{q_4}{q_1^2}\frac{\partial q_1}{\partial z} \end{pmatrix} . \tag{3.55}$$

The temperature gradient can be computed as follows:

$$\nabla T = \frac{\gamma - 1}{q_1 R} \begin{pmatrix} \frac{\partial q_5}{\partial x} - \frac{q_5}{q_1}\frac{\partial q_1}{\partial x} + \frac{q_2^2}{q_1^2}\frac{\partial q_1}{\partial x} - \frac{q_2}{q_1}\frac{\partial q_2}{\partial x} + \frac{q_3^2}{q_1^2}\frac{\partial q_1}{\partial x} - \frac{q_3}{q_1}\frac{\partial q_3}{\partial x} + \frac{q_4^2}{q_1^2}\frac{\partial q_1}{\partial x} - \frac{q_4}{q_1}\frac{\partial q_4}{\partial x} \\ \frac{\partial q_5}{\partial y} - \frac{q_5}{q_1}\frac{\partial q_1}{\partial y} + \frac{q_2^2}{q_1^2}\frac{\partial q_1}{\partial y} - \frac{q_2}{q_1}\frac{\partial q_2}{\partial y} + \frac{q_3^2}{q_1^2}\frac{\partial q_1}{\partial y} - \frac{q_3}{q_1}\frac{\partial q_3}{\partial y} + \frac{q_4^2}{q_1^2}\frac{\partial q_1}{\partial y} - \frac{q_4}{q_1}\frac{\partial q_4}{\partial y} \\ \frac{\partial q_5}{\partial z} - \frac{q_5}{q_1}\frac{\partial q_1}{\partial z} + \frac{q_2^2}{q_1^2}\frac{\partial q_1}{\partial z} - \frac{q_2}{q_1}\frac{\partial q_2}{\partial z} + \frac{q_3^2}{q_1^2}\frac{\partial q_1}{\partial z} - \frac{q_3}{q_1}\frac{\partial q_3}{\partial z} + \frac{q_4^2}{q_1^2}\frac{\partial q_1}{\partial z} - \frac{q_4}{q_1}\frac{\partial q_4}{\partial z} \end{pmatrix} . \tag{3.56}$$

Two different approaches have been used to compute the viscous fluxes.

Direct Use of Available Gradients

The simplest approach to compute a viscous flux along an edge, as it can be seen in figure 3.16, is the following. On the nodes i and $\{i, j\}$ values for $\nabla \mathbf{q}$ are available. Using their arithmetic mean value on the edge, it is possible to compute the primitive gradients, using (3.56) and (3.55). There is one major drawback, however. Consider a nodal arrangement, as it can be seen in figure 3.17. Furthermore assume a gradient, that is perfectly aligned with the edges $i \leftrightarrow 1$ and $i \leftrightarrow 3$. This would be nearly true in case of a boundary layer for example. Dependent on the aspect ratio and the curvature of the mesh, however, the computed gradient will contain a strong error. This error is caused by the contribution of the nodes 2 and 4. It will only decrease, if the aspect ratio will be reduced. In case of anisotropic effects (e.g. boundary or shear-layers) this cannot be done, without increasing the amount of nodes dramatically.

Use of the Directional Derivatives Along an Edge

A way to overcome the problem, described in the previous paragraph, is to use the directional derivative along the edge in question. In figure 3.17 the interface between i and 3 is perfectly orthogonal to the gradient vector ∇q_k. The dominant viscous fluxes are caused by tangential forces along the interface, which are determined by velocity gradients perpendicular to it. A robust way of computing central gradients on an edge (figure 3.16) is the following:

1. The arithmetic average of the standard gradient is computed

$$(\nabla \mathbf{q})_{i,j} = \frac{1}{2}\left((\nabla \mathbf{q})_i + (\nabla \mathbf{q})_{\{i,j\}}\right) .$$

2. The transverse gradient is computed by subtracting the directional portion of the standard gradient:

$$(\nabla \mathbf{q})_{\mathrm{I,(i,j)}} = (\nabla \mathbf{q})_{(i,j)} - \frac{1}{|\mathbf{x}_{\{i,j\}} - \mathbf{x}_i|} (\nabla \mathbf{q})^T_{(i,j)} (\mathbf{x}_{\{i,j\}} - \mathbf{x}_i) .$$

3. By differencing the values on i_{from} and i_{to} the directional derivative along the edge can be computed:

$$(\nabla \mathbf{q})_{\mathrm{II,(i,j)}} = \frac{1}{|\mathbf{x}_{\{i,j\}} - \mathbf{x}_i|} (\mathbf{q}_{\{i,j\}} - \mathbf{q}_i) .$$

4. The directional derivative is added to the transverse one and this gradient will then be used to compute the viscous fluxes.

The corrected gradient on the edge $i \leftrightarrow \{i,j\}$ can be expressed by the following formula:

$$(\nabla \mathbf{q})_{\mathrm{corr,(i,j)}} = (\nabla \mathbf{q})_{(i,j)} - \frac{1}{|\mathbf{x}_{\{i,j\}} - \mathbf{x}_i|} \left((\nabla \mathbf{q})^T_{(i,j)} (\mathbf{x}_{\{i,j\}} - \mathbf{x}_i) - \mathbf{q}_{\{i,j\}} + \mathbf{q}_i \right) . \quad (3.57)$$

$(\nabla \mathbf{q})_{(i,j)}$ is a simple average value of the nodal gradients, as they will be usually provided by the library:

$$(\nabla \mathbf{q})_{(i,j)} = \frac{1}{2} \left((\nabla \mathbf{q})_i + (\nabla \mathbf{q})_{\{i,j\}} \right) . \quad (3.58)$$