# Chapter 4

# Mesh Generation

This chapter describes the efforts which have been made to develop a flexible meshing algorithm. As it has been described in chapter 3, a multi-element unstructured grid-type has been chosen to offer the highest flexibility possible. The first part of this chapter will give a brief introduction into the generation of purely triangular grids. This part of the meshing process has been developed by R. Vilsmeier (see [1] and [3]). See also [10] for details on triangular mesh generation. Then a new way of describing boundaries with the help of scalar field functions will be introduced. This method offers the possibility of keeping large regions Cartesian and thus it benefits the accuracy as well as the computational efficiency. A typical two dimensional triangular grid has six adjacent edges per node, opposed to four in case of a Cartesian grid. For a typical CFD application the dominant factor, in terms of computational costs, is the evaluation of the flux integral. Thus the usage of a Cartesian mesh will roughly save 30% CPU-time. In three dimensions the effect would be even more dramatic. Level-sets as boundary description offer valuable information, which can be used to create anisotropic layers. Such layers would be needed to resolve boundary layers or other viscous effects. Unfortunately the new meshing strategy has not been used for three dimensional problems so far.

## 4.1 Generation of Unstructured Simplex Grids

If a geometry is given as a set of closed polygons, describing its boundary, the following strategy can be used to generate an unstructured grid.

### 4.1.1 Initial Triangulation

An initial triangulation can be computed, using the rising bubble algorithm (see [1]). This algorithm provides a first triangulation, based on the Delaunay criterion. Figure 4.1 shows an example of such a first triangulation. Computing this first triangulation is a non-trivial process and especially in three dimensions this part of the meshing process can cause severe problems. The level-set based approach, which will be described later in this chapter, offers a way of overcoming this initial triangulation problem.
Another meshing strategy exists, which is called marching front method. This method

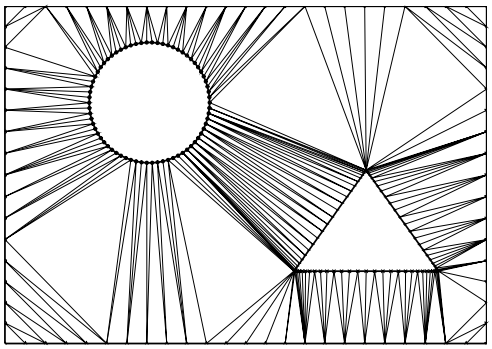

Figure 4.1: Initial triangulation.

starts from a closed boundary and will generate the mesh into the computational domain. This mesh represents the final grid and not an initial stage for an optimization process. It has, however, similar problems, especially if the geometry consists of several complex shapes.

### 4.1.2 Mesh Optimization

Based on an initial triangulation the mesh can be optimized. This process is easier to control and more reliable than the initial triangulation. The principle optimization algorithm, described in this paragraph, is the same that will be used for triangular regions within hybrid grids, later on. The following local operations are used to enhance the grid.

#### Adding New Nodes

Certain regions of a computational domain require a special mesh resolution. This can be described, using the mesh density, which represents the inverse of the desired length of the edges. Based on this mesh density it can be decided if nodes have to be locally inserted or not. Different strategies exist to do this. Figures 4.2 and 4.3 illustrate two different methods. It is important that a certain threshold is used for the insertion criterion in order to avoid oscillating mesh configurations. The mesh density can either be given for discrete regions or it can be computed using an elliptical iteration, based on the mesh density values given on the boundary. This equation has to be continously solved while the new mesh develops (see [1]).

#### Deleting Nodes

Similar to the insertion of new nodes it might become necessary to remove nodes. Due to the application of mesh-smoothing it is possible that regions with a higher than the desired mesh-density exist. Figure 4.6 shows how an existing node can be eliminated via the elimination of an edge. What has been said about the threshold for node insertion applies here as well.

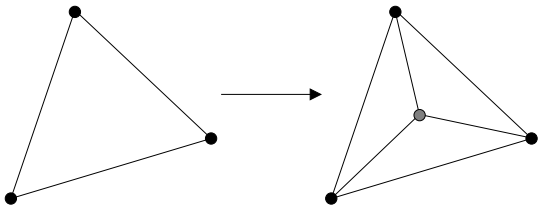

Figure 4.2: Inserting a node.

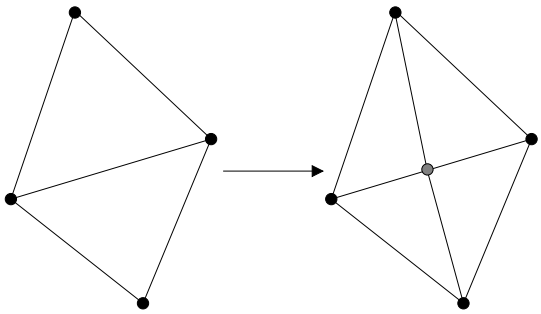

Figure 4.3: Splitting an edge.

### Local Reconfiguration (Edge-Swap)

In order to get an isotropic and undistorted triangulation, the Delaunay criterions has to be fulfilled. For a triangulation that fulfills this criterion, the circumcircle of every triangle is an empty circle. This means it does not contain any other nodes. In order to fulfill the Delaunay criterion a local reconfiguration might become necessary. Figure 4.5 illustrates this process. A pair of triangles will be investigated and if the Delaunay criterion is not fulfilled for these two triangles, the common edge will be exchanged. In figure 4.5 the node D is within the circumcircle of triangle ABC and B in the circumcircle of triangle CDA respectively. Thus the criterion is not fulfilled and the edge AC will be exchanged to BD. If used in an iterative process, this local reconfiguration will lead to a globally fulfilled Delaunay criterion.

### Smoothing

To improve the overall mesh quality local smoothing has to be applied. The position of a node is altered, based on the position of the neighboring nodes. A very simple approach, which has been used in the scope of this work is the following:

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \omega \sum_{j=1}^{N_{\mathrm{ngh},i}} \left( \mathbf{x}_{\{i,j\}}^k - \mathbf{x}_i^k \right) , \tag{4.1}$$

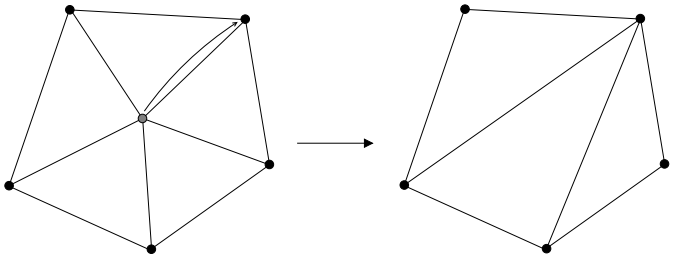

Figure 4.4: Deleting a node.

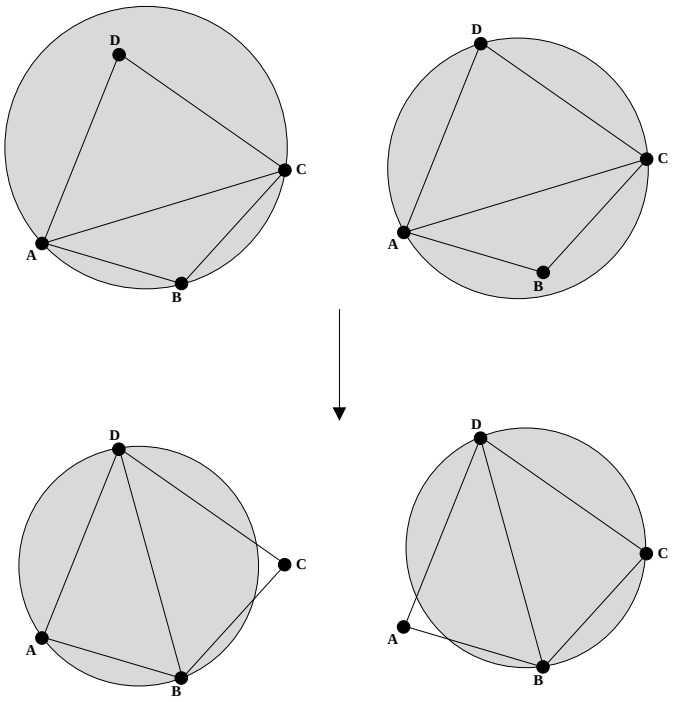


Figure 4.5: Local Delaunay criterion.

with $\omega$ as a relaxation factor. Initially $\omega = 1$ is used. If the new position is outside of the bounding polygon $\mathbf{x}^k_{\{i,1\}}, \mathbf{x}^k_{\{i,2\}}, \ldots, \mathbf{x}^k_{\{i,N_{\mathrm{ngh},i}\}}$ an irregular mesh configuration would be created. In this case $\omega$ has to be reduced and the final position can only be reached after further smoothing steps of the neighboring nodes. It might also be necessary to apply edge swapping in the vicinity of this node. Practical tests, however, show that for most smoothing operations $\omega = 1$ works well. Another approach, which has been successfully used by R. Vilsmeier [1] is to minimize a quality-function, which is based on the ratio between triangle areas and corresponding circumcircle areas. This approach, based on the circumcircles, proved to create grids with a very good quality, but it is computationally more expansive than the simple approach which has been described above. The simple approach has the advantage, that it is principally possible to apply it to hybrid grids as well. Practical experience showes, however, that more sophisticated strategies are necessary in order to ensure a proper alignment of quadrangular elements.

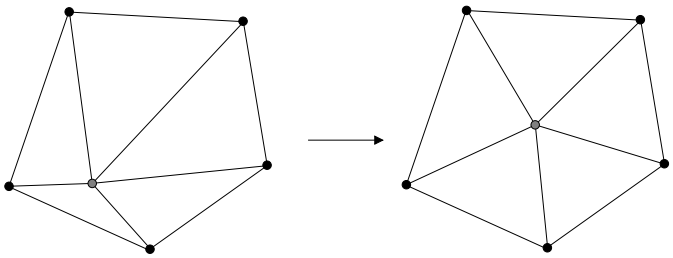

Figure 4.6: Local mesh smoothing.

## 4.2 Mesh Generation Based upon Level-Sets

As already stated in the introduction to this chapter, an alternate way of describing geometries as input for mesh-generation has been applied. The boundaries of a computational domain are described by iso-lines of scalar field functions, commonly called level-sets. In the past years, level set methods gained substantial attention to describe the location and motion of advancing interfaces or discontinuous solutions (e.g.. shock waves in compressible gas dynamics). The basic idea is to describe the position of a front by a distinct iso-value of a scalar function. This scalar function can be stored on a given, fixed mesh. Opposed to some other methods for discontinuous solutions, level-sets avoid the re-meshing process and thus offer a fairly simple algorithm. Level set methods as a computational tool have been introduced by Osher and Sethian in 1988 [11]. Further numerical variants have been developed by Mulder et al. [12] and Sussman et al. [13]. They have been applied to arbitrary problems in fluid dynamics. In [14] an application of level sets for a marching-front like, structured mesh generator can be found. Other methods to describe fronts on fixed, non-moving grids, exist as well. One example is the volume of fluid or VOF method (see [15]).

### 4.2.1 Non Body-Conformal versus Body-Conformal Grids

Two distinctly different boundary descriptions for numerical simulations shall be briefly discussed in this paragraph.

#### Non Body-Conformal Grids

A surface of an object could also be considered a front, seperating the fluid from the solid material. Hence the boundary could be described, using similar fixed-grid techniques as it has been outlined above. It is thus possible to run computations with complex and curved boundaries on a Cartesian grid, that does not conform to the surface. Boundary conditions would have to be realized by interpolation or other techniques to describe them in between two discrete vertices. This concept is not new and has various drawbacks. See [16], or [17] for examples that date back to 1966. Anisotropic refinement, however, normal to the surface is very difficult and even impossible for many cases. Singularities in the geometry (corners in two dimension) need local mesh refinement to be accurately described. This has to be done, even if the computational problem does not require refinement at those points. Hence it will lead to large numbers of nodes and thus increase runtimes for simulations. Figure 4.7 shows how a corner looks, when represented by differently fine resolved meshes. The obvious advantage of this method is its simple manner of describing complex boundaries. Furthermore it is able to deal with moving and changing shapes.

#### Body-Conformal Grids

The most common way of describing boundaries is to align the computational grid with the domain borders. Thus the realization of boundary conditions becomes fairly easy, compared to the approach described in the paragraph before. This strategy, however,

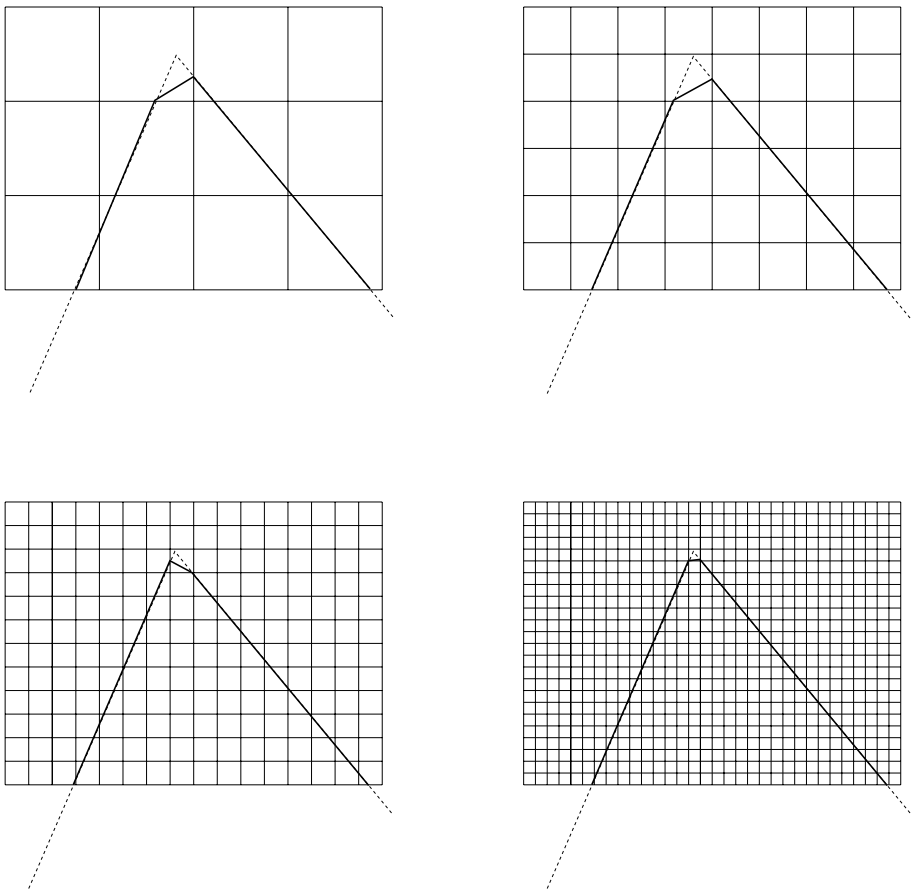

Figure 4.7: Resoultion of corners.

requires far more efforts for mesh generation. Traditionally, mesh generators take a parameterized description of the boundaries as input. In two dimensions this could be a set of closed polygons. Then the space in between the boundaries has to be filled with a mesh. Large parts of the mesh are, however, isotropic in most of the cases. This means that a mesh generator will usually spend a long time in regions where a simple Cartesian grid would just be perfect.

### Combining both Approaches

The meshing algorithm presented here tries to combine the advantages of boundaries described on a fixed grid and body conformal grids. The starting point is a uniform Cartesian grid. Shapes, roughly representing the surfaces of objects to be meshed, are cut out in the vicinity of the surface. The resulting mesh will be body conformal, after several optimization iterations. Unstructured techniques are used to fill the space between the surfaces and the Cartesian base grid. A point-moving algorithm, similar to classical unstructured mesh smoothers, is used to push the later boundary points towards their final positions on the surfaces.

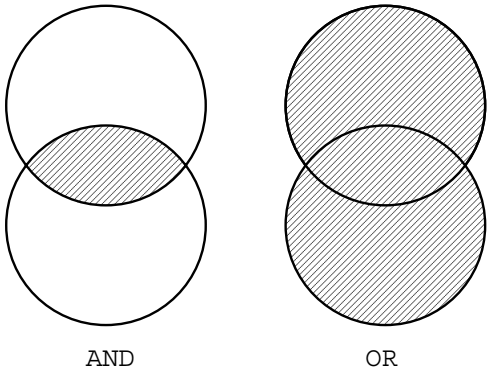


Figure 4.8: Boolean combination of level-sets.

### 4.2.2 Geometry Description

Boundaries will be represented by one or more scalar functions. These functions are referred as $g$-functions ($g(x, y)$). By using scalar functions, geometric effects can be described, which are one dimension below the basic dimension of the problem. This is done by referring to a distinct function-value. Doing so, iso-lines (or iso-surfaces in three dimensions) are obtained. To describe anything with even less dimensions (e.g. corners) it is necessary to use more than one level set or $g$-function. Intersections between different shapes are singularities in the boundary and thus are one dimension below the original boundary description. For two dimensional problems only one kind of such singularities is possible. Geometrical corners in two dimensions are represented by two intersecting $g$-functions. If the method is extended to three dimensions, two different singularities are possible. Edges of three dimensional shapes can be described by the intersection of two $g$-functions, just like corners in two dimensions. Three dimensional corners will be represented by the intersection of a geometric edge with another level set.
The actual shape of a geometrical object consists of faces and corners, represented by several bounding level-sets. Boolean operators are required to declare what is inside and what outside. Figure 4.8 illustrates how two circular shapes could be combined.
In this simple example, the corresponding $g$-function for a circular shape can be given in analytical form, therefore no discrete level-set representation is required:

$$g(\mathbf{x}) \;=\; r \;-\; \sqrt{(\mathbf{x} - \mathbf{x}_\mathrm{m})^2} \tag{4.2}$$

With $\mathbf{x}_\mathrm{m}$ as the middle point of the circle and $r$ as its radius. Per definition values of $g > 0$ refer to regions inside a shape. An abstract interface to level set information has been designed and used in the mesh generation software described here. With this interface it is now possible to use a variety of different input functions. The software-interface requires the value of $g$ as function of the position as well as its gradient. Level-sets could be given discretely or analytically. Discrete level-sets can be created from various sources (e.g. parameterized boundaries or images).

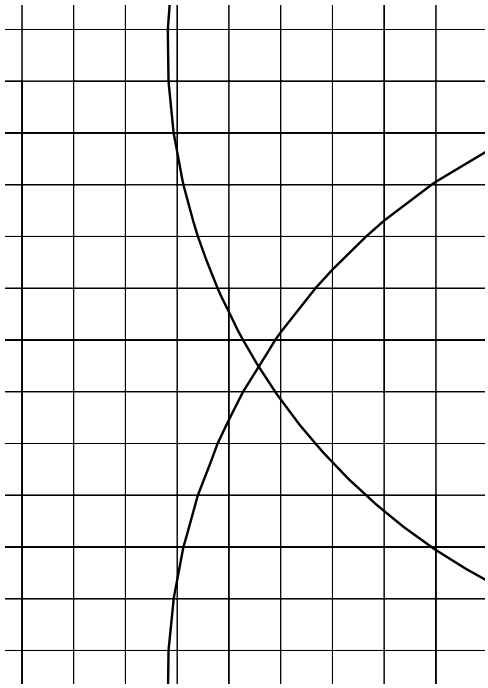

Figure 4.9: Intersecting surface-lines.

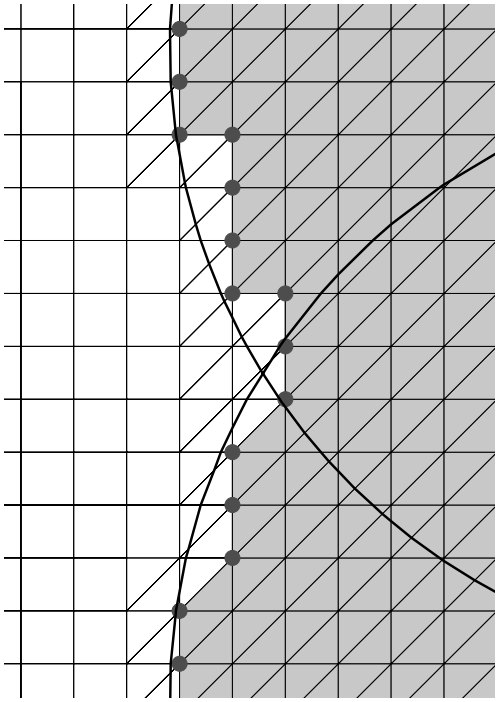

Figure 4.10: After first triangulation.

## 4.3 Isotropic Mesh Generation

A major advantage of using level-sets as input is the easy generation of anisotropic meshes. If the following condition is fulfilled:

$$|\nabla g| = C = const, \tag{4.3}$$

iso-lines of a $g$-function will be parallel to the desired boundary. Hence those iso-lines are ideal grid-lines for anisotropic refinement near boundaries. Before any anisotropic region can be refined, however, a basic isotropic grid has to be created. This grid has to exactly represent the shape defined by a boolean combination of different $g$-functions. A very important feature of the described meshing algorithm is, that there is always a mesh with all logical connections available. This is quite different from other algorithms which have to create the grid out of a boundary definition. Both popular methods, marching front and elliptic mesh generation (i.e. mesh optimization starting from an initial triangulation), share this.

In order to create a triangular grid, using an elliptic meshing algorithm, the mesh generator has to find an initial triangulation of a given set of boundary nodes. Nodes are then successively entered and the mesh is optimized, until the desired resolution has been reached. All this work has to be done to finally get a body conformal and isotropic triangular grid.

For isotropic regions a Cartesian mesh is very well suited. On Cartesian grids it is possi-

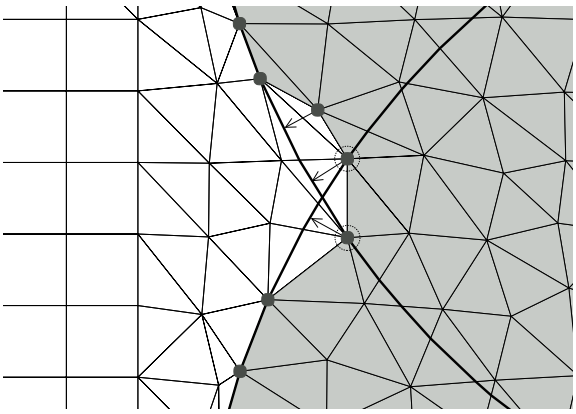

Figure 4.11: Finding the correct level-set.

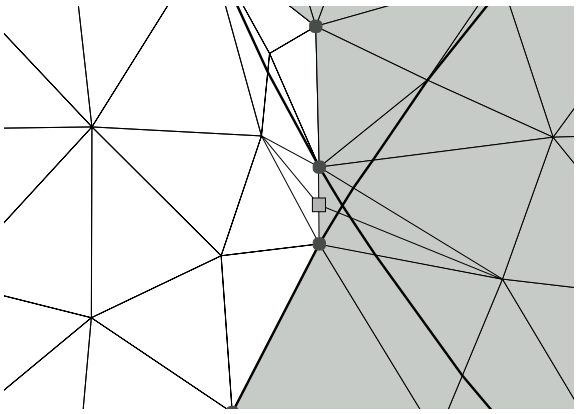

Figure 4.12: Recognition of corners.

ble to find accurate and efficient discretizations. The generation of such a grid is trivial and requires a fraction of the effort which is needed for other grid-types. As already mentioned before, the meshing algorithm described here will begin with a Cartesian grid. This grid might have locally refined regions, though. It is, however, still Cartesian except for small layers to provide connections between refined and not refined parts. Using the information, provided by the level sets, the Cartesian grid can be locally broken up into a triangular grid. Future surface nodes can be detected and marked. An elliptic mesh generation method is then used to smooth these regions and to get the surface nodes on their corresponding surfaces. The initial grid covers a domain which completely encloses all shapes and boundaries. During the generation process it will stay this way. That means there will be a mesh inside the objects to be meshed as well. After the meshing is finished, it is possible to cut out inner parts or to leave them in the mesh. If the inner parts are cut, the result will be exactly the same as with other algorithms, a body conformal grid. As stated above a Cartesian grid is a very good starting point for this algorithm. It is, however, not restricted to that. Curvilinear or triangulated or even a hybrid grids might serve as initial mesh.

Starting from a Cartesian mesh the way how to obtain a body conformal multi element mesh shall be briefly described now. Figure 4.9 shows a zoom of an initial grid plus two intersecting surface lines. The desired shape is to the right of the two lines in figure 4.9 (see also figure 4.8 for the complete shape). A boolean OR is used in this example. The first step is to detect all elements (quads) which are in the desired shape. These elements will be triangulated (cut in a half) and marked as inside-elements. Elements, which are within a certain neighboring region of the shape, will be triangulated as well, but not marked. The size of this neighbor region is adjustable. After the initialization the set of inside-elements forms a rough representation of the desired body. All surface nodes can be detected and marked as well. Every surface node will be assigned to the nearest surface-line. These surface nodes now have to move to their assigned surface-lines. Figure 4.10 illustrates this initial situation. Inside elements are shaded in grey. Surface nodes are marked with small circles. In triangular regions standard elliptic meshing techniques

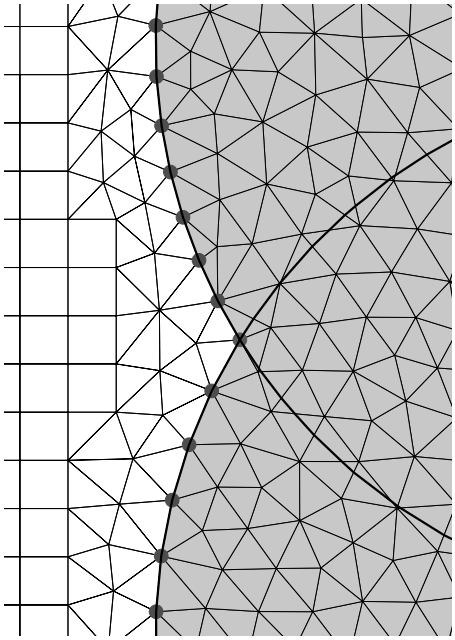

Figure 4.13: Final grid around corner.

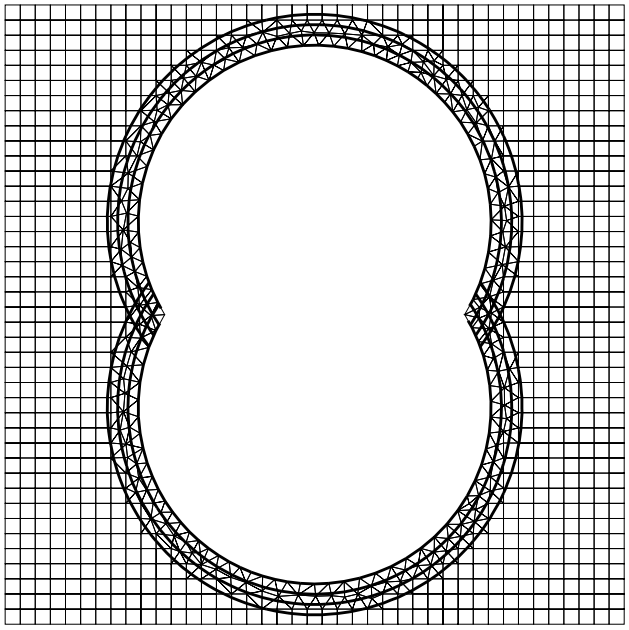

Figure 4.14: Isotropic grid.

are used. Points are moved in order to optimize the shape of attached triangles (mesh-smoothing). Local reconfigurations (edge-swapping between two triangles) are performed to fulfill the Delaunay criterion. Furthermore points might be added or deleted within and outside the shape. It is, however, important to keep the surface. For example it is forbidden to reconfigure an inside with an outside triangle, since this would break up the surface. The so called surface nodes try to move onto their assigned surface-lines. An iterative procedure is used to achieve this. A new position for a surface node is computed as follows:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \omega \left( \frac{g(\mathbf{x}^k)}{(\nabla g)^2} \nabla g \right) \tag{4.4}$$

Where $\omega$ is a relaxation parameter. In case of $|\nabla g| = const$ in the whole domain and $\omega = 1$ a position on the zero iso-line of the $g$-function would be reached in one step. Otherwise the iteration (4.4) becomes a Newton-like iteration perpendicular to the boundary. It is, however, possible that the surrounding mesh hinders a node moving onto its surface. In this case the node is moved as far as possible. In figure 4.11 a node can be seen which already moved onto its surface. This surface-line, however, is not the real surface of the shape. Here a simple rule has to be applied to assign the node to a different level set. Whenever a node has reached $g = 0$ of its level set, but it is still inside the total shape, the next best level set will be used for this node. The decision if a point is within the whole shape or not is easy, since the values of all level set functions on the node are known. With this knowledge and the boolean operators between the different $g$-functions it can be decided if a position is inside the shape or not. Next best means the one with

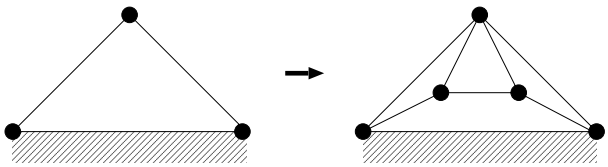

Figure 4.15: Inserting a quadrangle.

the lowest positive g-value. Please note that a node cannot be reassigned to a level set it once has been connected to before. By doing so loops are prevented from occurring in this iterative process. If a node would have to be reassigned, a similar movement method as it is used for corners will be applied.
Another important topic is the recognition of corners. Corners are intersections of different level sets. Of course only those intersections are corners, which are on the real surface. In figure 4.12 two nodes are shown, which both reached the outer surface of the desired shape. These nodes are assigned to different $g$-functions. That means there should be a corner between the nodes. In this case a new node is created in the middle of the edge connecting both surface nodes (shown as a small square). The new node will then be assigned to both level sets. Movement of corners is hence performed using the following formulation:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \omega_1 \left( \frac{g_1(\mathbf{x}^k)}{(\nabla g_1)^2} \nabla g_1 \right) - \omega_2 \left( \frac{g_2(\mathbf{x}^k)}{(\nabla g_2)^2} \nabla g_2 \right) . \tag{4.5}$$

The relaxation parameters $\omega_1$ and $\omega_2$ should be less than $\frac{1}{2}$ for corner nodes. Figure 4.13 shows the surrounding of the corner with all surface and corner nodes in their proper positions. Now it has to be decided if the mesh within the shape should be cut out, or not. If it has to be cut out the former surface and corner nodes become boundary nodes. Generation of an isotropic unstructured grid is finished at this point. The method could be summarized as follows:

1. A Cartesian mesh is generated, which is large enough to cover the whole domain.
2. Optionally this base mesh can be refined in certain regions.
3. Quads which are close to the desired boundaries are triangulated.
4. Later surface nodes are detected and marked.
5. Surface nodes are moved onto the real surfaces. During this process corners can be detected and moved on the correct positions.
6. If wanted so, inner parts of the grid can be deleted.

## 4.4 Creation of Quad-Layers

Figure 4.14 shows a zoom around the left corner of the shape, together with iso-lines for both $g$-functions. It can be seen, that the iso-lines would be perfect grid lines for a

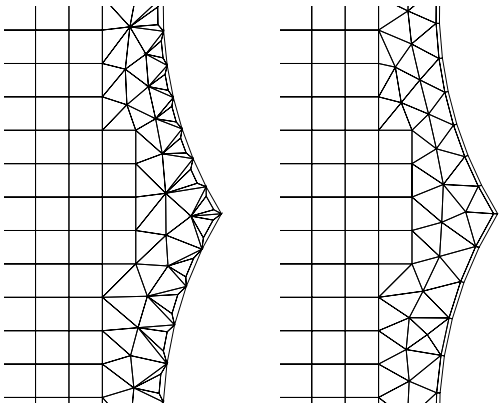

Figure 4.16: First layer of quads.

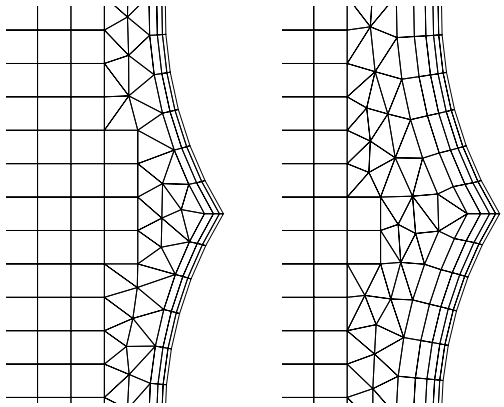

Figure 4.17: Additional layers.

boundary layer grid. Almost everything that is needed to create layers of quadrangular elements around the shape is already included in the basic mesh generation algorithm. To create one layer an initial set of quads is created around the shape. Figure 4.15 illustrates how a bilinear element is created on a surface edge. The same method can also be used to create a new quadrangular element on top of another one. Now the new nodes are moved in the same way as the surface nodes have been treated during the initial meshing. They do not move on the zero iso-line anymore. Instead a value $g_{\text{line}}$ is used, which has to be stored for every node belonging to one of the quadrangular layers. The movement formula becomes:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \omega \left( \frac{g(\mathbf{x}_{old}) - g_{\text{line}}}{(\nabla g)^2} \nabla g \right) . \tag{4.6}$$

In figure 4.16 it can be seen how this algorithm creates an initial quadrangular layer for the example mesh. Figure 4.17 shows the insertion of two more layers into this grid. The mesh basically consists of three different zones. A Cartesian far field, a curvilinear boundary layer grid and a small triangular region to fill the space in between. Although these different zones can be observed, the mesh is not treated in zones. The whole mesh is unstructured containing different element types.

## 4.5 Different Input Types

Three different types of inputs have been used so far. They shall be briefly explained in the three following paragraphs.

### Analytical Functions

Analytical functions are probably the most obvious input for this kind of mesh generator. They proved to be highly efficient for testing and developing this approach. There are, however, many geometrical features which can be ideally described by analytical functions, also for "real world" grids. Consider for example bounding boxes of computational

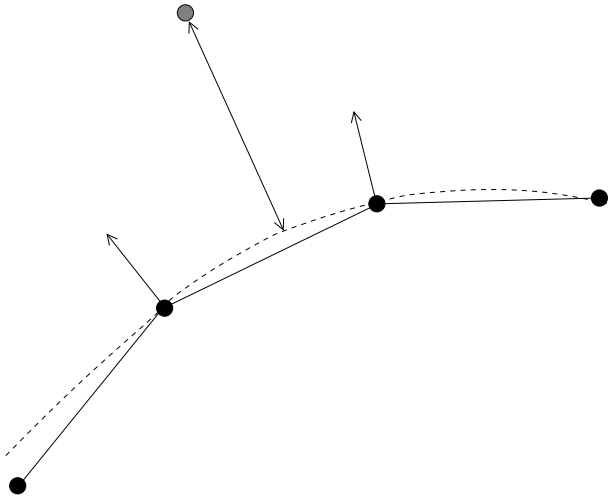

Figure 4.18: Polygons as input.

domains, or straight lines, which occur in many geometries. As far as possible analytical functions should be used, because they offer the fastest way of describing a geometry. This means fast in terms of computational time as well as in terms of necessary user interaction.

### Polygonal Input

Polygons are the classical way to describe two-dimensional boundaries for mesh generation. To make polygons usable for the approach presented here, they have to be converted into an appropriate scalar function. A simple geometric reconstruction can be made (see figure 4.18). For any point $\mathbf{x}$ a corresponding segment of a given polygon can be found. A polynomial ansatz can be made to describe the, originally curved, boundary for this segment (see the dashed line in figure 4.18). For practical applications a third order polynomial proved to give good results. Unfortunately this method involves a search-process for the segment to be used. In principle this search has a linear complexity with the number of segments. If the method should be extended to three dimensions, appropriate steps have to be taken to accelerate this search (e.g. geometrical rastering).

### Images as Input

Another possible input for this type of mesh generator is an image. An image can be easily converted into a $g$-function, which is given on discrete nodes. This scalar function could either be given on the natural grid, which is provided by the image's pixel spacing or alternatively on a coarser grid. A coarser grid will lead to a faster mesh generation and the resolution will still be sufficient for many problems. Furthermore high frequency digitalization errors can be smoothed out this way. Figure 4.19 shows a small image which has been digitalized from a hand-drawn sketch and which then served as an input for the mesh generator. To get a sufficient discretized level set for an image, equation (4.3) has

Figure 4.19: Image as input for mesh generation.

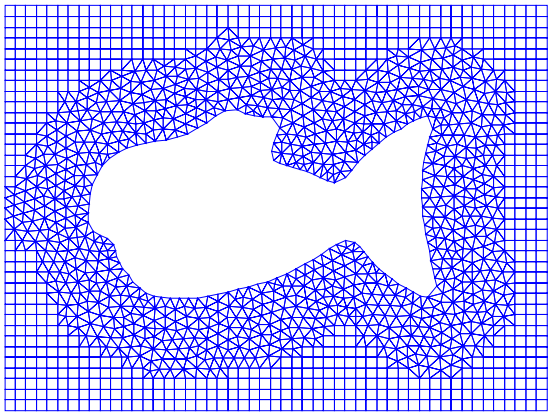

Figure 4.20: Mesh generated out of image.

to be roughly fulfilled in the vicinity of the boundary. Please note that (4.3) has not to be fulfilled as accurately as it is needed for level-set based front-tracking methods. The only thing that has to be exact is the zero-level of the $g$-function. The initial position of the zero-level can be obtained by comparing all pixel intensities with a threshold value. Figure 4.20 shows an isotropic grid which has been created using an image. A single level-set function has been used and the values were discretized on the natural mesh given by the image.

## 4.6 Moving Boundaries

Level sets as geometry description offer an easy handling of moving boundaries. Even geometries, changing in time, would be possible to treat. In this case a transport equation for the scalar level-set function must be used:

$$\frac{\partial g}{\partial t} + \mathbf{c}\,\nabla g = 0\,. \tag{4.7}$$

This transport equation is in fact exactly what is used to track discontinuous solutions, while the term $\mathbf{c}\,\nabla g$ represents the local normal propagation speed. In analogy, applied to the boundaries here, the term represents the local normal surface speed.

### 4.6.1 Two Moving Circles

The test case shown here, however, does not have changing geometries. The $g$-functions are only moved through the grid. Two circular shapes exist in the domain. Both are also meshed in their inside. This is needed to enable these shapes to merge together. The upper circle slowly moves downwards in the mesh. In the above equation 4.7, this corresponds to a constant propagation speed $\mathbf{c}$. As the upper circle hits the lower one,

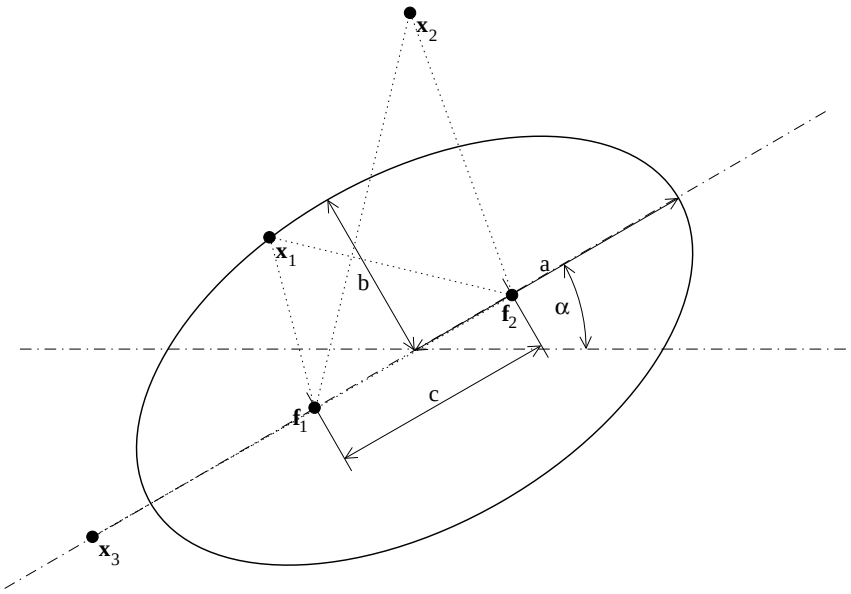


Figure 4.21: $g$ function for an ellipse.

both will merge and form a new body. The new shape will be similar to the one used in sections 4.3 and 4.4. Figure 4.23 shows a couple of snapshots of the upper circle moving downward. and merging with the lower one. As it moves further downward the two circles will part and form to independent shapes again. Movement of bodies is realized in a very simple manner. First of all the origin of the $g$-function is shifted. After this is done the same inside-element detection as described in section 4.3 is performed. This might reduce the amount of mesh-smoothing needed. It becomes essential if two shapes have to merge. This simple example shows how the meshing algorithm is able to deal with a changing mesh-topology.

### 4.6.2 A Rotating and Deforming Shape

This example will use two intersecting ellipses to define a shape. The semimajor axes of both ellipses form an angle of 90 degrees, whereas the centers are identical. The eccentricity is the same and by altering the eccentricity different shapes can be obtained. Figure 4.21 shows a universal ellipse. The eccentricity is defined as

$$e = \sqrt{1 - \frac{b^2}{a^2}} \tag{4.8}$$

with $a$ as the semimajor and $b$ the semiminor axis. For $e = 0$ a circle would be obtained, whereas $e = 1$ would lead to a square. As input function for the mesh generator a simplified $g$-function shall be used.

$$g(\mathbf{x}) = |\mathbf{x} - \mathbf{x}_{\mathrm{f1}}| + |\mathbf{x} - \mathbf{x}_{\mathrm{f2}}| - 2a \tag{4.9}$$

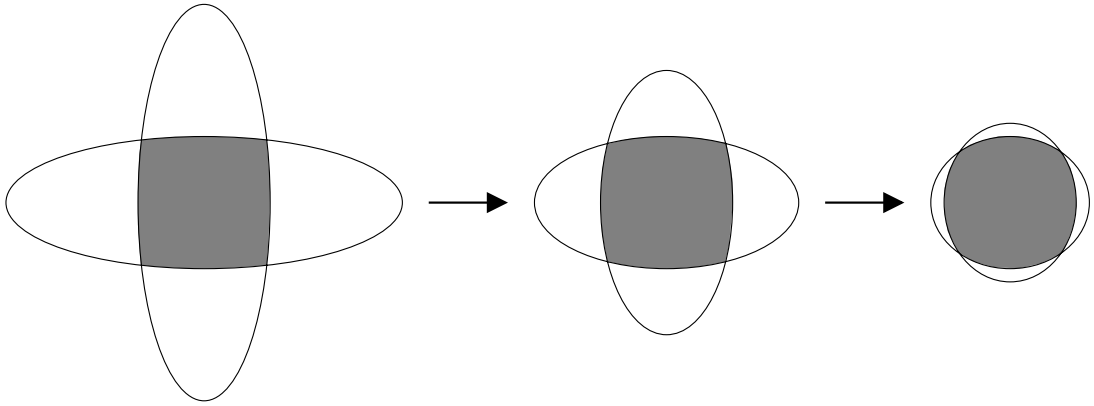

Figure 4.22: Two intersecting ellipses.

$\mathbf{x}_{f1}$ and $\mathbf{x}_{f2}$ describe the two focal points of the ellipse. (4.9), however, does not define an exact distance function. For any point, that is on the ellipse (e.g. point $\mathbf{x}_1$), (4.9) will return zero, and thus it exactly describes the geometry. A point on the semimajor axis, or on its extension outside the ellipse (e.g. point $\mathbf{x}_3$), (4.9) returns the exact distance. Figure 4.22 shows how shapes ranging from a circle to a square can be obtained by using two ellipses and varying the eccentricity $e$. If now the level-sets are tilted, or their eccentricity is modified, the mesh generator is able to adapt the grid to the changed situation. Figure 4.24 shows how the mesh evolves and follows the level-sets.

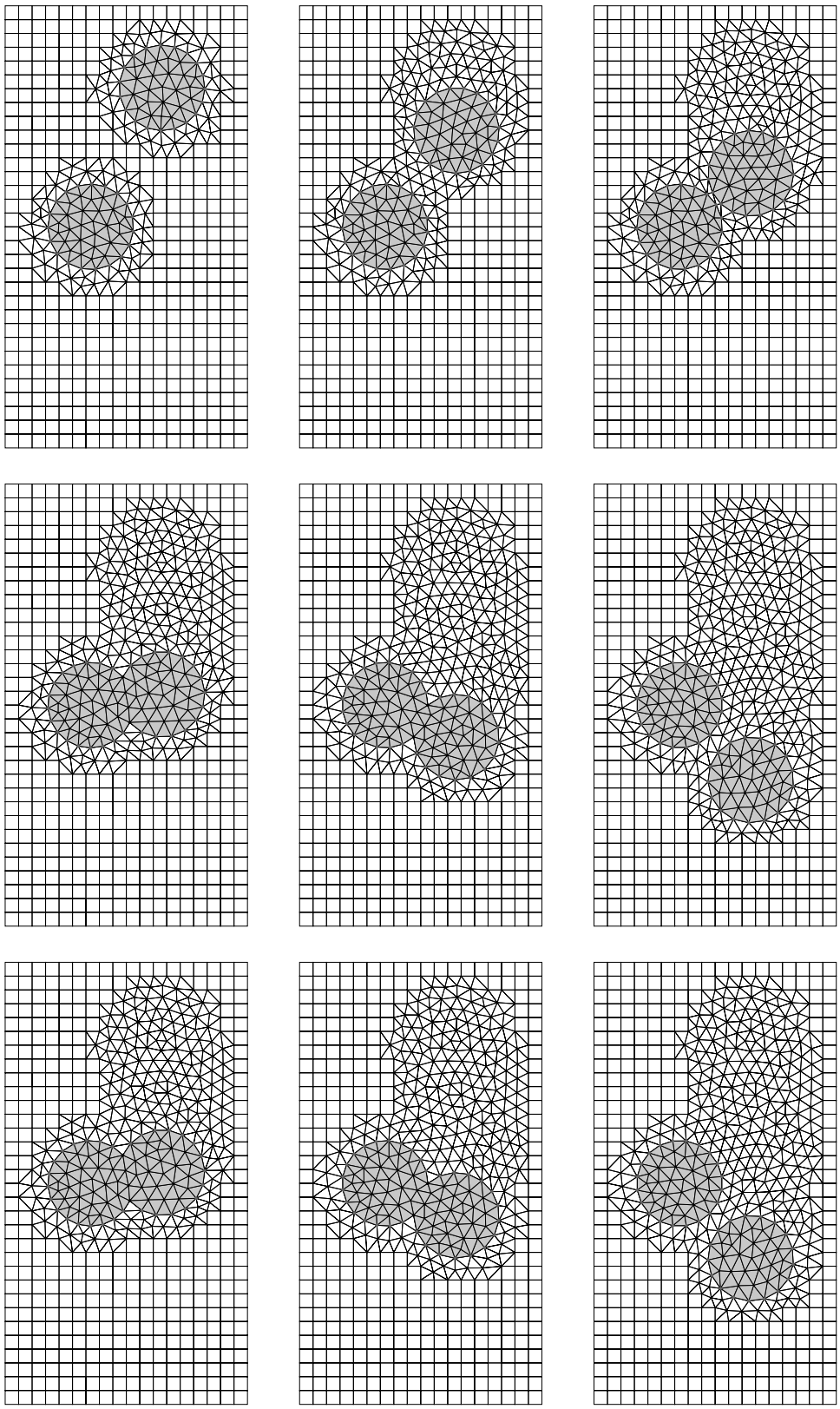

Figure 4.23: Merging and parting shapes.

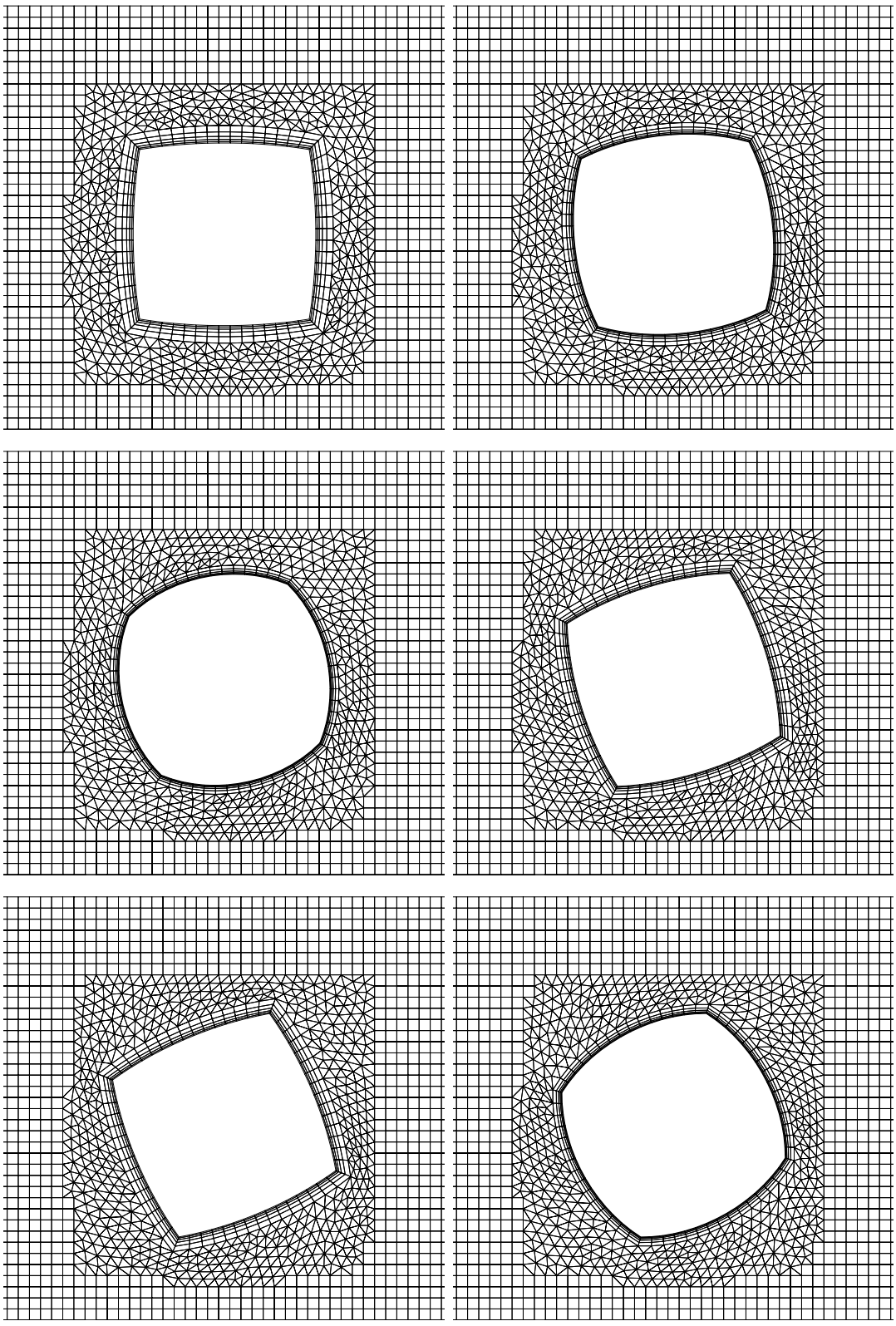

Figure 4.24: A changing geometry.