Algorithms for Quadrilateral and Hexahedral Mesh Generation

Robert Schneiders
MAGMA Gießereitechnologie GmbH
Kackertstr. 11
52072 Aachen
Germany
Email: R.Schneiders@magmasoft.de

Abstract

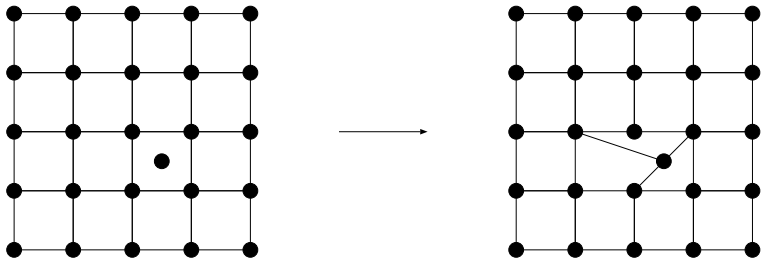
This lecture reviews the state of the art in quadrilateral and hexahedral mesh generation. Three lines of development – block decomposition, superposition and the dual method – are described. The refinement problem is discussed, and methods for octree-based meshing are presented.

1 Introduction

Quadrilateral and hexahedral elements have been proved to be useful for finite element and finite volume methods, and for some applications they are preferred to triangles or tetrahedra. Therefore quadrilateral and hexahedral mesh generation has become a topic of intense research.

It turned out that especially hexahedral mesh generation is a very difficult task. A hexahedral element mesh is a very "stiff" structure from a geometrical point of view, a fact that is illustrated by the following observation: Consider a structured grid and a new node that must be inserted by using local modifications (fig. 1). While this can be done – not in a very elegant way – in 2D, it is impossible in 3D! Thus, one cannot generate a hexahedral element mesh by point insertion methods, a technique which has been used successfully for the generation of tetrahedral element meshes (Delaunay-type algorithms).

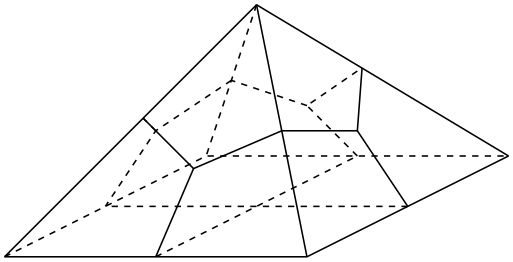
Figure 1: Inserting a point into a structured quadrilateral element mesh



Many algorithms for the generation of tetrahedral element meshes are advancing front methods, where a volume is meshed starting from a discretization of its surface and

building the volume mesh layer by layer. It is very difficult to use this idea for hex meshing, even for very simple structures! Fig. 2 shows a pyramid whose basic square has been split into four and whose triangles have been split into three quadrilateral faces each. It has been shown that a hexahedral element mesh exists whose surface matches the given surface mesh exactly [Mitchell 1996], but all known solutions [Carbonera] have degenerated or zero-volume elements.

Figure 2: Surface mesh for a pyramid



The failure of point-insertion and advancing-front type algorithms severely limits the number of approaches to deal with the hex meshing problem. Most algorithms can be classified either as block-decomposition, superposition or dual methods, which will be presented in section 2, section 3 and section 4.

Adaptive mesh generation is more difficult than for triangular and tetrahedral meshes. Fig. 3 shows an example: The mesh in fig. 3a has been derived from a structured quadrilateral mesh by recursively splitting elements. In order to get rid of the hanging nodes, neighboring elements are split to create a conformal transition to the coarse part of the mesh (fig. 3b). This probleem is equivalent to the generation of quadrilateral/hexahedral meshes from a quadtree/octree structure.

Figure 3: Quadrilateral mesh refinement

a) b)

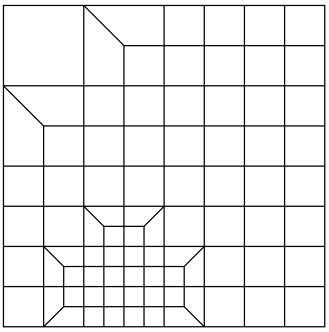
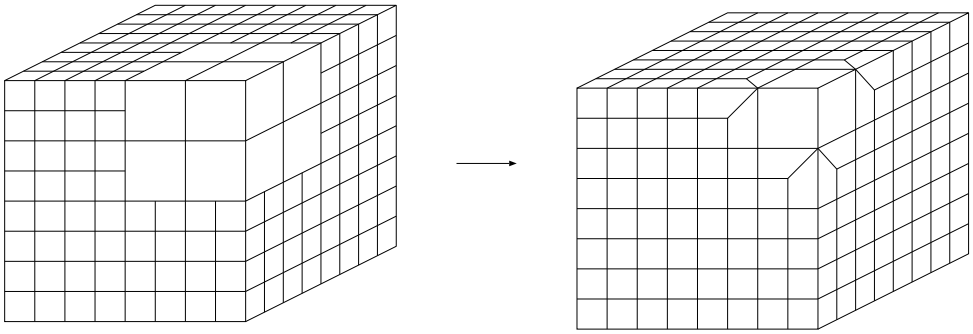


Fig. 4 shows a simple example for the 3D case (motivated by octree-based meshing) where the fine region seems to be connected to the coarse in a reasonable manner. Unfortunately, the solution is not valid!

This can be concluded from the following relation between the number of elements H, the number of internal faces F_i and the number of boundary faces F_b of a hexahedral mesh:

$$6 \cdot H = 2 \cdot F_i + F_b \tag{1}$$

Figure 4: Non-convex transitioning



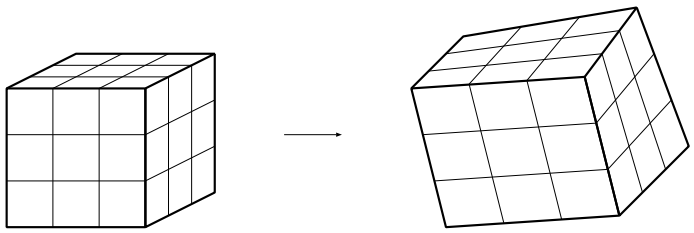
Therefore, for any hexahedral mesh the number F_b of boundary faces must be even. This does not hold for the example of fig. 4, so that no hexahedral mesh for that surface mesh exists, and the transitioning is not valid. The mesh refinement problem will be discussed in detail in section 5, where we will also solve the problem presented in fig. 4.

Much of the research work has been presented in the *Numerical Grid Generation in Computational Fluid Dynamics* and in the *Mesh Generation Roundtable and Conference* conference series, and detailed information can be found in the proceedings. The proceedings of the latter one are available online at the *Meshing Research Corner* [Owen 1996], a large database with literature on mesh generation maintained at Carnegie Mellon University by S. Owen. Another valuable source of online information is the web page *Mesh Generation and Grid Generation on the Web* [Schneiders 1996d] which provides links to software, literature and homepages of research groups and individuals. An overview on the state of the art in mesh generation is given in the *Handbook of Grid Generation* [Thompson 1999], which has been compiled by the *International Society on Grid Generation* (`http://www.isgg.org`).

2 Block-Decomposition Methods

In the early years of the finite element method, hexahedral element meshes were the meshes of choice. The geometries considered at that time were not very complex (beams, plates), and a hexahedral element mesh could be generated with less effort than a tet mesh (graphics workstations were not available at that time). Meshes were generated by using mapped meshing methods: A mesh defined on the unit cube is transformed onto the desired geometry Ω with the help of a mapping $F : [0,1]^3 \rightarrow \Omega$. This method can generate structured grids for cube-like geometries (Fig. 5).

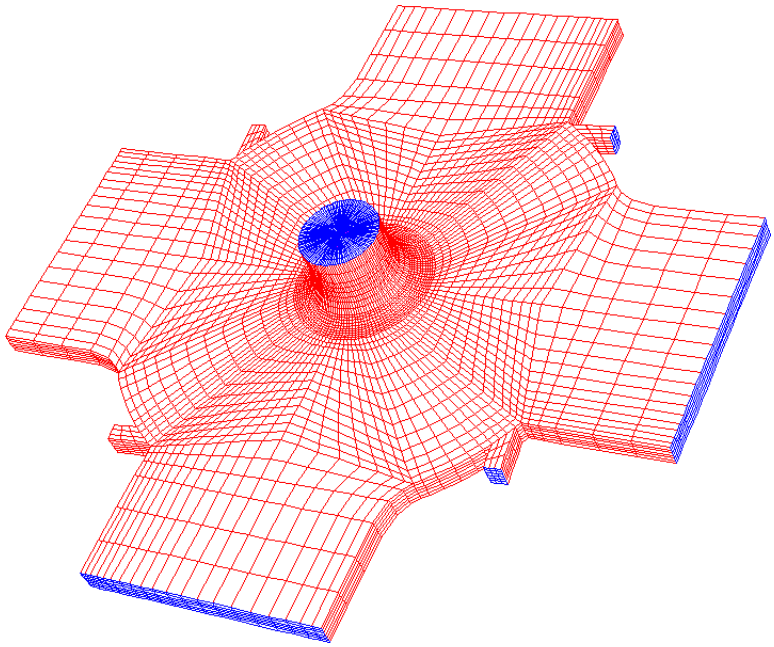
Figure 5: Mapped meshing



The mapping F can be specified explicitly (isoparametric or conformal mapping) or implicitly (solution of an ellpitic or hyperbolic partial differential equation). The problem of finding a suitable mapping F has been the object of major research efforts in recent years, and an overview is given elsewhere in this volume. A summary of the results can be found in the books of Thompson [Thompson 1985] and Knupp [Knupp 1995].

If the geometry to be meshed is too complicated or has reentrant edges, meshes generated by mapped meshing methods usually have poorly-shaped elements and cannot be used for numerical simulations. In this case, a preprocessing step is required: The geometry is interactively partitioned into blocks which are meshed separately (the meshes at joint interfaces must match, a problem considered in [Tam and Armstrong 1993] and [Müller-Hannemann 1995]). These multiblock-type methods are state of the art in university and industrial codes. Fig. 6 shows an example mesh that was generated with Fluent Inc.'s GEOMESH[1] preprocessor.

Figure 6: Multiblock-structured mesh



In principle, most geometries can be meshed in this way. However, there is a limitation in practice: The construction of the multiblock decomposition, which must be done interactively by the engineer. For complex geometries, e.g. a flow field around an airplane or a complicated casting geometry, this task can take weeks or even months to complete. This severely prolongs the simulation turnaround time and limits the acceptance of numerical simulations (a recent study suggests that in order to obtain a 24-hour simulation turnaround time, the time spent for mesh generation has to be cut to at most one hour).

One way to deal with that problem is to develop solvers based on unstructured tetrahedral element meshes. In the eighties, powerful automatic tethrahedral element meshers have been developed for that purpose (they are described elsewhere in this volume).

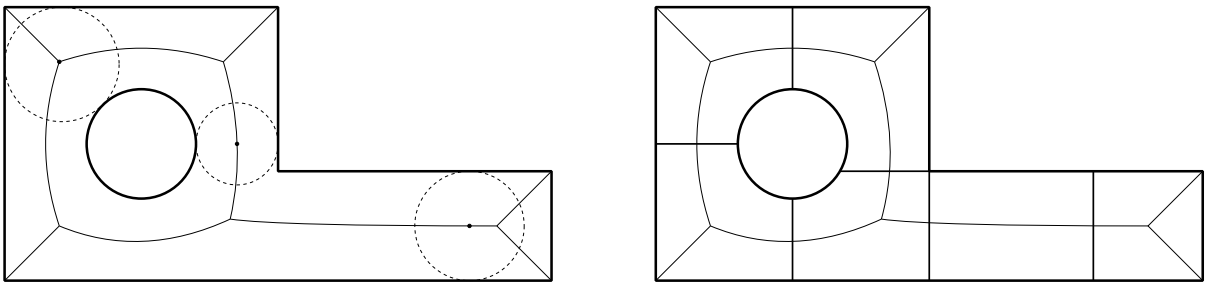
The first attempt to develop a truly automated hex mesher was undertaken by the

[1]GEOMESH is a trademark of Fluent Inc.

finite element modeling group at Queens University in Belfast (C. Armstrong). Their strategy is to automate the block decomposition process. The starting point is the derivation of a simplified geometrical representation of the geometry, the medial axis in 2D and the medial surface in 3D. In the following we will explain the idea (see [Price, Armstrong, Sabin 1995] and [Price and Armstrong 1997] for the details).

We start with a discussion of the 2D algorithm. Consider a domain A for which we want to find a partition into subdomains A_i. We define the *medial axis* or skeleton of A as follows: For each point $P \in A$, the touching circle $U_r(P)$ is the largest circle around P which is fully contained in A. The medial axis $M(A)$ is the set of all point P whose touching circles touch the boundary δA of A more than once.

Figure 7: Medial axis and domain decomposition



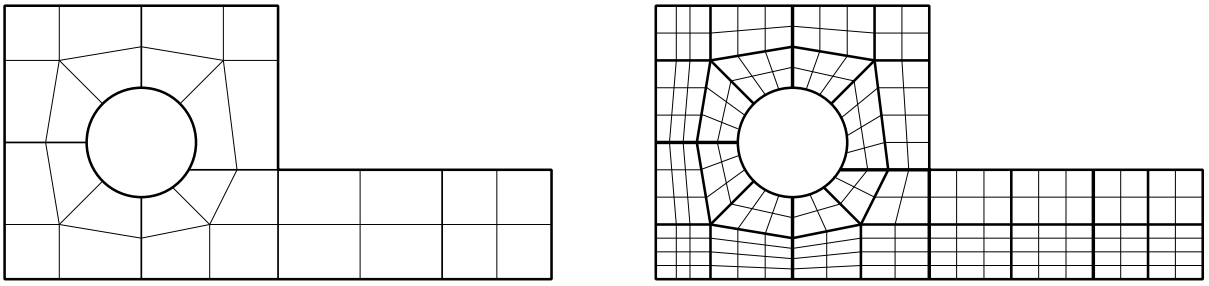
The medial axis consists of nodes and edges and can be viewed as a graph. An example is shown in fig. 7: Two circles touch the boundary of A exactly twice; the respective midpoints fall on edges of the medial axis. A third circle has three common points with δA, the midpoint is a branch point (node) of the medial axis. The medial axis is a unique description of A: A is the union of all touching circles $U_r(P), P \in M(A)$.

The medial axis is a representation of the topology of the domain and can thus serve as a starting point for a block decomposition (fig. 7 and 8). For each node of $M(A)$ a subdomain is defined, its boundary consisting of the bisectors of the adjacent edges and parts of δA (a modified procedure is used if non-convex parts of δA come into play [Price, Armstrong, Sabin 1995]). The resulting decompostion of A consists of $n-$polygons, $n \geq 3$, whose interior angle are smaller than 180^o. A polygon is then split up by using the midpoint subdivision technique [Tam and Armstrong 1993], [Blacker and Stephenson 1991]: It's centroid is connected to the midpoints of it's edges, the resulting tesselation consists of convex quadrilaterals. Fig. 8 shows the multiblock decompostion and the resulting mesh which can be generated by applying mapped meshing to the faces.

It remains to explain how to construct the medial axis. This is done by using a Delaunay technique (fig. 9a): The boundary δA of the domain A is approximated by a polygon p, and the constrained Delaunay triangulation (CDT) of p is computed. One gets an approximation to the medial axis by connection of the circumcircles of the Delaunay triangulation (the approximation is a subset of the Voronoi diagram of p).

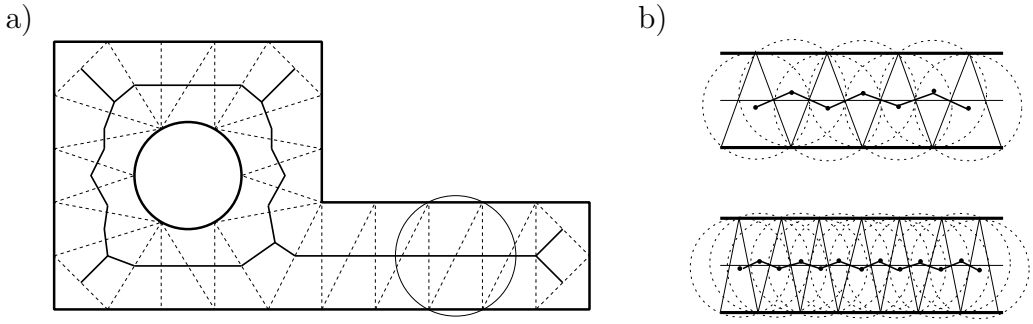
By refining the discretization p of δA and applying this procedure one gets a series of approximations that converges to the medial axis (fig. 9b). Consider a triangle of the CDT to p: Part of it's circumcircle overlaps the complement of A. The overlap for the

Figure 8: Multiblock-decomposition and resulting mesh



circumcircle of the respective triangle of the refined polygon's CDT is significantly smaller. If the edge lengths of p tend to zero, the circumcircles converge to circles contained in A which touch δA at least twice. Their midpoints belong to the medial axis.

Figure 9: Approximating the medial axis



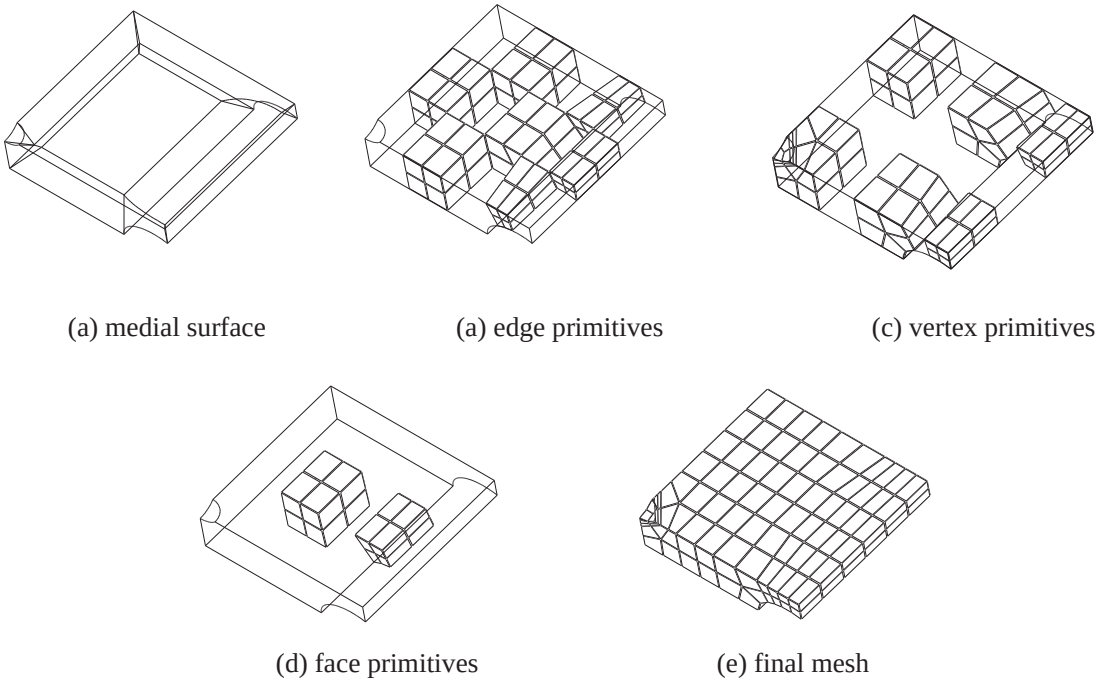
In three dimensions, the automization of the multiblock decomposition is found by using the medial surface. The medial surface is a straightforward generalization of the medial axis and is defined as follows: Consider a point P in the object A and let $U_r(P)$ the maximum sphere centered in P that is contained in A. The medial surface is defined as the set of all points P for which $U_r(P)$ touches the object boundary δA more than once. P lies on

- a face of the medial surface, if $U_r(P)$ touches δA twice
- an edge of the medial surface, if $U_r(P)$ touches δA three times
- a node of the medial surface, if $U_r(P)$ touches δA four times or more.

The medial surface is a simplified description of the object (again, A is the union of the touching spheres $U_r(P)$ for all points P on the medial surface). The medial surface preserves the topology information and can therefore be used for finding the multiblock decomposition.

Armstrong's algorithm for hexahedral element mesh generation follows the line of the 2D algorithm (fig. 10). The first step is the construction of the medial surface with the help of a constrained Delaunay triangulation (Shewchuk [Shewchuk 1998] shows how to construct a surface triangulation for which a constrained Delaunay triangulation exists). The medial surface is then used to decompose the object into simple subvolumes. This is the crucial step of the algorithm, and it is much more complex than in the two-dimensional case. A number of different cases must be considered, especially if non-convex

Figure 10: Medial-surface algorithm for the generation of hexahedral element meshes

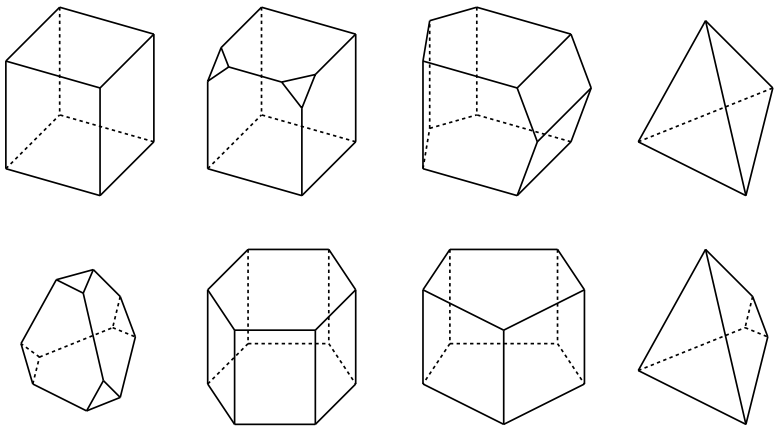


(a) medial surface (a) edge primitives (c) vertex primitives

(d) face primitives (e) final mesh

edges are involved; they will not be discussed here, the interested reader is referred to [Price and Armstrong 1997] for the details.

Figure 11: Meshable primitives (selection)



Armstrong identifies 13 polyhedra an object is decomposed to (fig. 11 shows a selection). These meshable primitives have convex edges, and each node is adjacent to exactly three edges. The midpoint subdivision technique [Tam and Armstrong 1993] can therefore be used to decompose the object into hexahedra: The midpoints of the edges are connected to the midpoints of the faces (fig. 12). Then both the edge and face midpoints are connected to the center of the object, and the resulting decompostion consists of valid hexahedral elements.

Fig. 13 shows a mesh generated for a geometry with a non-convex edge. The example highlights the strength of the method: The mesh is well aligned to the geometry, it is a "nice" mesh – an engineer would try to create a mesh like this with an interactive tool.

Figure 12: Volume decomposition by midpoint subdivision

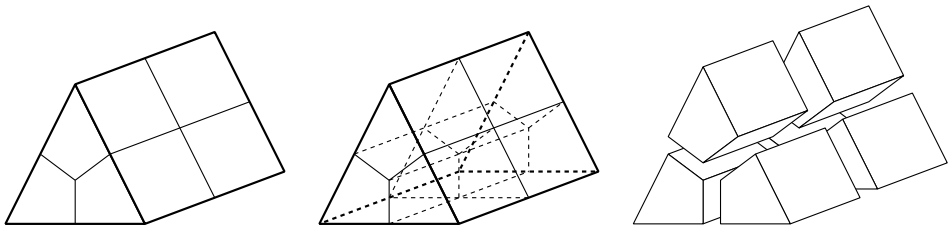
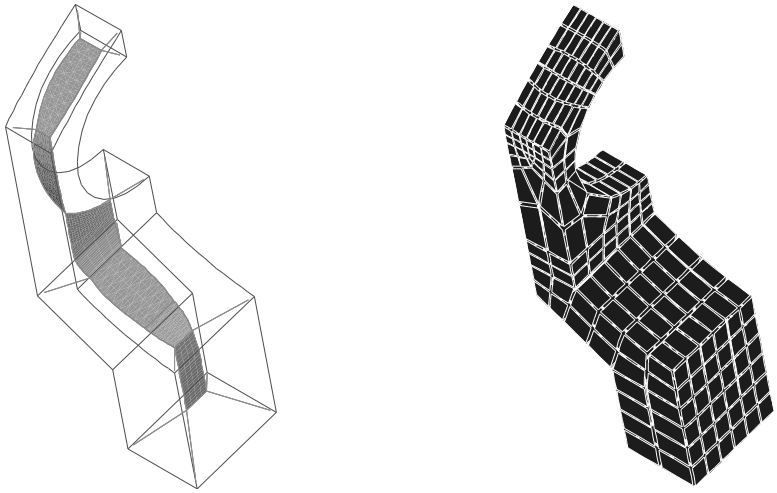


Figure 13: Medial surface and mesh for a mechanical part



The medial surface technique tries to emulate the multiblock decomposition done by the engineer "by hand". This leads to the generation of quality meshes, but there are some inherent problems: Namely, it does not answer the question whether a good block decomposition exists, which may not be the case if the geometry to be meshed has small features. Another problem is that the medial surface is an unstable entity: Small changes in the object can cause big changes in the medial surface and the generated mesh.

Nevertheless, the medial surface is extremly useful for engineering analysis: It can be used for geometry idealization and small feature removal, which simplifies the medial surface, enhances the stability of the algorithm and leads to better block decompositions. The method delivers relatively coarse meshes that are well aligned to the geometry, a highly desirable property especially in computational mechanics. It is natural that an approach to high-quality mesh generation leads to a very complex algorithm, but the problems are likely to be solved.

Two other hex meshing algorithms based on the medial surface are known in the literature. Holmes [Holmes 1995] uses the medial surface concept to develop meshing templates for simple subvolumes. Chen [Turkkiyah 1995] generates a quadrilateral element mesh on the medial surface which is then extended to the volume.

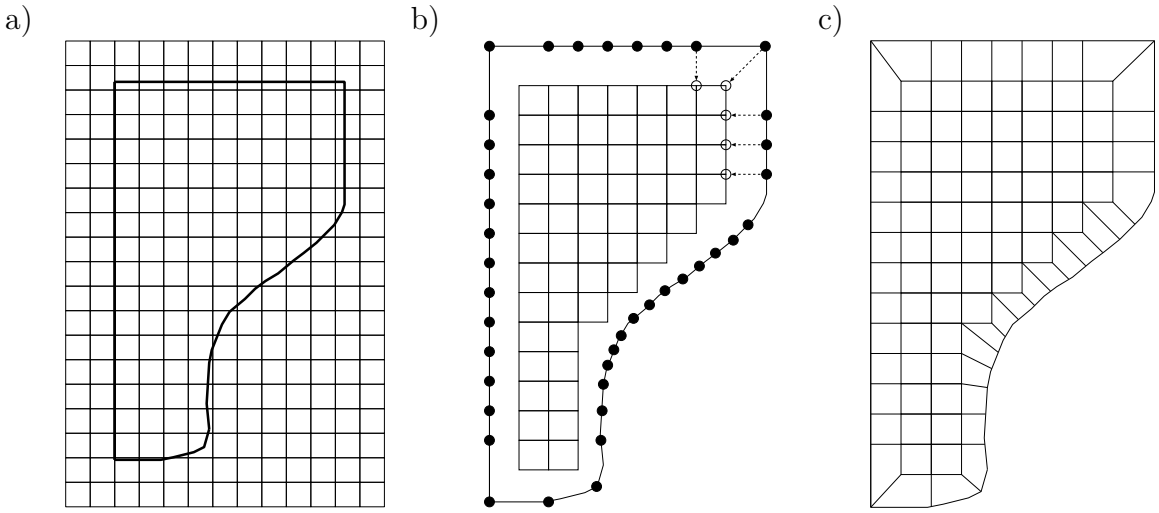
3 Superposition Methods

The acronym *superposition methods* refers to a class of meshing algorithms that use the same basic strategy. All these algorithms start with a mesh that can be more or less easily

generated and covers a sufficiently large domain around the object, which is then adapted to the object boundary. The approach is very pragmatic, but the resulting algorithms are very robust, and there are several promising variants.

Since we have actively participated in this research, we will concentrate on a description of our own work, the grid based algorithm [Schneiders 1996a]. Fig. 14 shows the 2D variant: A sufficiently large region around the object is covered by a structured grid. The cell size h of the grid can be chosen arbitrarily, but should be smaller than the smallest feature of the object. It remains to adapt the grid to the object boundary – the most difficult part of the algorithm.

Figure 14: 2D-grid based algorithm



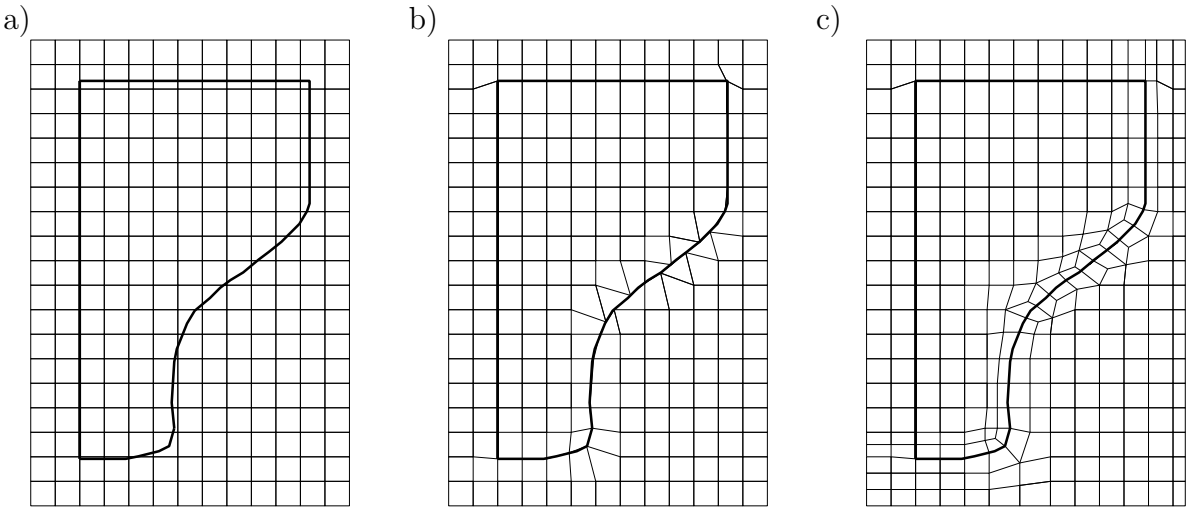
According to [Schneiders 1996a], all elements outside the object or too close to the object boundary are removed from the mesh, with the remaining cells defining the *initial mesh* (fig. 14a, note that the distance between the initial mesh and the boundary is approximately h). The region between the object boundary and the initial mesh is then meshed with the isomorphism technique: The boundary of the initial mesh is a polygon, and for each polygon node, a node on the object boundary is defined (fig. 14b). Care must be taken that characteristic points of the object boundary are matched in this step, a problem that is not too difficult to solve in 2D. By connecting polygon nodes to their respective nodes on the object boundary, one gets a quadrilateral element mesh in the boundary region (fig. 14c).

The "principal axis" of the mesh depends on the structure of the initial mesh, and in the grid based algorithm the element layers are parallel to one of the coordinate axis. Consequently, the resulting mesh (fig. 14) has a regular structure in the object interior and near boundaries that are parallel to the coordinate axis, irregular nodes can be found in regions close to other parts of the boundary. This is typical for a grid based algorithm, but can be avoided by choosing a different type of initial mesh.

The only input parameter for the grid based algorithm is the cell size h. In case of failure, it is therefore possible to restart the algorithm with a different choice of h, a fact that greatly enhances the robustness of the algorithm.

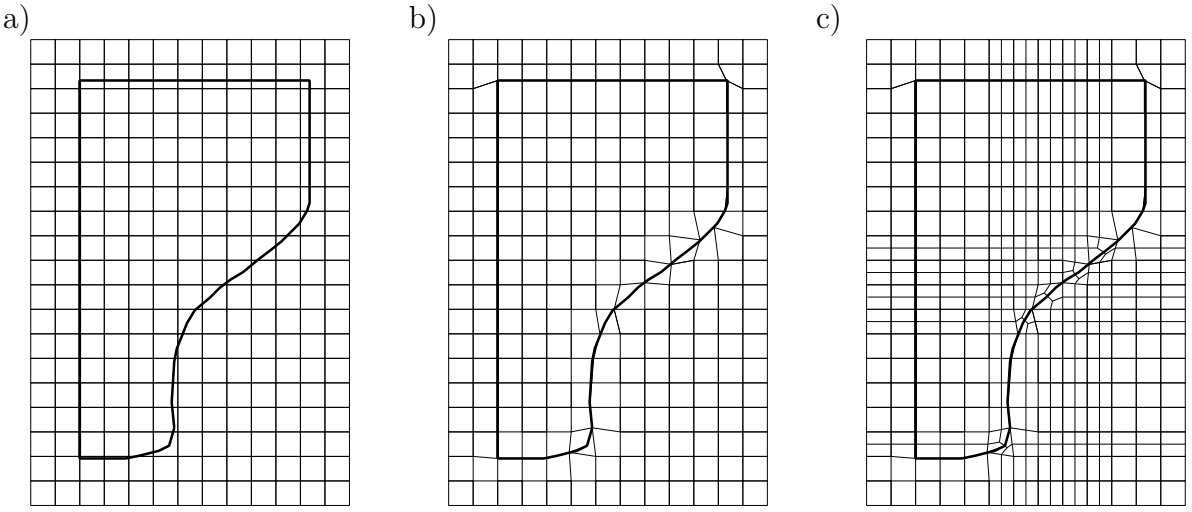
Another way to adapt the initial mesh to the boundary, the projection method, was proposed in [Taghavi 1994] and [Ives 1995]. The starting point is the construction of a structured grid that covers the object (fig. 15a), but in contrast to the grid based algorithm, all cells remain in place. Mesh nodes are moved onto the characteristic points of the object and then onto the object edges, so that the object boundary is fully covered by mesh edges (fig. 15b). Degenerate elements may be constructed in this step, but disappear after buffer layers have been inserted at the object boundary (fig. 15c, the mesh is then optimized by Laplacian smoothing).

Figure 15: Grid based algorithm – boundary adaption by projection technique



The projection method allows the meshing objects with internal faces; the resulting meshes are similar to those generated with the isomorphism techniques, although there tend to be high aspect ratio elements at smaller features of the object. In contrast to the isomorphism technique, the mesh is adapted to the object boundary before inserting the buffer layer.

Figure 16: Grid based algorithm – boundary adaption by cell splitting technique

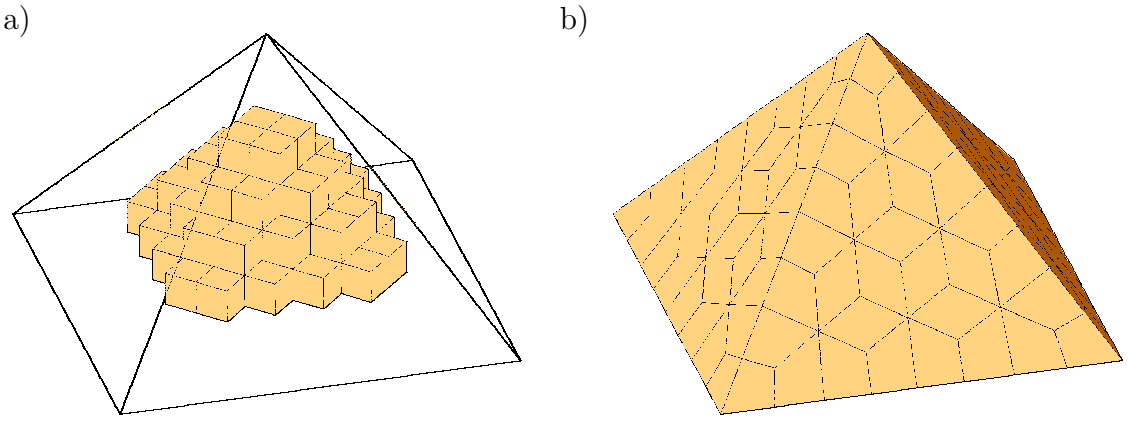


Only recently, a third method has been proposed [Dhondt 1999]. With this approach,

only nodes that are very closed to the bounary are projected onto it. Grid cells that are crossed ty a boundary edge are split (fig. 16b). Midpoint subdivision is used to split the 3- and 5-noded faces (fig. 16c). Dangling nodes are removed by propagating the split throughout the mesh.

The superposition principle generalizes for the 3D case. The idea of the grid based algorithm is shown for a simple geometry, a pyramid (1 quadrilateral, 4 triangular faces, fig. 17). The whole domain is covered with a structured uniform grid with cell size h. In order to adapt the grid to the boundary, all cells outside the object, that intersect the object boundary or are closer the $0.5 \cdot h$ to the boundary are removed from the grid. The remaining set of cells is called the initial mesh (fig. 17a).

Figure 17: Initial mesh and isomorphic mesh on the boundary



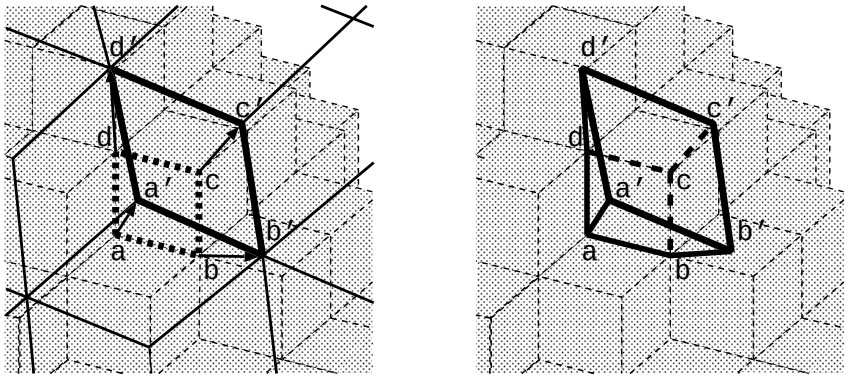
The isomorphism technique [Schneiders 1996a] is used to adapt the initial mesh to the boundary, a step that poses many more problems in 3D than in 2D. The technique is based on the observation that the boundary of the initial mesh is an unstructured mesh M of quadrilateral elements in 3D. An isomorphic mesh M' is generated on the boundary: For each node of $v \in M$ a node $v' \in M'$ is defined on the object boundary, and for each edge $(v, w) \in M$ an edge $(v', w') \in M'$ is defined. It follows that for each quadrilateral $f \in M$ of the initial mesh's surface there is exactly one face $f' \in M'$ on the object boundary. Fig. 17b shows the isomorphic mesh for the initial mesh of fig. 17b.

Fig. 18 shows the situation in detail: The quadrilateral face $(A, B, C, D) \in M$ corresponds to the face $(a, b, c, d) \in M'$. The nodes A, B, C, D, a, b, c, d define a hexahedral element in the boundary region! This step can be carried out for all pairs of faces, and the boundary region can be meshed with hexahedral elements in this way.

The crucial step in the algorithm is the generation of a good quality mesh M' on the object boundary: All object edges must be matched by a sequence of mesh edges, and the shapes of the faces $f' \in M'$ must be non-degenerate. If the surface mesh does not meet these requirements, the resulting volume mesh does not represent the volume well or has degenerate elements. Fulfilling this requirement is a non-trivial task, also the implementation becomes a problem (codes based on superposition techniques usually have more than 100.000 lines of code). We will not describe the process in detail, but some important steps will be discussed for the example shown in figs. 19-24.

Fig. 19a shows the initial mesh for another geometry that does not look very compli-

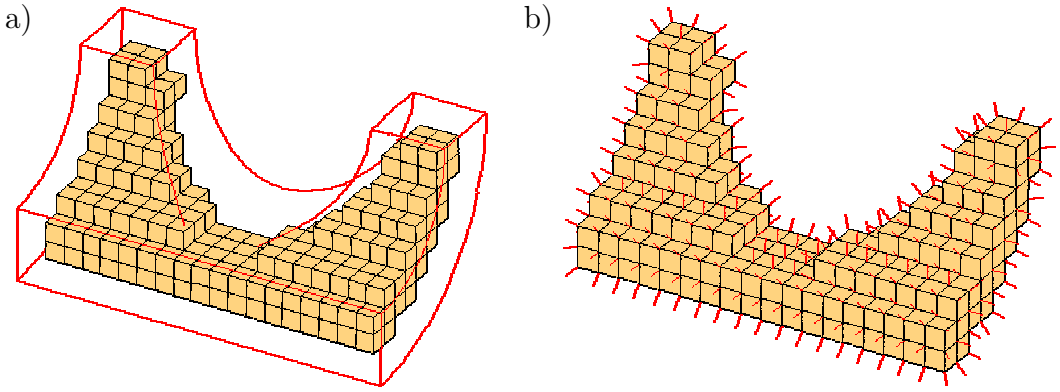
Figure 18: Construction of hexahedral elements in the boundary region



cated but nevertheless is difficult to mesh. The first step of the algorithm is to define the coordinates of the nodes of the isomorphic mesh. Therefore, normals are defined for the nodes on the surface of the initial mesh by averaging the normals N_f of the n adjacent faces f (cf. fig. 19b):

$$N_v = \frac{1}{n} \sum_{f \text{ adj. } v} N_f$$

Figure 19: Initial mesh and normals



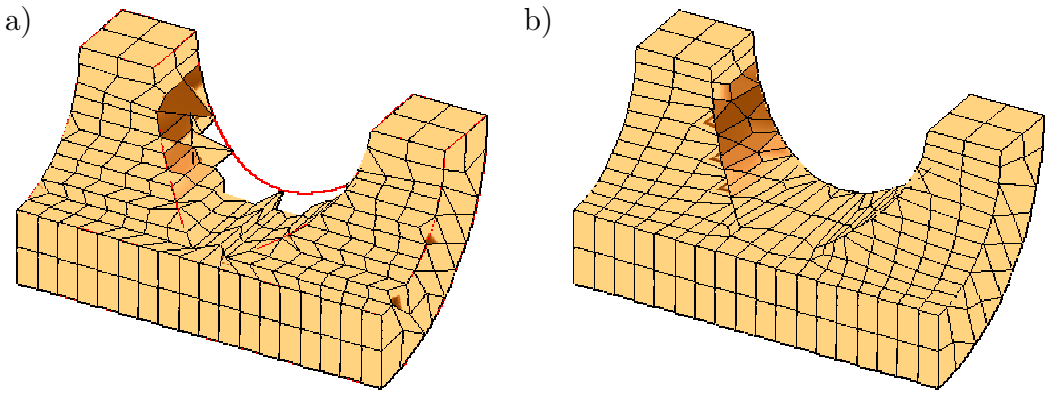
For each point $v \in M$, the position of the corresponding point $v' \in M'$ is calculated as the intersection of the normal N_v with the object boundary. The point v' is then projected onto

- a characteristic vertex P of the object, if $dist(v', P) \leq 0.1 \cdot h$
- a characteristic edge E of the object, if $dist(v', E) \leq 0.1 \cdot h$

In case of projection, a flag is set for the respective node to indicate the entity it has been fixed to. Fig. 20a shows that the quality of the generated surface mesh is unsatisfactory but that at least some of the characteristic vertices and edges of the object are covered by mesh nodes and edges.

For the generation of hexahedral elements in the boundary region, the topology of the surface mesh M' must not be changed, but we are free to modify the location of the

Figure 20: Isomorphic surface mesh



nodes in space. This allows the optimization of the surface mesh by moving the nodes v' to appropriate positions (fig. 20b shows that the quality of the surface mesh can be improved significantly). A Laplacian smoothing is applied to the nodes of the surface mesh: The new position x_i^{new} of a node v' is calculated as the average of the midpoints S_k of the N adjacent faces.

$$x_i^{new} = \frac{1}{N} \sum_{k=1}^{N} S_k$$

The following rules are applied in the optimization phase:

- After a correction step, the nodes are reprojected onto the object boundary.
- Nodes that are fixed to a characteristic vertex of the object are not considered.
- Nodes that are fixed to a characteristic edge are reprojected onto that edge.
- Nodes that are fixed to a characteristic edge but whose neighbors are not fixed are released from that edge.

In the next step, the object vertices and edges are covered by mesh nodes and edges:

- Each object vertex is assigned the closest mesh node.
- Edge capturing: Starting from a vertex, mesh nodes are projected on to an object edge (fig. 21).
- The smoothing procedure is reapplied.

Figure 21: Edge repair

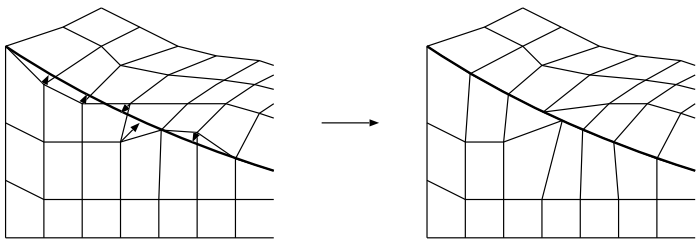


Fig. 20b shows that the surface mesh accurately represents the object geometry and that the overall mesh quality has been improved. Nevertheless, degenerate faces can result

from the edge capturing process if three nodes of a face are fixed to the same characteristic edge. This cannot be avoided if the object edges are not aligned to the "principal axes" of the mesh (cf. fig. 21). There are two ways to deal with the problem.

First, the boundary region is filled with a hexahedral element mesh. Due to the meshing procedure, there are two rows of elements adjacent to a convex edge (fig. 22a). If the solid angle alongside the edge is sufficiently smaller than 180°, the mesh quality can be improved by inserting an additional row of elements, followed by a local resmoothing. At object vertices where three convex edges meet, one additional element is inserted.

Figure 22: Inserting additional elements at sharp convex edges

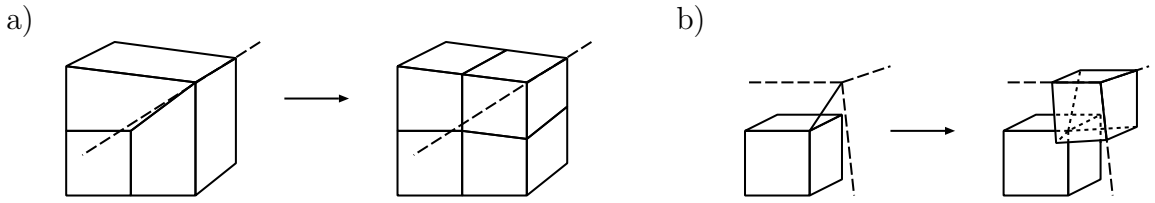


Fig. 24a shows the resulting mesh after the application of the optimization step (note that many degeneracies have been removed). The remaining degenerate elements are removed by a splitting procedure.

Figure 23: Splitting degenerated elements

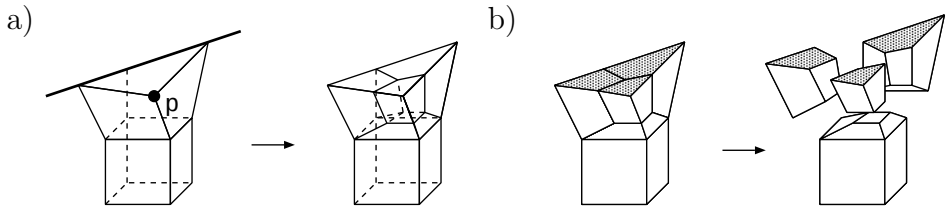


Figure 24: Removing degenerated elements

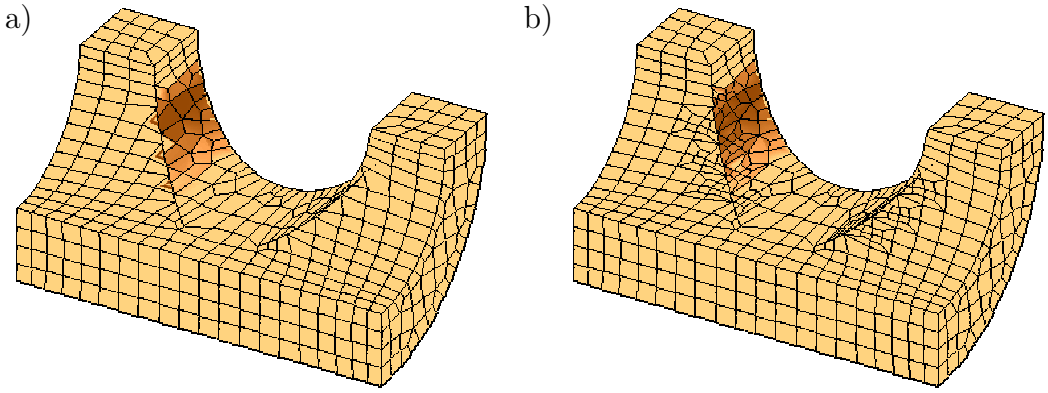
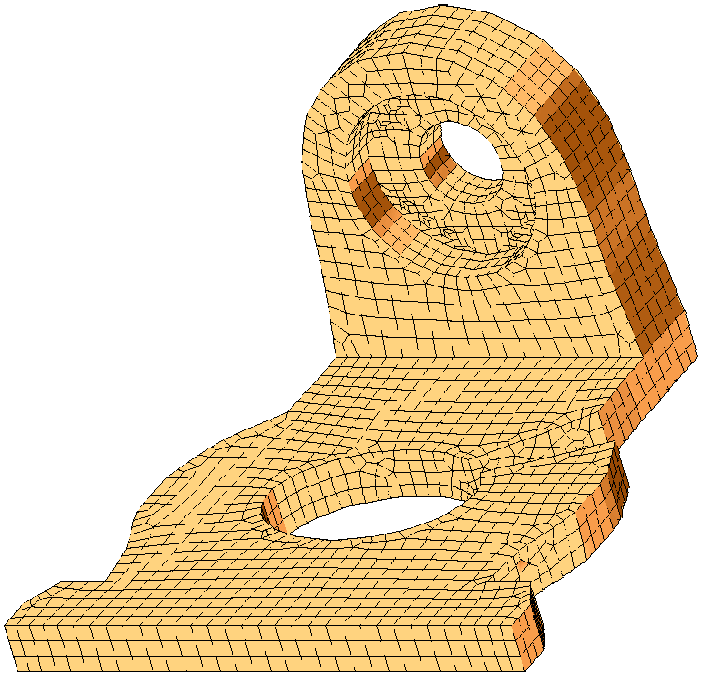


Fig. 23 shows the situation: Three points of a face have been fixed to a characteristic edge, the node P is "free". This face is split up into three quadrilaterals in a way that the flat angle is removed (fig. 23b). The adjacent element can be split in a similar way into four hexahedral elements. In order to maintain the conformity of the mesh, the neighbor

elements must be split up also; it is, however, important that only neighbor elements adjacent to P must be refined, the initial mesh remains unchanged.

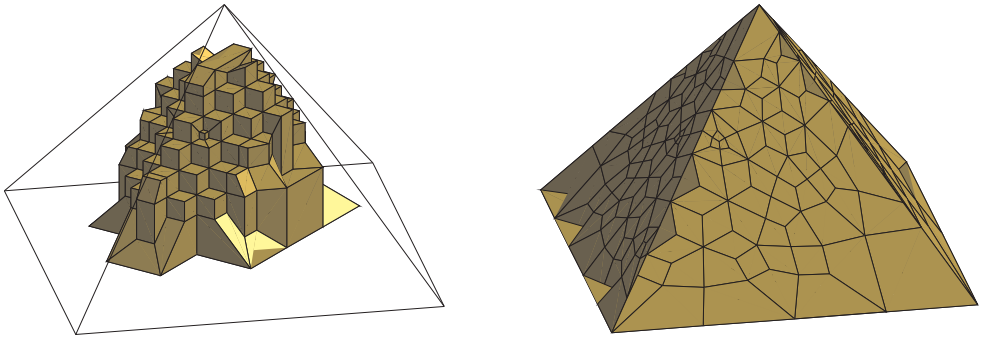
Fig. 24b shows the resulting mesh. Note that the surface mesh is no longer isomorphic to the initial mesh (fig. 19a), since removing the degenerated elements has had an effect on the topology (the mesh in fig. 20b is isomorphic to the initial mesh). The mesh has a regular structure at faces and edges that are parallel to one of the coordinate axes. The mesh is unstructured at edges whose adjacent edges include a "flat" angle and where degenerate elements had to be removed by the splitting operation. Fig. 25 shows another mesh for a mechanical part.

Figure 25: Mesh for a mechanical part



The grid based algorithm is only one out of many possible ways use the superposition principle. One can take an arbitrary unstructured hexahedral mesh as a starting point – the adaptation to the boundary is general, no special algorithm is needed. Fig. 26 shows an example where a non-uniform initial mesh has been generated and adapted to the object boundary by the isomorphism technique.

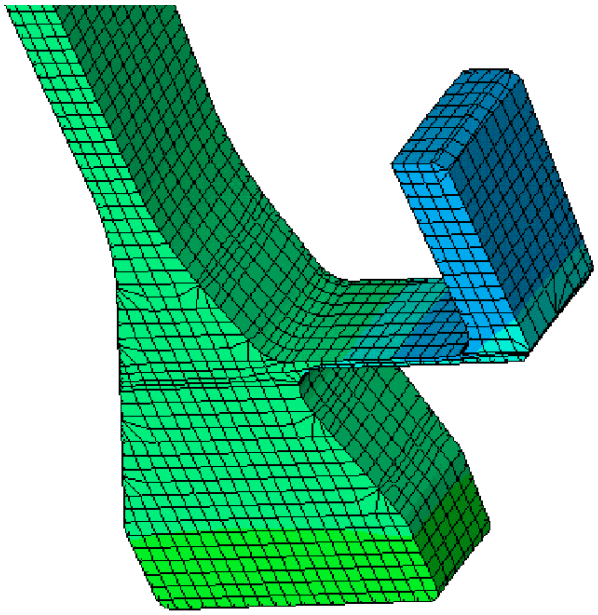
Figure 26: Octree-based initial mesh and isomorphic surface mesh



This makes the superposition principle a powerful tool for hex meshing. Several variants of the idea have been proposed.

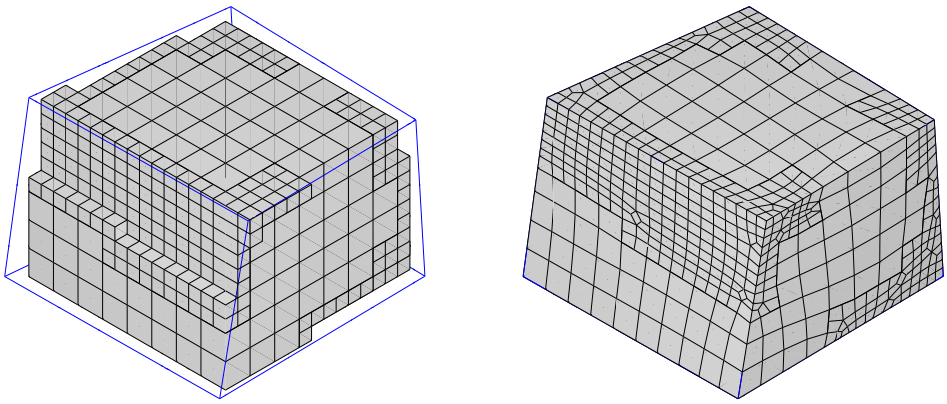
Fig. 27 shows a hexahedral mesh generated to model a part of a turbine. For the small feature, a thin initial mesh has been generated. The mesh was adapted to the boundary by using the method proposed in [Dhondt 1999].

Figure 27: Mesh for a part of a turbine



A weak point of the grid based method is the fact that the elements are nearly equal-sized. This can cause problems, since the element size h must be chosen according to the smallest feature of the object – a mesh with an inacceptable number of elements may result. The natural way to overcome this drawback is to choose an octree-based structure as an initial mesh. NUMECA's new unstructured grid generator IGG/Hexa[2] [Tchon 1997] uses this technique, fig. 28 shows an example of how it works. The initial mesh has hanging nodes, and so does the the mesh on the boundary.

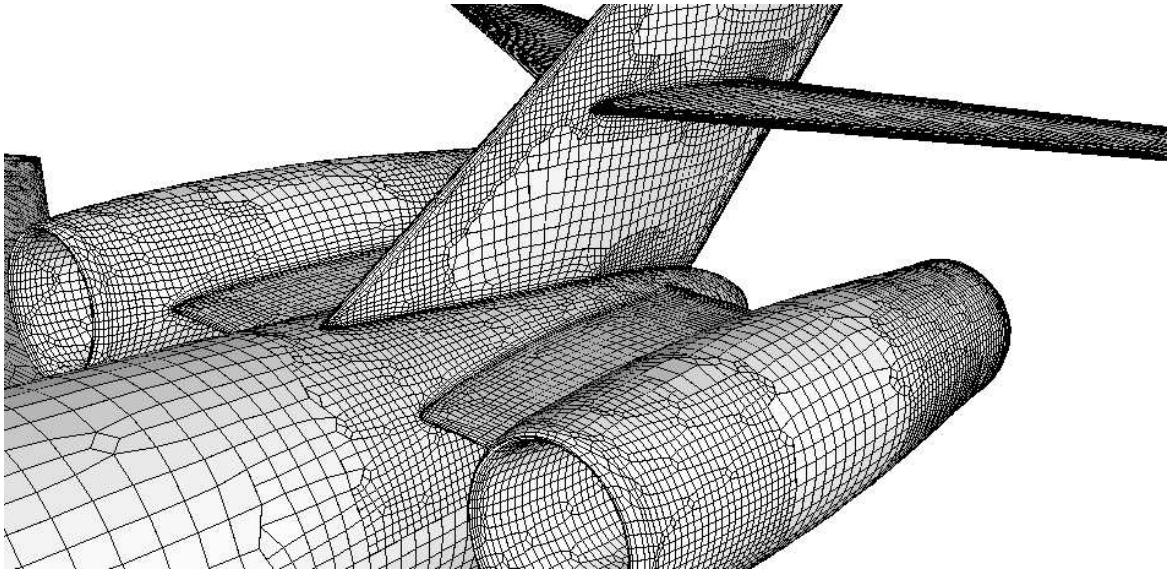
Figure 28: Octree-based mesh generation



[2]IGG/Hexa is a trademark of NUMECA Int.

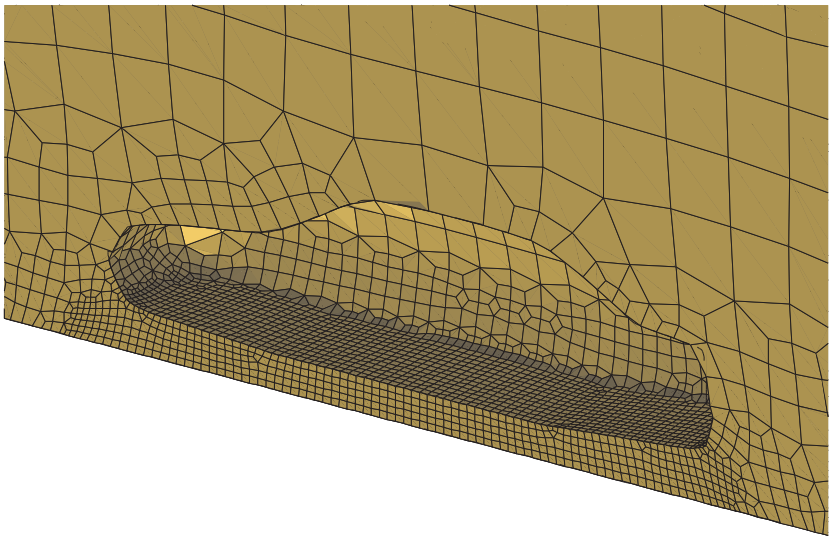
Fig. 29 shows a mesh generated with IGG/Hexa around a generic business jet configuration. The mesh was generated automatically from an initial cube surrounding the geometry. Anisotropic adaptation to the geometry is performed in a fully automated way with a criteria based on normal variation of the surface triangles intersected by the cell. Three layers of high aspect ratio cells are introduced close to the wall.

Figure 29: Mesh around a jet configuration



The mesh in fig. 29 has hanging nodes in the interior, which is of disadvantage for some applications. In this case, one can use an octree structure as a starting point, remove the hanging nodes from the octree and use this as an initial mesh. This has been proposed in [Schneiders 1998], fig. 30 shows a shows part of a mesh that has been generated for the simulation of flow around a car. This technique will be discussed in detail in chapter 5 of this lecture.

Figure 30: Hexahedral element mesh for the simulation of flow around a car



The grid- and octree-based algorithms presented here prove that the superposition

principle is a very useful algorithmic tool to deal with the hex meshing problem. Most commercial hexahedral mesh generators are of this type. . The algorithms mentioned here are not the only variants of the strategy, combinations with the other methods seem promising. Further research may reveal the full potential of superposition methods.

4 The spatial twist continuum

The techniques presented so far can also be used for the generation of tetrahedral element meshes. In contrast to that, the spatial twist continuum is a unique concept for quadrilateral and hexahedral element mesh generation.

Many of the results presented here were achieved by the CUBIT team, a joint research group at SANDIA National Laboratories and Brigham Young University that is in quadrilateral and hexahedral element meshing research since the beginning of the 90's. The group is working on algorithms that generate a mesh starting from discretization of the object surface into quadrilaterals. As part of their research, the paving [Blacker and Stephenson 1991] and plastering [Blacker 1993] advancing-front type mesh generators have been developed. These algorithms will be described in chapter 6, here we will present other results.

Def. 1 *Given an unstructured quadrilateral element mesh $M = (V, E, F)$, the* spatial twist continuum (STC) *[Murdock et al. 1997] $M' = (V', E', F')$ is defined as follows:*

- *For each face $f \in F$, the midpoint v' is a node of V'.*
- *For each edge $e \in E$ we define an edge $e' = (v'_1, v'_2) \in E'$ where v'_1 and v'_2 are the midpoints of the two quadrilaterals that share e.*

For each node $v \in V$ a face $f' \in F'$ is defined by the midpoints of the adjacent quadrilaterals. The STC is the combinatorial dual [Preparata and Shamos 1985] of the quadrilateral mesh, just as a Delaunay triangulation and it's corresponding Voronoi diagram are combinatorical dual.

Figure 31: Quadrilateral mesh and spatial twist continuum

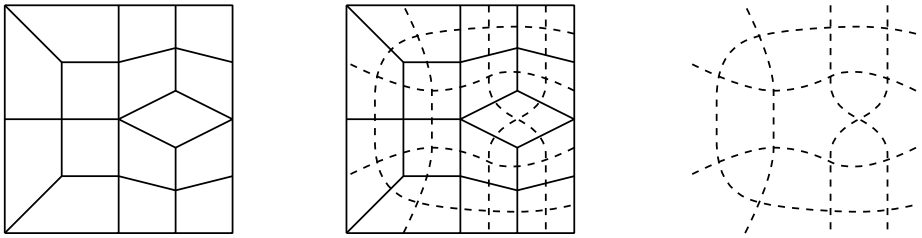


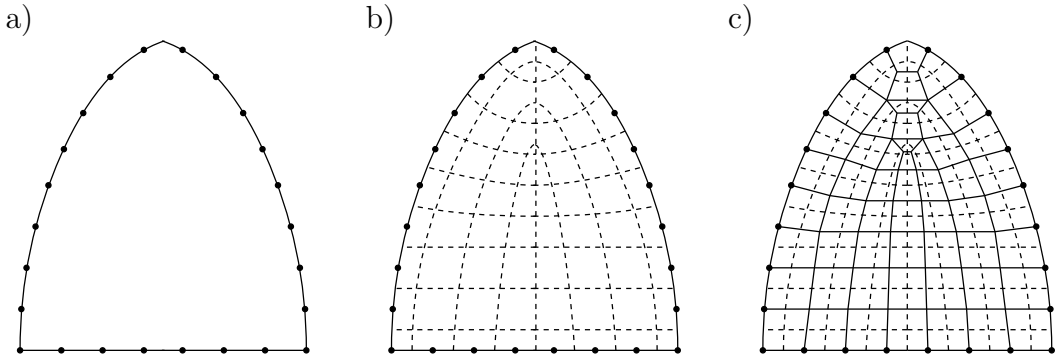
Fig. 31 shows a quadrilateral mesh (straight lines) and the STC (dotted lines). The edges of the STC are displayed not as straight lines but as curves. This allows the recognition of chords, a very important structure: One can start at a node, follow an edge e_1 to the next node, then choose the edge e_2 "straight ahead" that is not adjacent to e_1, continue to the next edge e_3 and so on. The sequence $e_1, e_2, \ldots$ forms a chord (displayed as a smooth curve in fig. 31). Chords can be closed or open curves and can have self-intersections, and no more than 2 chords are allowed to intersect at one point.

A chord corresponds to a row of quadrilaterals in the mesh. The STC is not as flexible a structure as the Voronoi diagram.

By definition, an STC corresponds to a quadrilateral element mesh. Thus, in order to generate a quadrilateral mesh, one can construct an STC by arranging a set of chords and then generate the mesh by constructing the dual.

Fig. 32 shows an example: We want to construct a quadrilateral mesh for a given polygon (fig. 32, it follows from the 2D equivalent of eq. 1 that the number of polygon segments must be even). The polygon is filled with a set of chords such that each polygon segment intersects with one chord. The result is an STC for the input polygon (fig. 32b). The midpoints of the STC faces define the mesh nodes, nodes beloging to neighboring faces are connected and give the quadrilateral mesh (fig. 32c).

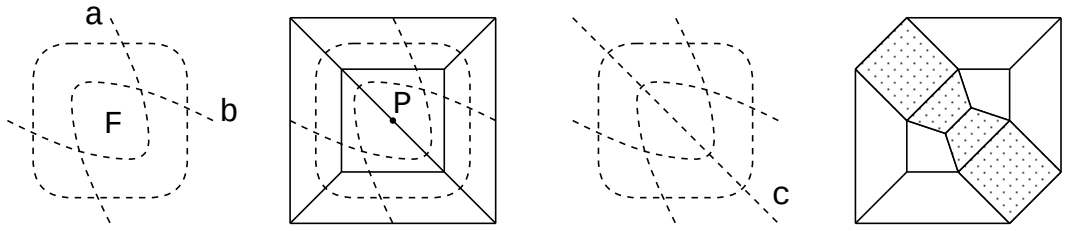
Figure 32: Mesh construction via the STC



The example shows that for quadrilateral mesh generation it makes sense to start with the generation of the dual; the fact the the STC is a more constrained structure than the Voronoi diagram makes it also more useful. The drawback is that STC is not in some way geometrically optimal as the Voronoi diagram resp. the Delaunay triangulation is. The STC is a purely topological concept, geometric quality does not matter; this is still a topic of research.

Not all STC's dualize to a good quadrilateral mesh. Fig. 33 shows an example: The chords a and b intersect twofold, and the STC has a face F with 2 edges. In the corresponding mesh, this faces corresponds to a node P with only two quadrilaterals adjacent; these quadrilaterals share two edges and both have an obtuse angle.

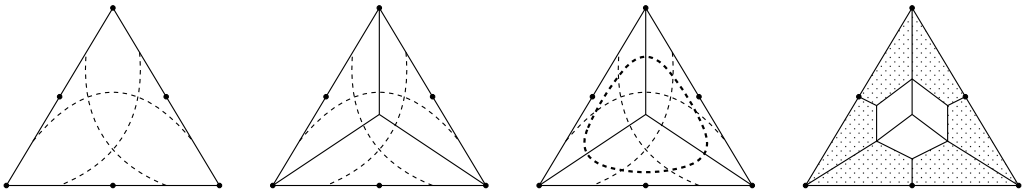
Figure 33: Removing a degeneracy from a quadrilateral mesh



Fortunately, one can overcome the problem by adding an additional chord to the STC. This corresponds to the insertion of an additional row of elements into the mesh, and the degeneracy is removed.

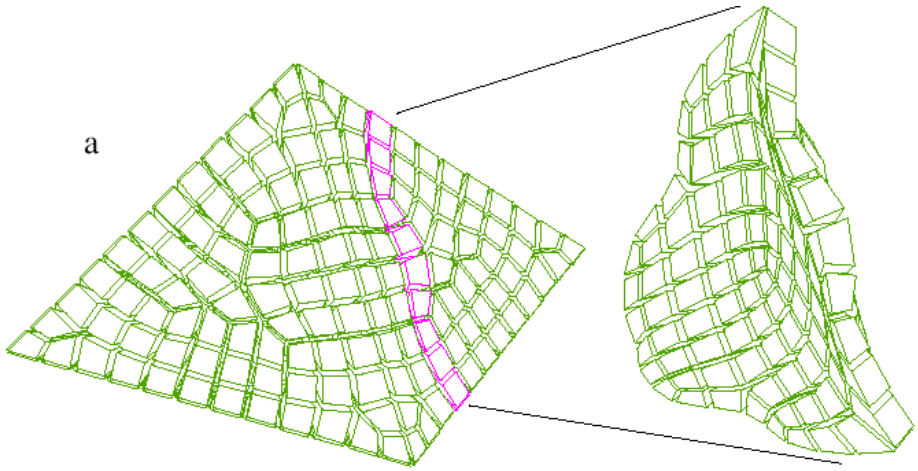
Another kind of degeneracy is shown in fig. 34, where each node of the STC is connected to 2 boundary segments. This leads to the construction of quadrilaterals that touch the boundary at 2 edges, leading to an obtuse angle. Again the solution is to insert a buffer layer at the boundary, which is equivalent to the insertion of a buffer layer of quadrilaterals.

Figure 34: Degeneracy at boundary



The STC is a very useful tool to analyze and improve mesh generation algorithms, and the good thing is that it generalizes for the 3D case! This was noticed by the CUBIT team and lead to important theoretical results [Mitchell 1996]. A 3D analogon to the quadrilateral meshing algorithm was presented in [Müller-Hannemann 1999], and in the CUBIT project the whisker weaving algorithm was developed. These points will now be discussed in detail.

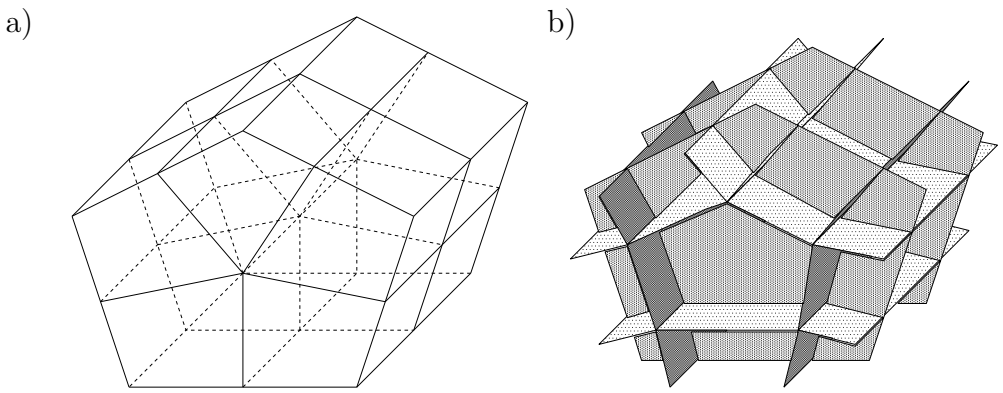
Figure 35: Hexahedral sheet [Calvo2000]



In 3D, the STC is the combinatorial dual of a hexahedral element mesh (fig. 36):

- The midpoints v of the hexahedra are the nodes of the STC.
- For each pair of adjacent hexahedra, their midnodes define an edge $e = (v_1, v_2)$ in the STC.
- For an edge in the hexahedral mesh, consider all hexahedra that share that edge. Their midnodes $v_1, \ldots, v_n$ define a face in the STC.
- A row of hexahedral elements corresponds to a chain of edges in the STC (a chord); in the example of fig. 36 all chords start and end at the mesh boundary, but there may also be cyclic chords in an STC.

Figure 36: Hexahedral mesh and STC



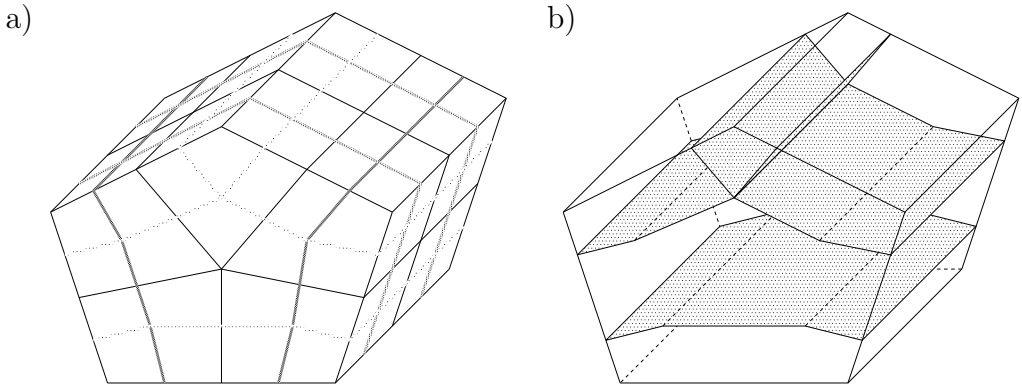
- A layer of hexahedral elements corresponds to a sheet of faces in the STC (fig. 35).

A chord is defined by the intersection of two sheets or by a self-intersecting sheet. Vertices of the STC that correspond to hexahedra are defined by the intersection three sheets (or less in case of self-intersections). In the example of fig. 36, a hexahedral mesh with 10 elements corresponds to an STC with 7 sheets. Note that the intersection of the STC with the surface of the mesh is itself an 2D STC on the surface (fig. 37a).

The STC is a set of intersecting sheets, and if no more than 2 sheets intersect at a given point, and if no tangential intersections are present, "dualizing" the STC gives a hexahedral mesh. It is clear that it is difficult to apply a local change to a hexahedral mesh, since that is equivalent to a modification of the STC. The only operations allowed for an STC are the insertion or deletion of a sheet, and both will likely have a global effect.

The ideas of the 2D dual-meshing algorithm can be extended to the 3D case. Most of the 3D algorithm is due to [Müller-Hannemann 1999], some interesting ideas can be found in [Calvo2000].

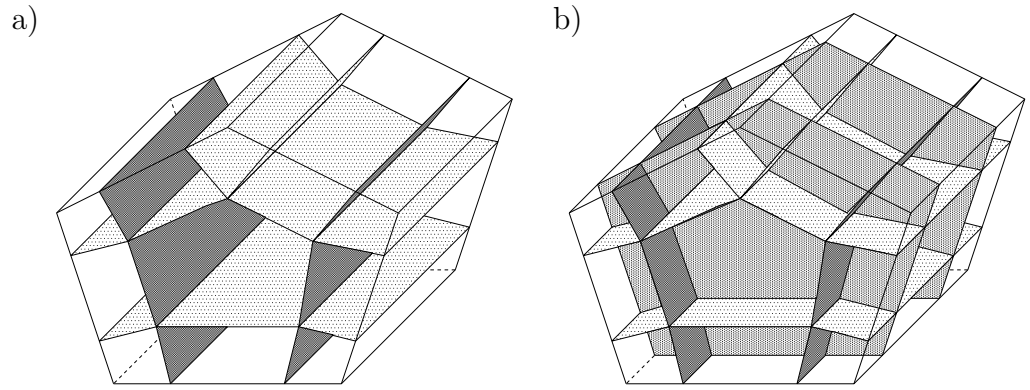
Figure 37: Construction of the STC



Starting point is a quadrilateral surface mesh with an even number of faces (fig. 37a), from which the surface STC is derived. The challenge is to find a 3D STC whose intersection with the surface mesh matches the surface STC. For each chord, a sheet is inserted into the 3D STC. Fig. 37 and fig. 38 show the process: 3 sheets are used to initialize

the 3D STC; the intersections of the sheets with the surface correspond to 3 chords of the surface STC. Then 2 vertical sheets matching 2 surface chords are added, followed by the sheets for the remaining 2 chords. Dualizing the 3D STC gives the hexahedral mesh (fig. 36a).

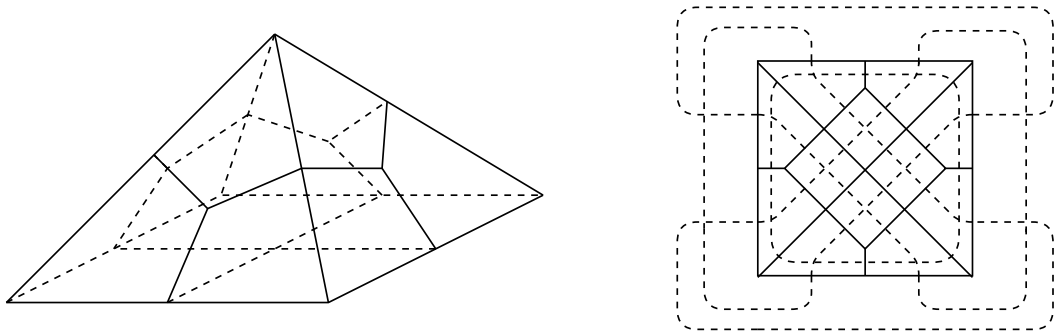
Figure 38: Construction of the STC continued



In [Mitchell 1996], it is proved that, given a surface mesh with an even number of elements, a hexahedral element exists. Mitchell shows that an STC that respects all constraints can be generated by inserting sheets into the original STC. In 3D, there are more invalid configurations (STC cells with 2 faces, ...) than in 2D (fig. 33, fig. 34), all of them can be removed by inserting sheets that do not intersect the surface.

The proof is non-constructive, and the problem remains to find a good arrangement of sheets which allows for the derivation of a quality hexahedral mesh. For many surface STC's, it is very difficult to find a matching 3D STC. Fig. 39 shows an example: The surface mesh of the pyramid corresponds to an STC of two chords, one of them with 8 self-intersections! It is nearly impossible to construct a sheet for this chord.

Figure 39: Complex surface STC



In [Müller-Hannemann 1999] an algorithm is presented that can deal with such problems. It starts from a surface mesh and follows the lines outlined above, but modifies the surface mesh in case of problems. The key idea of his algorithm is to remove self-intersections of chords in the surface STC (fig. 40a): A self-intersecting chord is doubled, so that there are 4 intersections present, 2 of them self-intersections (fig. 40b). The pattern at the intersections points is modified (fig. 40c), and the resulting STC has no self-intersections.

Figure 40: Removal of self intersections

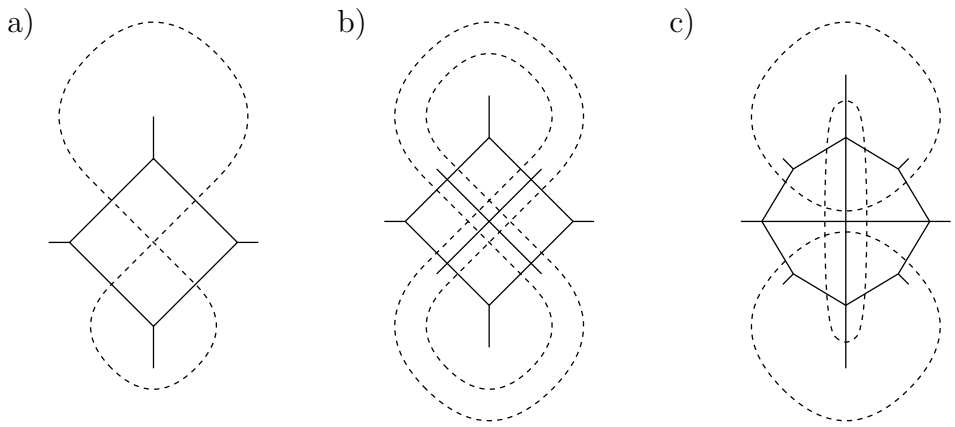
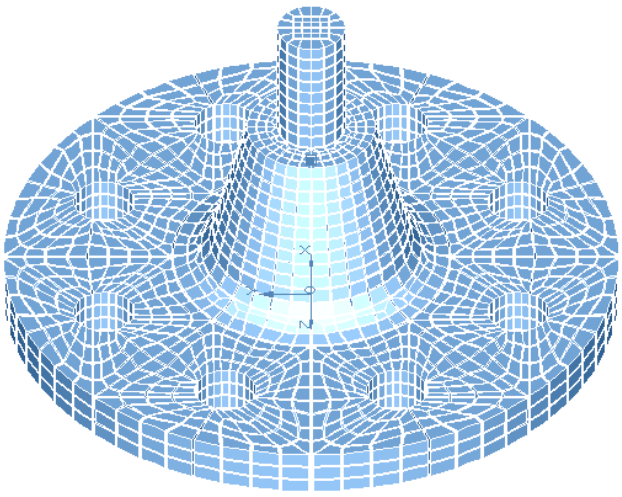


Fig. 40 also shows how the quadrilateral surface mesh is modified (black lines). Due to the doubling of the chord, there is a node with 4 faces adjacent in the mesh. The structure is modified by rotating the pattern, the neighboring elements are not changed.

After removing the self-intersections from the surface STC, for each chord a sheet must be added to the 3D STC. Therefore, a perfect elimination scheme is presented, i.e. a sequence of insertions that leads to a good mesh. This is the crucial step of the algorithm, it is not guaranteed that such a scheme exists. The proof that no degeneracies are present in the mesh uses concepts from graph theory and topology, for the details see [Müller-Hannemann 1999].

Fig. 41 shows a mesh for the model of a shaft with a flange. The algorithm does not deal with object with holes, so the model was decomposed into simpler objects. No self-intersecting chords were present in the surface STC's of the subregions, and a perfect elimination scheme was found for each, leading to a quality hexahedral mesh.

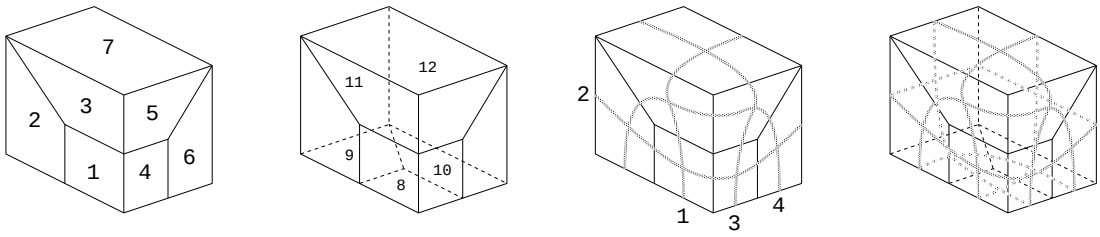
Figure 41: Hexahedral mesh of a shaft with flange



For the rest of the chapter, we will focus on another STC-based scheme, the whisker-

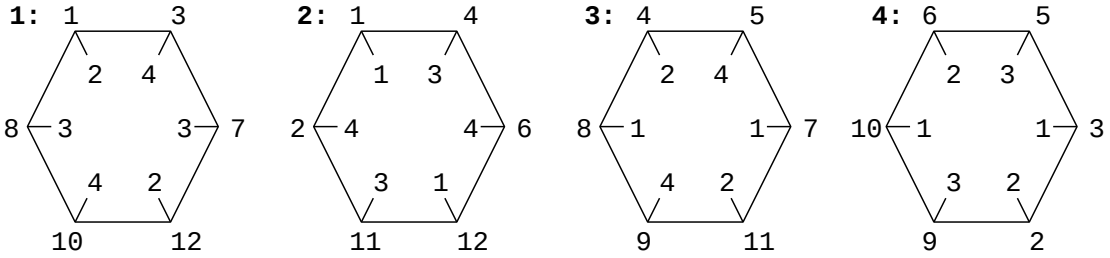
weaving algorithm. It was developed in the CUBIT project, and it attempts to find a hexahedral mesh for a surface mesh which. It will be explained for the example of fig. 42.

Figure 42: STC of a surface mesh



The first step of the algorithm is the initialization of *whisker sheet diagrams*. A whisker sheet corresponds to a sheet of the STC to be constructed, so there is one whisker sheet for each loop. Fig. 43 shows the whisker sheets for the loops of fig. 42: The vertices of a sheet correspond the faces that the loop intersects, and are labeled outside by the face numbers. Since the faces also correspond to the intersection of two loops, the vertices are labeled inside with the number of the intersecting loop (whisker sheet).

Figure 43: Initial set of whisker sheets



The next step in whisker weaving is the formation of a hex by crossing three chords on three sheets. Two sheets correspond to two chords on the third sheet, and it is required that the chords start at adjacent faces. The chords are pairwise crossed and define three vertices which correspond to the same STC vertex (hexahedron).

In the example of fig. 43 the sheets 1, 2 and 3 have been selected, fig. 44a shows the result. By duality, this step is equivalent to the construction of a hexahedron at the faces 1, 4 and 8 (fig. 45a).

Next the sheets 2, 3 and 4 that correspond to the chords starting at the faces 2, 9 and 11 are selected. The result is shown in fig. 44b and is equivalent to the constrution of another hexahedron (fig. 45b). Obviously glueing the hexahedra is the next step (fig. 44c), which is equivalent to joining the chords 2 and 3 in the sheets 2 and 3 (fig. 44c).

In the following the chords corresponding to the faces 3, 5, 7 and 6, 10, 12 are joined. The glueing operation completes the construction of the whisker sheet (fig. 46). Having dualized the STC, one gets the mesh shown in fig. 42d.

The dualizing process does not always result in a valid hex mesh: Hexahedra with more than two faces in common may be present in the mesh or invalid elements may be constructed if their base faces are nearly coplanar. Mitchell [Mitchell 1996] identifies 7

Figure 44: Whisker weaving: Example

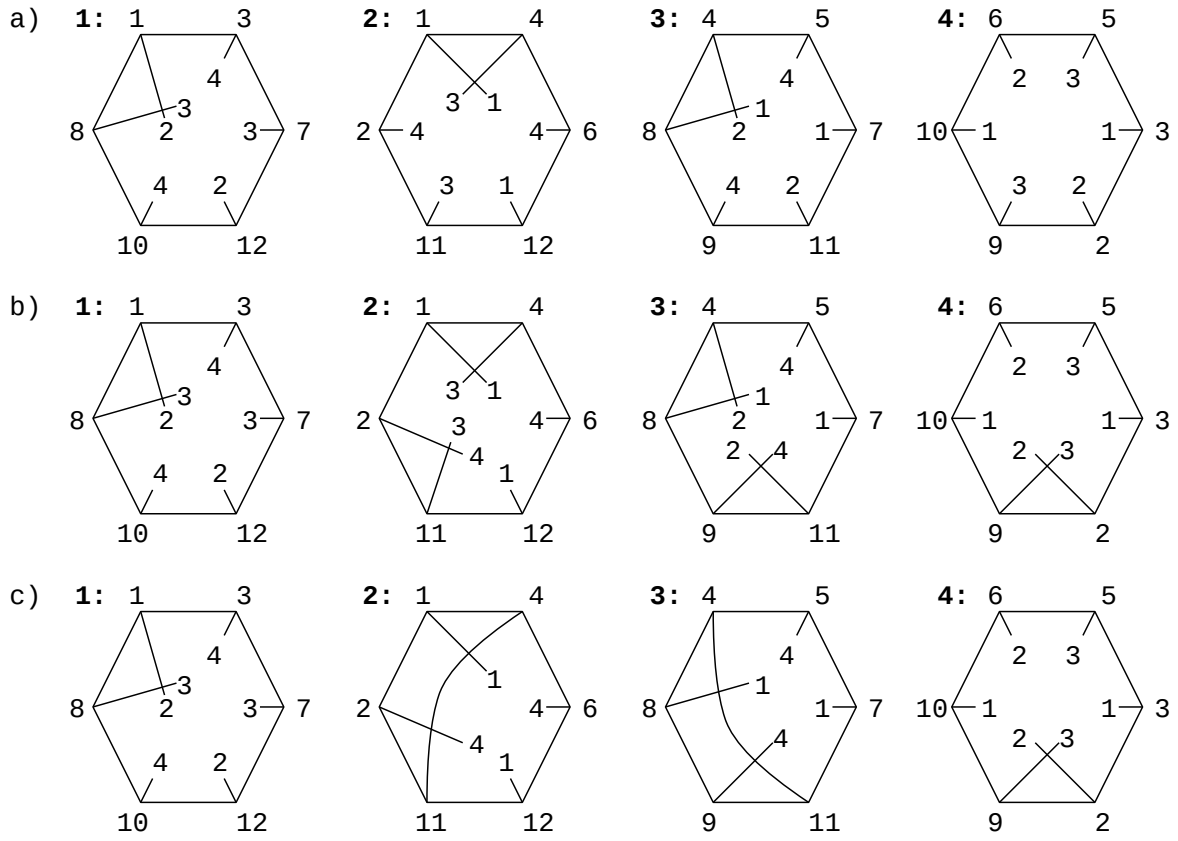
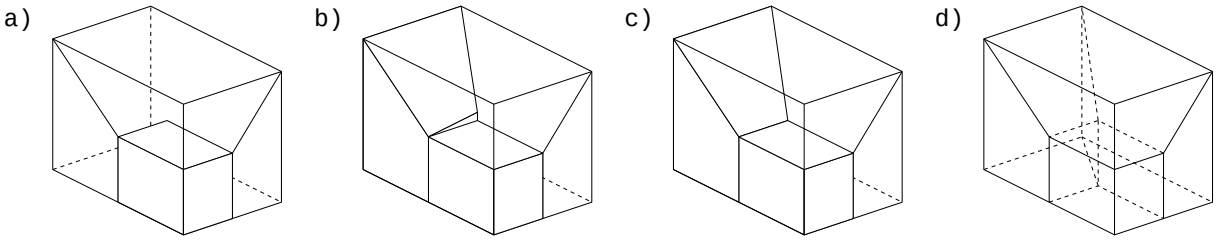


Figure 45: Dualized view of whisker weaving

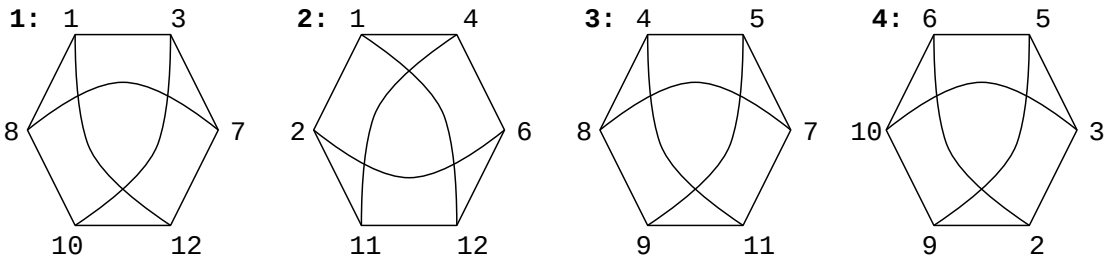


constraints an STC must fulfil in order to guarantee that dualizing results in a valid hex mesh. This is done by inserting additional sheets into the STC, see [Mitchell 1996] for the details.

It proved to be very difficult to derive a stable version of the whisker weaving algorithm. If the surface STC has self intersections, it may be nearly impossible (the STC corresponding to the surface in fig. 2 consists of two loops, one of them with 8 self-intersections – whisker weaving done "by hand" is very difficult). This is probably due to the fact that the algorithm is quite indeterministic and relies more on topological than on geometrical information.

Compared to the methods that use the STC, , block-decomposition and superposition methods are easier to realize, since they are not constrained by a given surface discretization. Algorithmic complexity is the prize one has to pay for the potential benefit of the STC approach, and it is still a subject of research. Nevertheless, the concepts presented

Figure 46: Resulting whisker sheets



in this chapter give lots of insight in the nature of hexahedral mesh generation, and the techniques are useful in enhancing block-decompostion of superposition type algorithms (the templates in fig. 58 where constructed by using the STC concept).

5 Mesh refinement, octree-based mesh generation

Most algorithms for triangular and tetrahedral mesh generation can be used for mesh refinement (e.g. by point insertion). This does not hold for quadrilateral and hexahedral mesh, and the example of fig. 1 shows that especially hexahedral mesh refinement is a very difficult task. In the following we discuss the mesh refinement problem for arbitrary quadrilateral/hexahedral meshes and consider the related problem of quadtree/octree-based mesh generation.

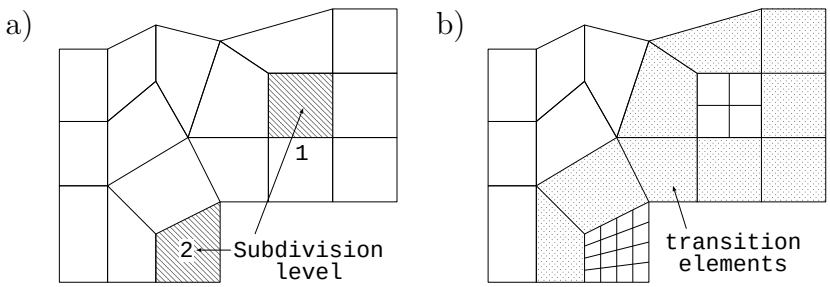
5.1 Formal statement of the refinement problem

Let M be a mesh of quadrilateral (hexahedral) elements, constituted by the set V of its nodes and the set H of its elements. Other entities of interest are the set E of mesh edges and the set F of faces (for the 3D case).

Def. 2 *A quadrilateral (hexahedral) element mesh is* conforming *if two distinct, non-disjoint elements intersect at common nodes or edges (or faces) only.*

Def. 3 *A quadrilateral (hexahedral) element mesh is* structured *if each interior node v belongs to 4 quadrilateral elements (8 hexahedral elements); otherwise it is called* unstructured.

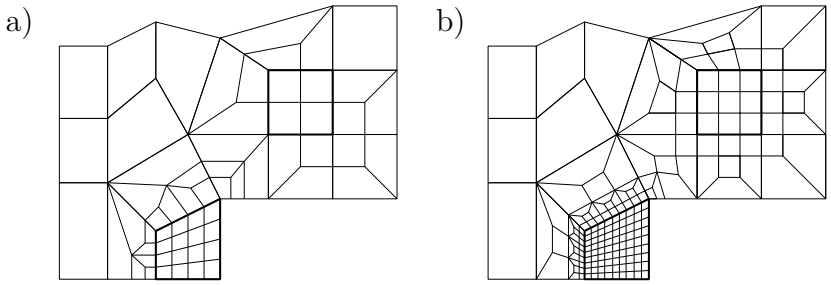
Figure 47: Subdivision level assignement



In the following we consider the mesh refinement problem for unstructured conforming meshes. We start with a problem specification for the 2D case: Given a mesh M, the elements f of M are assigned subdivision levels $S(f)$ that indicate that f must be split into $4^{S(f)}$ quadrilaterals (fig. 47a).

The refinement levels can be obtained by estimating the numerical error: If an element h of size h has to be replaced by elements of size $h' < h$, f is assigned the subdivision level $\lceil \log_2 \frac{h}{h'} \rceil$. The resulting mesh is not conforming; therefore one must refine the neighbor elements appropriately (fig. 48a).

Figure 48: Refined meshes



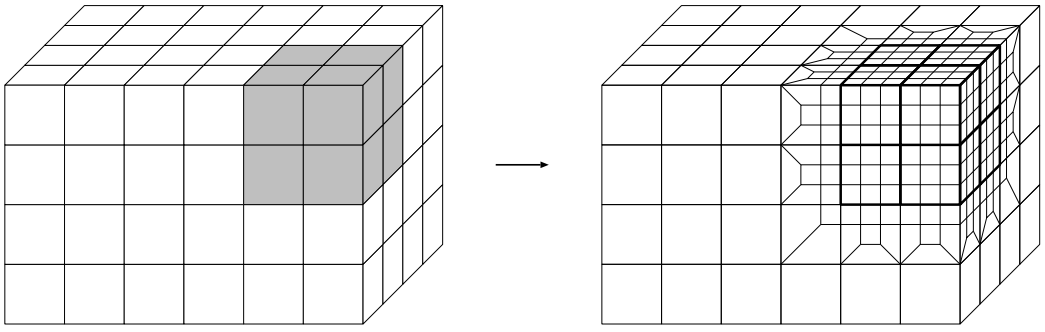
Because of the choice $b = 2$ for the basis this technique is called 2-refinement. Setting $b = 3$, one gets a 3-refinement (fig. 48b): $f \in H$ is split up into $9^{S(f)}$ quadrilaterals.

In the same way, 2- and 3-refinement can be defined for the 3D case. Subdivision levels $S(h)$ can be assigned to the elements h of the coarse mesh M, which means that

- $h \in H$ is split up into $8^{S(h)}$ elements (2-refinement)
- $h \in H$ is split up into $27^{S(h)}$ elements (3-refinement)

An example of 3-refinement is shown in fig. 49 (the marked elements have refinement level 1).

Figure 49: 3-Refinement of a hexahedral element mesh

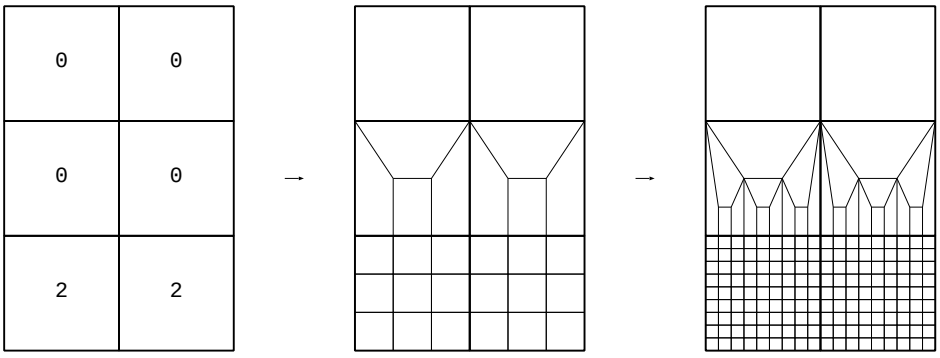


5.2 Mesh refinement by template insertion

The task in quadrilateral/hexahedral mesh refinement is to find a transitioning from the refined part to the coarse part of the mesh such that the resulting mesh is conforming (fig. 47b→fig. 48a). This is done by inserting appropriate templates into the transitioning

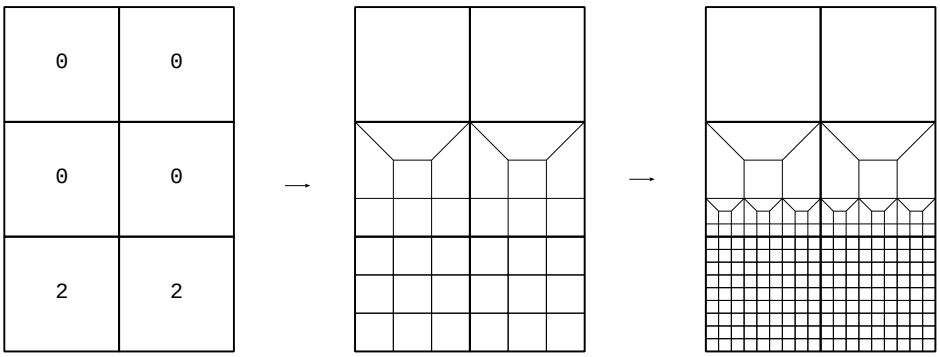
elements. Fig. 51 demonstrates the idea: The lower element has subdivision level 2, it's neighbor level 0. In the first step the lower face is split into 9 faces, the upper face is split into 4 (the mesh remains conforming). In the second step the same split is done for the quadratic elements, the adjacent faces are split recursively.

Figure 50: Unstable transition



As fig. 50 shows, the result is not satisfactory, since ill-shaped elements can be found in the mesh. In the example of fig. 51, another template is chosen for the transitioning element. The elements that are split in the second step are all quadratic, so that no angle $\alpha < 45^{\circ}$ can be found in the mesh.

Figure 51: Stable transition



This motivates the following definition:

Def. 4 *A refinement is stable if the minimum angle of the refined mesh does not depend on the subdivision level S:*

$$\alpha_{min} \geq \gamma > 0 \qquad S \to \infty \tag{2}$$

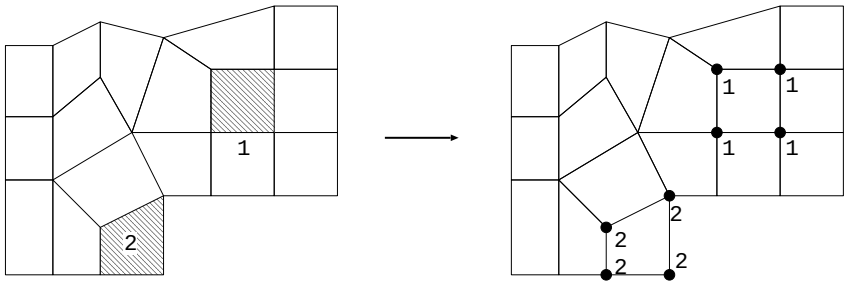
The refinement in fig. 51 is stable, the one in fig. 50 is not.

In the following we present an algorithm that guarantees a stable refinement of quadrilateral meshes. For ease of explanation, we will do that for 3-refinement, the extension for 2-refinement is indicated. Then we will extend the 3-refinement algorithm for the 3D case and give a negative result for 2-refinement.

5.3 3-Refinement

The way a transition element is refined depends on the subdivision levels of its neighboring elements. To achieve a conforming refinement, the subdivision levels are propagated to the mesh nodes (fig. 52).

Figure 52: Defining nodal subdivision levels



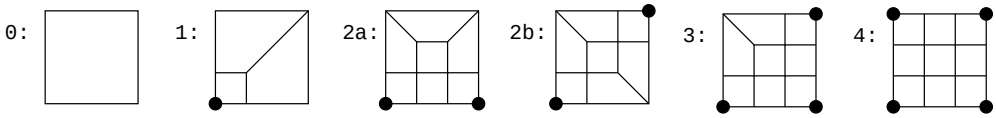
Def. 5 *The subdivision level of a node $v \in V$ is defined as the maximum subdivision level of its n adjacent elements:*

$$S(v) = \max_{i=1,n} S(f_i) \tag{3}$$

A node v with $S(v) > 0$ is called marked.

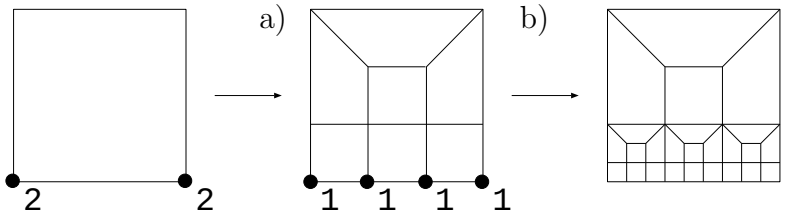
It turns out that the nodal subdivision levels uniquely define how to refine an element, and that element refinement can be done in a purely local manner. Consider an element $f \in M$ and its marked nodes: If one only considers whether a node is marked or not, six combinations exist (fig. 53).

Figure 53: Templates for element refinement



An element is then split by selecting the appropriate template from fig. 53. The nodal subdivision levels are decreased by 1, and appropriate refinement levels of the newly introduced nodes are defined (fig. 54a gives an example). The procedure is recursively done for the newly generated elements.

Figure 54: Example of element refinement



The new subdivision levels S' are determined with the help of auxiliary subdivision levels of the element $f = (v_1, v_2, v_3, v_4)$ and its edges $e_i, i = 1, 4$:

$$\begin{aligned} S'(v_i) &= \max\{v_i - 1, 0\}, \quad i = 1, 4 \\ S'(e_i) &= \max\{S'(v_a), S'(v_b)\}, \quad e = (v_a, v_b) \\ S'(f) &= \max_{i=1,4} S'(v_i) \end{aligned}$$

The subdivision levels of the newly introduced nodes v are defined as follows:

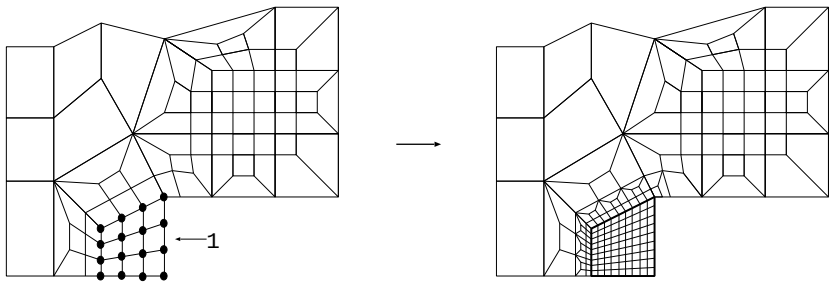
- $S'(v) = S'(e)$ if v is inserted on the edge e
- $S'(v) = S'(f)$ if v lies in the interior of f.

In the example of fig. 54a the subdivision levels of the interior nodes are set to zero while the nodes on the basic edge are assigned the level 1. The procedure is then applied recursively to the newly generated elements (fig. 54b). Formally the mesh refinement algorithm can be stated as follows:

```
procedure refine ( element )
   Select appropriate template;
   Refine element;
   Update nodal subdivision levels;
   forall newly generated elements elenew
      refine ( elenew );

procedure mesh_refinement
   Define nodal subdivision levels;
   forall elements
      refine ( element );
```

Figure 55: 3-Refinement: Example



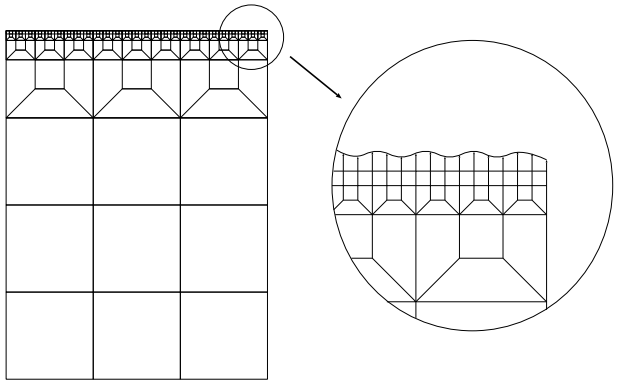
It remains to prove that the refined meshes are conforming and that the refinement is stable. Note that

- an edge with 2 marked nodes is split into 3 edges
- an edge with 1 marked node is split into 2
- an edge with no edge marked is not split.

These rules, together with the rules for updating the nodal subdivision levels, uniquely define how to split up an edge. The number of new points that are inserted on an edge is determined by its nodal subdivision levels only, and thus the refinement process results in a conforming mesh. Stability is ensured by the fact that the recursively refined elements in fig. 53 that have at least one node marked are self-similar to their "parent" element.

Fig. 55 shows how the algorithm proceeds for the example mesh. An application of the refinement algorithm is shown in fig. 56a where a rectangular shape whose upper side has a very fine structure has to bo meshed. This was done by creating a coarse mesh and a assigning the nodes on the upper side refinement level 3. The nodes were then projected onto the exact contour (fig. 56b).

Figure 56: 2D-modeling of fine boundary structures



It is possible to derive a similar algorithm for the 3D case. Subdivision levels are computed for the nodes, and elements with marked nodes are split by recurisively inserting templates.

First, nodal subdivision levels are defined as the maximum subdivision levels of the adjacent elements:

$$S(v) = \max S(h_i) \tag{4}$$

The problem in is to find an appropriate set of 3D templates. 22 classes of marker assignements to hexahedron nodes must be considered; for each, an appropriate template must be found. The idea is to use the 2D template to split the hexahedron faces (fig. 53), so that neighboring elements are split in the same way at common faces. This guarantees that the resulting mesh is conforming. Fig. 57 shows the surface tesselations for a selection.

It remains to find 3D templates whose surfaces match the tesselations defined in fig. 57. Fig. 58 shows some examples:

- 3D template 1 is split according to 2D template 1.
- The splitting for 3D template 2 is more complicated (a U-shaped sweep around the marked edge).
- The splitting for template 8 (4 marked nodes) is very complicated. It is done in 2 steps: First, the marked face is split into 3 strips by splitting the element into four. Then, the 3 newly created elements are split in the same way, but in the other direction. Finally, a buffer layer is created on top.

Figure 57: 3D-templates

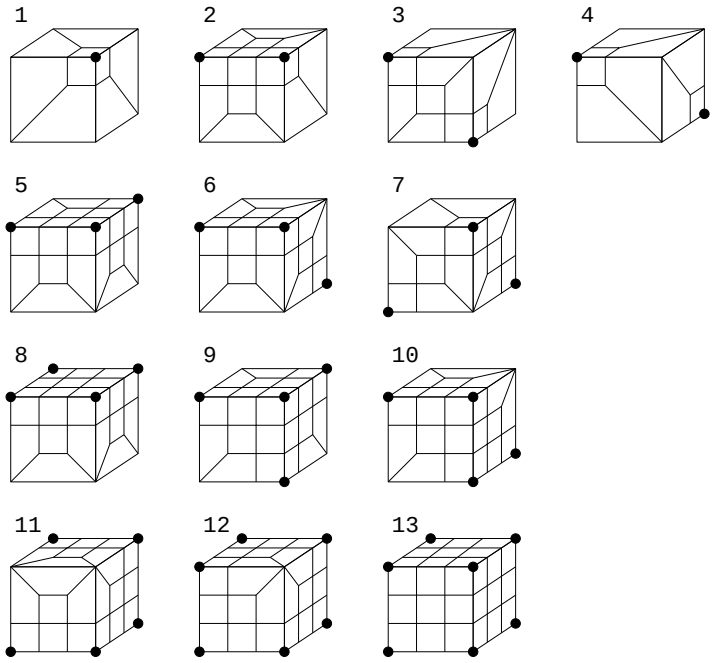
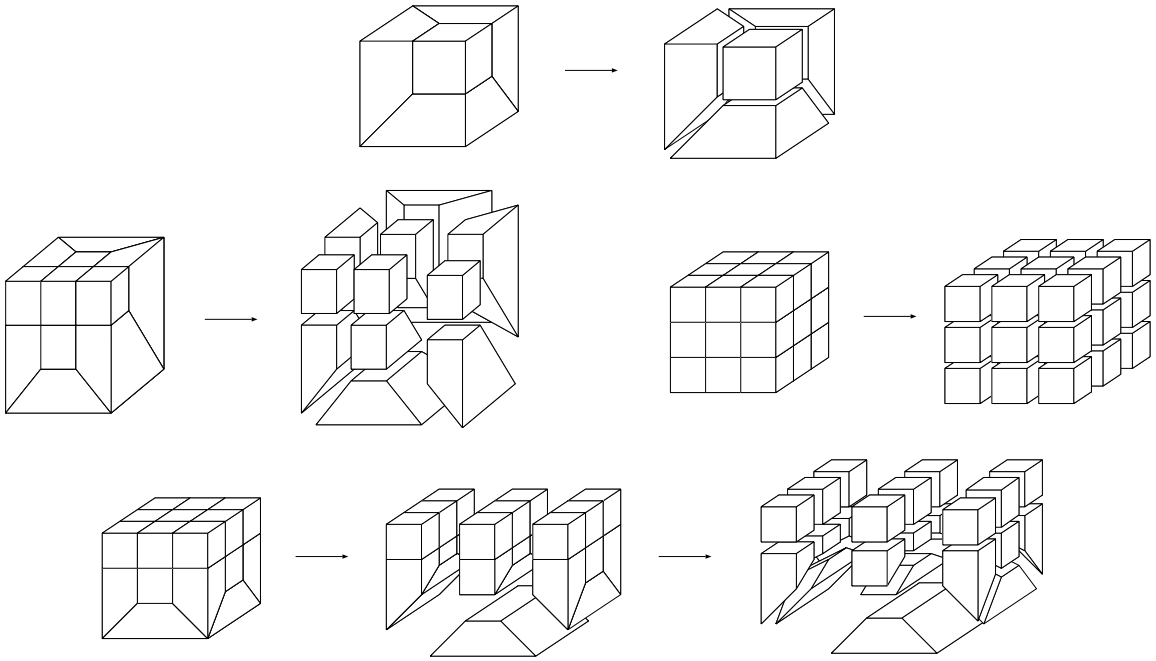


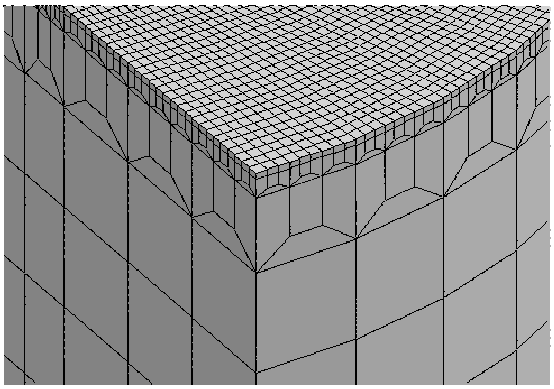
Figure 58: 3D templates - details



The newly created elements at the marked nodes are self-similar to their "parent" elements. Since these are the only elements that might be split recursively, the refinement is stable.

A similar splitting procedure can be done for most of the 22 3D templates. Unfortunately, it is not general. Consider template 12 in fig. 57: The surface tesselation consists of 51 quadrilateral faces. This violates equation (1) which states that a hexahedral mesh must have an even number of boundary faces. Thus, no 3D tesselation for template 12 exists, and the set of 3D templates from fig. 57 is not complete.

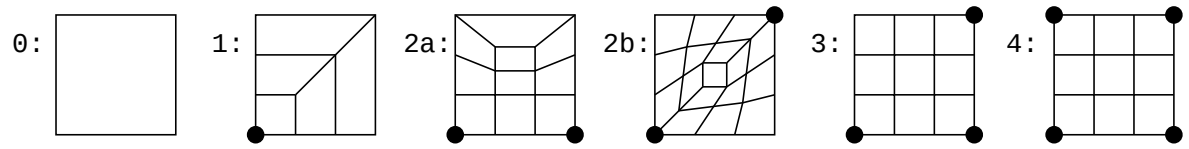
Figure 59: Example of 3D refinement



In practice, many refinement problems are "convex", and the templates defined in fig. 58 can be used for that case. Fig. 59 shows an example where small structures on the top face of a cylinder have been modeled by a refined hexahedral mesh.

The problem can be solved by selecting another set of 2D templates. Fig. 60 shows the solution: Templates 1, 2a, 2b and 3 have been changed by inserting a new buffer layer of elements. Note that an edge is split into 3 parts if at least one of it's nodes is marked.

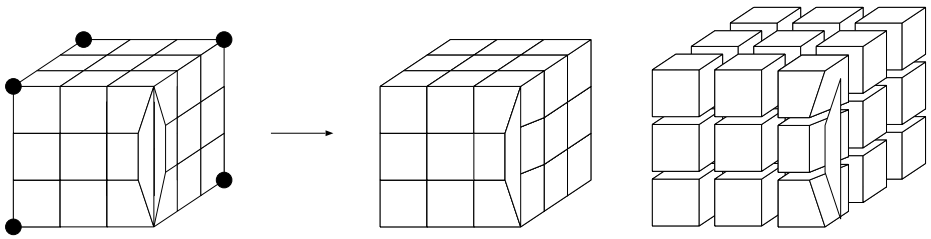
Figure 60: Modified 2D templates



The fact that template 3 template 3 and 4 are identical simplifies the construction of 3D templates. As an example consider the 3D template with 6 marked nodes (fig. 61). This template is very difficult to find for the original template set, but a simple solution exists for the new. The 2 trapezoidal faces on the surface become a dangling face in the 3D tesselation, the remaining part is equivalent to the template with 8 marked nodes for which a natural split exists.

The critical template with 7 nodes is equivalent to the template with a marked nodes and is split into 27 hexahedra.

Figure 61: Hexahedral mesh for templated with 6 nodes marked

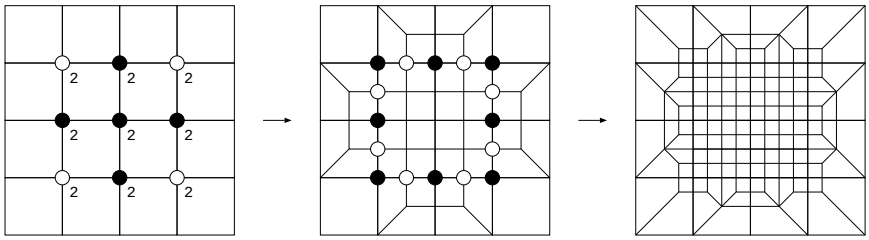


5.4 2-Refinement

2-Refinement (fig. 48a) is better than 3-refinement since the resulting element sizes are likely more closed to the maximum sizes allowed. Usually the refined meshes can be expected to have fewer elements. It is, however, more difficult to give an algorithm for 2-refinement, especially in the 3D-case.

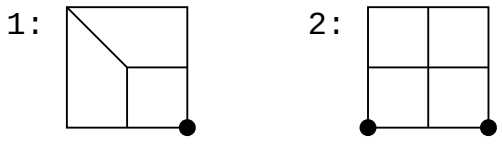
The simplicity of the 3-refinement algorithm is due to the fact that the transition could be done locally. This does not hold for 2-refinement where one must consider pairs of adjacent elements (fig. 62).

Figure 62: 2-Refinement



2-Refinement can be achieved with a modified version of algorithm 1. The mesh nodes are assigned nodal subdivision levels as usual. In order to determine the element pairs in the transition zone, one has to consider the sequence $\{v_1, v_2, \ldots, v_{2k-1}, v_{2k}\}$ on the boundary of the submesh of M that must be refined (note that the number of boundary nodes of a quadrilateral element mesh is even). The nodes $v_{2i}, 1 \leq i \leq k$, are deactivated, and the remaining active notes determine the way an element is split up: If an element has one active marked node, template 1 (fig. 63) is inserted, if it has more than one active node, template 2 is used. All boundary nodes of M are set active while the newly inserted nodes remain inactive. This procedure is recursively applied to the newly generated elements.

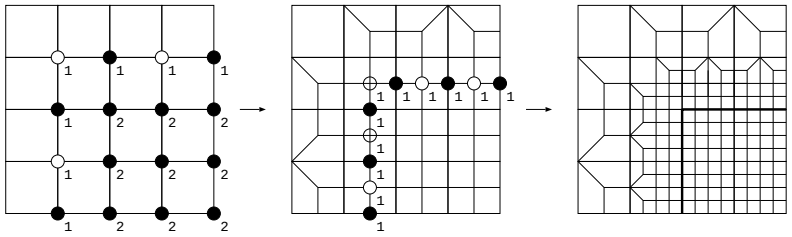
Figure 63: Templates for 2-refinement of structured quadrilateral element meshes



Because of the choice of the templates and the activation strategy one can easily show that the refinement process is stable and results in a conforming mesh. However, sometimes elements with sidelength differing by a factor 4 are generated in the transition

zone (fig. 62). One can avoid this problem if one uses two layers of M for transitioning (fig. 64).

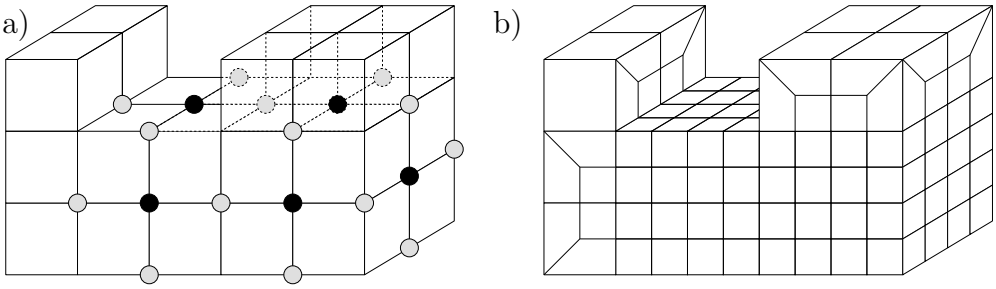
Figure 64: 2-Refinement (two transition layers)



One can easily generalize the 2-refinement algorithm for unstructured quadrilateral element meshes. Unfortunately this is not possible in 3D where a general solution exists only for structured meshes. We will discuss this in the context of octree-based mesh generation, here we will take a short look at the prolem.

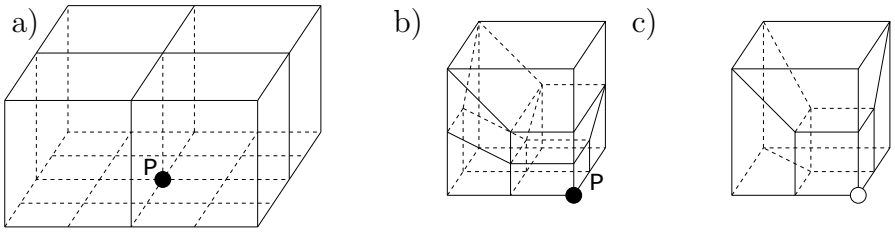
Fig. 65 shows a subdivision level assignement for a structured hexahedral element mesh (all marked nodes are assigned the subdivision level 1).

Figure 65: 2-Refinement of structured hexahedral element meshes



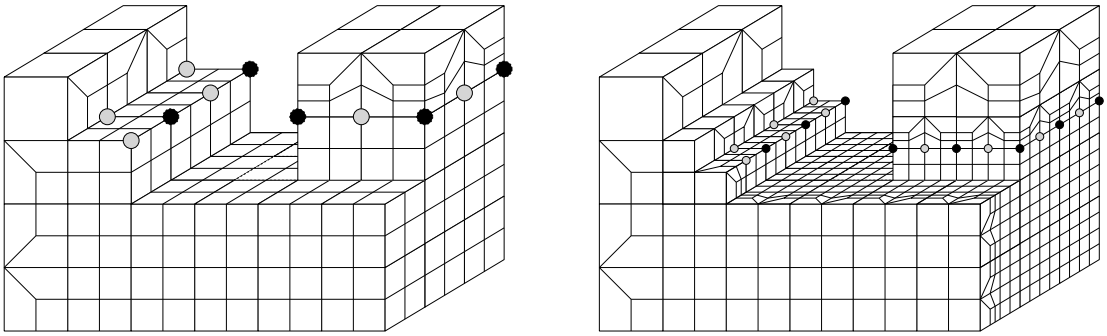
In the 3D-case one must find templates for the transition at faces and edges of the submesh to be refined (fig. 65a). At faces the transition can only be achieved by considering four elements that are adjacent to a node P (fig. 66a). Fig. 66b shows how each of the four elements is split up.

Figure 66: Templates for 2-refinement of structured hexahedral element meshes



Mesh refinement with refinement levels $S > 1$ can only be realized with the help of two transition layers. The algorithm is a straightforward-generalization of the 2D-algorithm, fig. 67 shows an example of how it works.

Figure 67: 2-Refinement of structured meshes



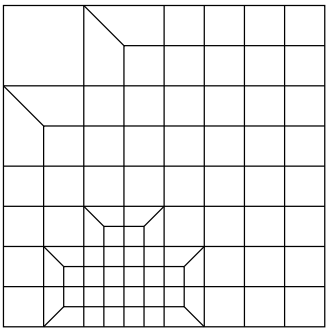
5.5 Formal statement – quadtree/octree based meshing

Quadtrees and octrees are useful structures to faciliate the storage and retrieval of spatial data. A quadtree is a hierarchical decomposition of a rectangular box that is constructed as follows (fig. 68a):

1. The initial rectangle that contains the geometry to be meshed defines the root quadrant.
2. The root quadrant is split into 4 smaller quadrants (the children).
3. The children are split recursively.

Figure 68: Quadtree decomposition

a) b)



To use the quadtree structure as initial mesh for quadrilateral meshing, one must get rid of the hanging nodes, thereby avoiding to add too many new elements. The resulting strucure is called *conforming hull*; fig. 68b shows a solution for the example.

In the refinement process, each edge is split in half; therefore, in analogy to the mesh refinement procedure, this kind of quadtree is called 2-quadtree.

As in mesh refinement, finding the convex hull for a 2D structure is a challenging task. For ease of explanation, we consider the problem for a 3-quadtree, which is constructred in the same way as the 2-quadtree, with the exception that each quadrant is split into 9 subquadrants. Fig. 69 demonstrates the idea,

Obviously, the problem of how to find the convex hull is very similar to the mesh refinement problem. In the following we will derive an algorithm for the construction of

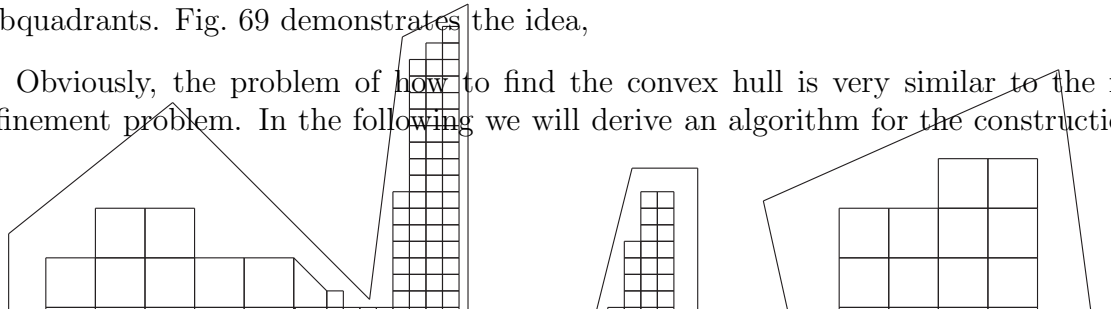
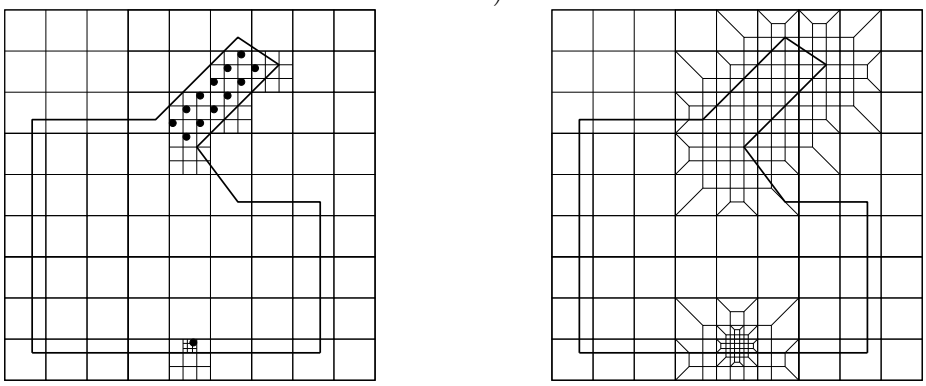


Figure 69: Quadtree decomposition – 3-quadtree

a) b)



the convex hull for 2- and 3-quadtrees and for 3-octrees. For ease of explanation, we will start with the 3-quadtree-algorithm; we will also outline how the mesh can be adapted to the boundary.

5.6 Mesh generation based on 3-quadtrees and 3-octrees

Given a geometry defined by a polygonal chain, a 3-quadtree decomposition of the geometry is constructed. The problem of how to find an octree that is as coarse as possible, but sufficiently fine to represent the geometry will not be discussed here. It is equivalent to a feature recognition problem, and scanline algorithms or adaptive Delaunay triangulation can be used for the solution.

The following definitions help to derive an algorithm for the construction of the conforming hull:

Def. 6 *(i) The* level $l(q)$ *of a quadrant q is defined as follows:*

$$\begin{aligned} l(q) &= 0 && q \text{ root quadrant} \\ l(q_s) &= l(q)+1 && q_s \text{ is a son of } q \end{aligned}$$

(ii) A quadrant without children is called leaf quadrant.

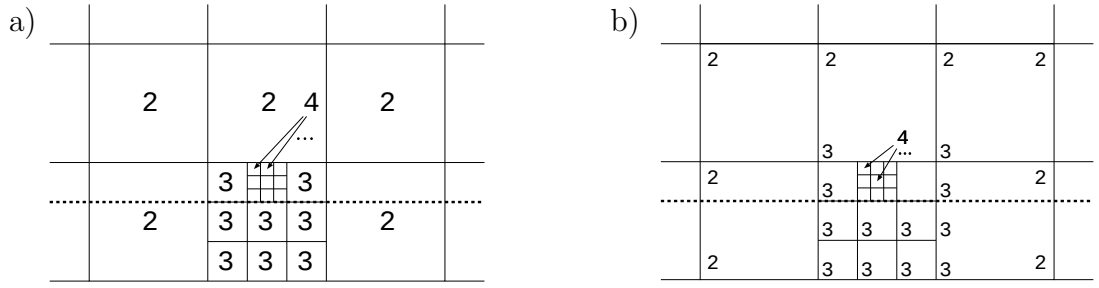
Fig. 70a shows a part of the quadtree and the quadrant levels. There are quadrants with hanging nodes at one or more edges if the levels of neighboring quadrants differ. These quadrants must be split up in order to get a conforming mesh.

First, the level information is transferred to the nodes of the quadtree. A node v is assigned the maximum level of it's adjacent quadrants (fig. 70b):

$$l(v) = \max\{l(q) |\ v \text{ is a node of } q\} \tag{5}$$

One can now use an algorithm similar to the mesh refinement algorithm to remove the hanging nodes. Starting from the quadrants with level 1, the hanging nodes are successively removed ty template insertion. Given a quadrant q of level $l(q)$, all quadrant nodes with level $l(v) > l(q)$ are marked. Then the appropriate template out of the list in fig. 53 is chosen and inserted into the quadrant. The newly generated nodes and faces get the level $l(q)+1$. This procedure is done for all quadrants of level $l, l = 0, \ldots, n$. Formally:

Figure 70: Quadrant and node levels

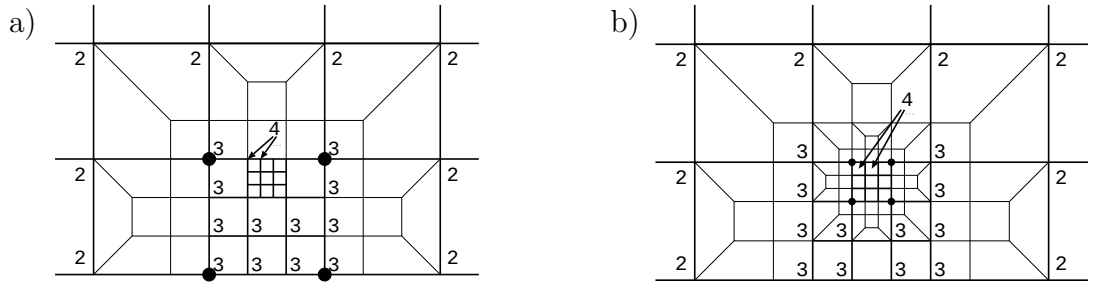



```
procedure conforming_closure

for l = 0 to maximum_level
   for all quadrants q with level l
      mark nodes v with level(v) > l;
      insert appropriate template;
      set new levels;
```

Fig. 71a shows an example: The nodes of all level-2-quadrants are marked, if necessary. Then templates 1 and 2a out of the list are inserted, and the mesh is now conforming at the former hanging nodes of level 3. The procedure is repeated for the level-3-quadrants, whereafter no hanging nodes are left.

Figure 71: Construction of the conforming closure

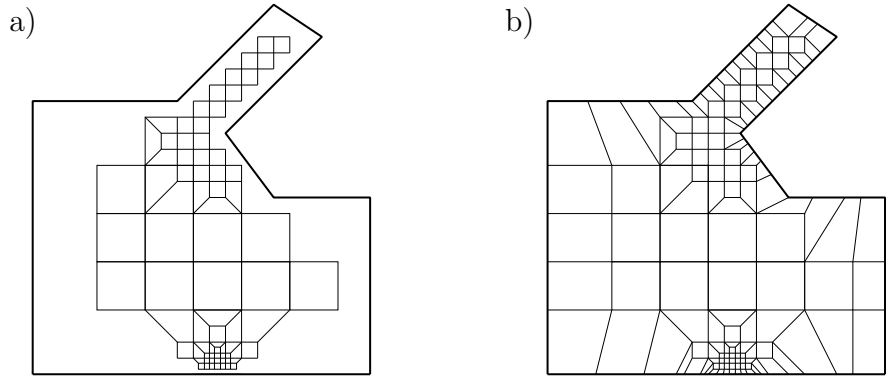


This procedure is very similar to the mesh refinement algorithm, and the choice of templates makes sure that the resulting mesh is conforming and that the minimum angle in the mesh is 45°.

Boundary fitting of the mesh can be done by using either the projection or the isomorphism technique; a short review of the latter one will be given here (see [Schneiders 1996b] for the details). A subset of the conforming quadrilateral element mesh is selected as the initial mesh (fig. 72a). This is not as straightforward as for the grid based algorithm: Care must be taken that the distance of each boundary edge e to the object boundary roughly equals the edge length (if this condition is not respected, elements with inacceptable aspect ratios may be generated).

One can then construct normals for the boundary nodes of the initial mesh, generate mesh nodes on the object boundary and construct elements in the boundary region

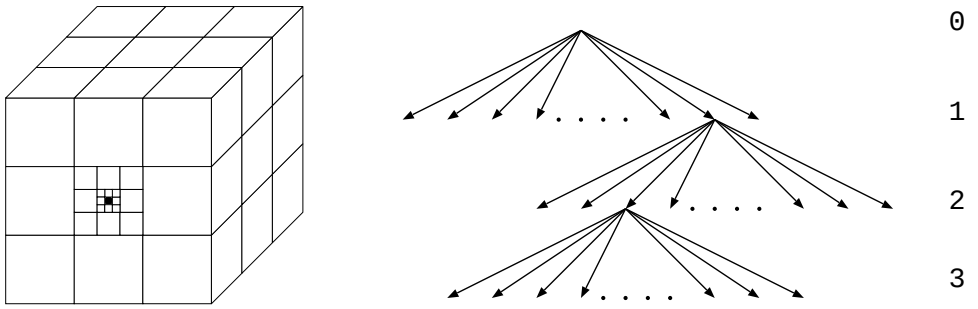
Figure 72: Adapting the mesh to the boundary



(fig. 72b). The mesh is then optimized, in a manner similar to grid-based mesh generation. The projection technique and the midpoint subdivision approach can also be used for boundary adaptation.

As in mesh refinement, octree-based mesh generation follows the same lines. Fig. 73 shows an example 3-octree, where the root cell has been divided into 27, 1 level-1 cell and one of it's children have been split.

Figure 73: Octree decomposition



Like in 2D, We can define the octant node levels $l(v)$ as the level of the smallest octant the node belongs to:

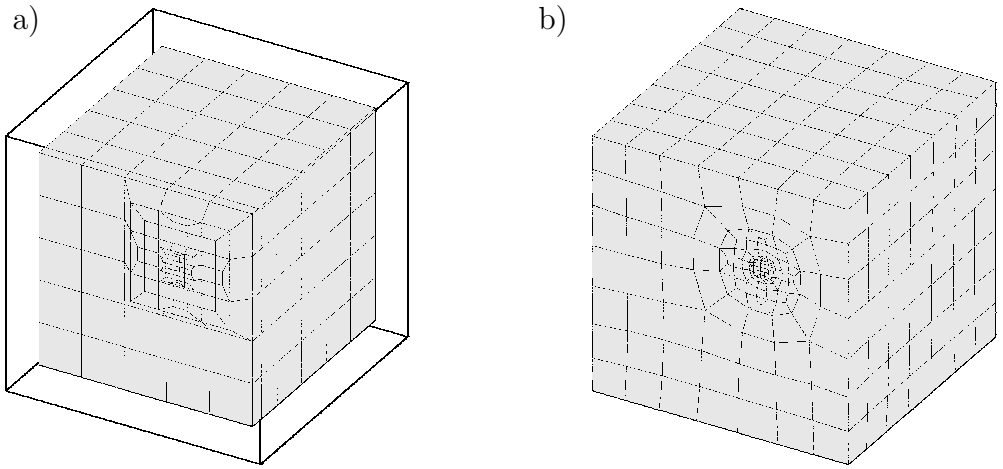
$$l(v) = \max\{l(q) |\ v \text{ is a node of } q\} \tag{6}$$

It is an easy task to extend the conforming closure algorithm for the 3D case. In the example above, the templates needed are all of "convex" type (fig. 58), so that a conforming hull can be found. In situations where this is not the case, the discussion of section 5.3 applies, and one can choose the modified set of 3D templates derived from the 2D templates in fig. 60.

The conforming hull can now be used as an initial mesh. To adapt the mesh to the object boundary, all nodes that are relatively closed to the boundary, and the adjecent elements, are removed (fig. 74a). The isomorphism technique is then used to construct a mesh at the boundary (fig. 74b).

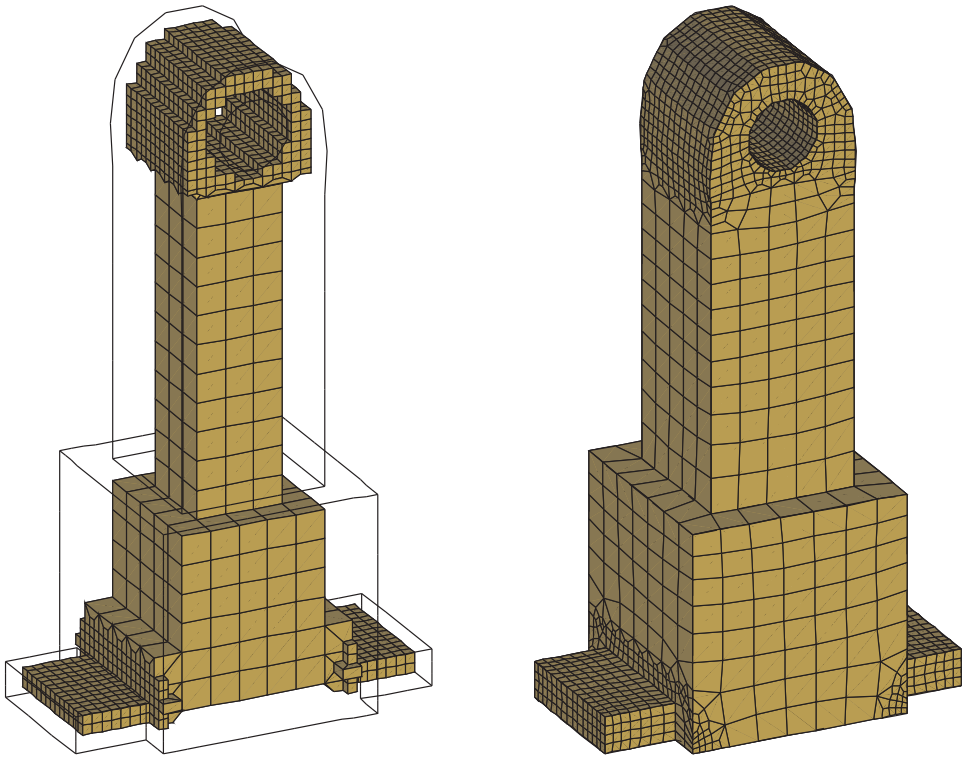
Fig. 75 shows an example mesh for a structural part. The fine parts of the mesh are

Figure 74: Initial mesh and hex mesh for a block



located in a way that the convex set of templates could be use; this allows for a large reduction of the number of elements compared to the grid-based algorithm.

Figure 75: Initial and hexahedral mesh for a structural part



5.7 Mesh generation based on 2-quadtrees and 2-octrees

Mesh generation based on the modification of 2-quadtrees and 2-octrees by template insertion poses severes problems. In particular, it is not possible to derive a complete set of 3D templates. In the following, we discuss how to derive the conforming hull from a 2-quadtree by template insertion, and the limits of this technique for the 3D case. In chapter 5.8 we will present another approach – directional refinement – that solves the problem for 3D.

A 2-quadtree is derived from the root quadrant by recursively splitting quadrants into 4 subquadrants. The depth in the quadtree defines the level of a quadrant (def. 6). The conforming hull can only be found for balanced quadtrees:

Def. 7 *A quadtree is called* balanced*, if the following holds:*

(i) For each quadrant, either none or all of its children have children themselves.

(ii) For the levels of adjacent quadrants q, q', $|l(q) - l(q')| \leq 1$ holds.

The following operations can be used for balancing a quadtree (fig. 76):

1. If condition (i) (def. 7) is violated for quadrant q, split those children of q that are leaf quadrants.
2. For adjacent leaf quadrants q, q' with $l(q) < l(q') - 1$, split q.

Figure 76: Quadtree balancing

a) b) c)

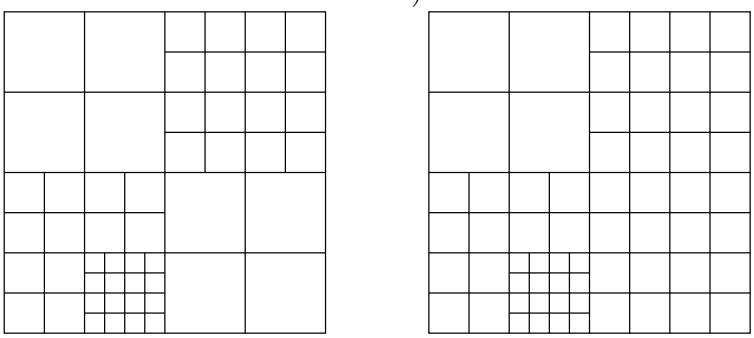


Fig. 76 shows the balancing process for the example quadtree. The marked nodes in fig. 76a are called *central nodes*: The centers of quadrants that have been splitted.

In order to control the template insertion process, each node is assigned the level of the smallest adjacent quadrant as the nodal subdivision level (def. 3). Quadrants of level $l(q)$ that have nodes with level $l(v) > l(q)$ (marked nodes) must be split, and the configuration of marked nodes determines the templates out of the list in fig. 77b that are inserted into the transitioning quadrants:

- If more than 2 nodes are marked, template 2 is chosen.
- If 2 nodes are marked, the position of the central node determines how template 1 is inserted.

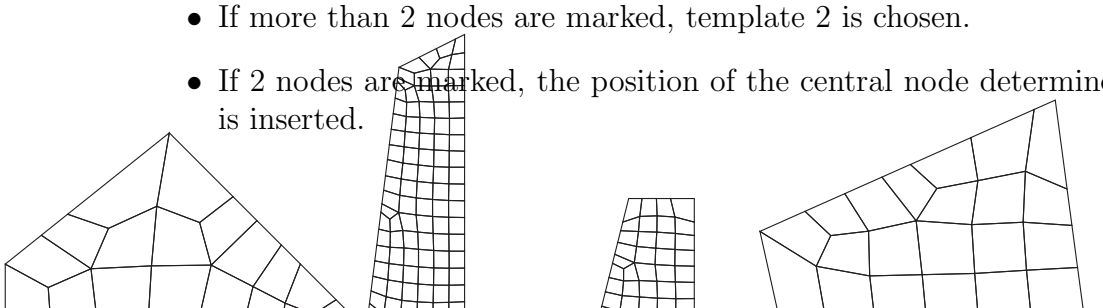
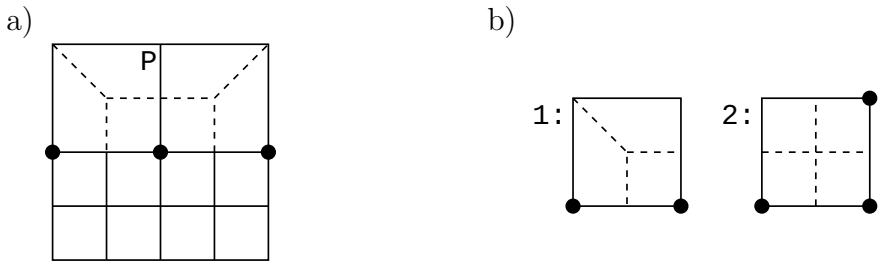


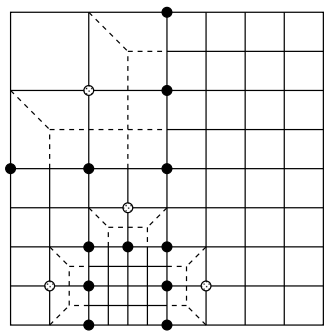
Figure 77: Local construction of the conforming hull



- In case of less than 2 marked nodes, the quadrant remains unchanged.

Fig. 78 shows the resulting mesh. Note that conformity is guaranteed by the refinement patterns, and that the balancing operations prevents the generation of sharp angles.

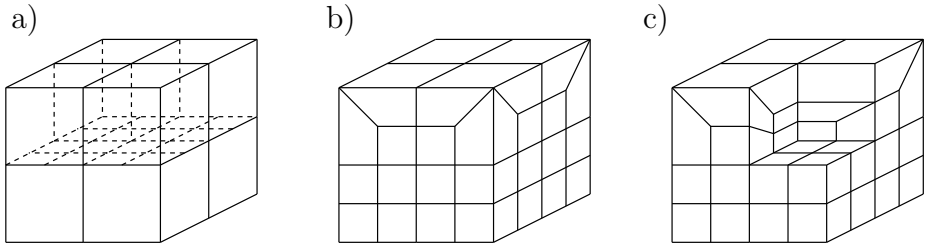
Figure 78: Resulting quadrilateral mesh



The algorithm naturally generalizes for the 3D case. The initial octree is balanced by octant splitting, octant and node levels can be computed. It remains to construct a 3D equivalent to the 2D set of templates (fig. 77). Unfortunately, it will turn out that not such set exists in 3D.

The transitioning cannot be done locally for one octant, 4 octants must be taken into account. Fig. 79 shows one case (cf. fig. 77a and case 3 in fig. 58): On the outward faces, the pattern is identical to the 2D one, so that conformity is guaranteed. The template will be discussed in detail in chapter 5.8.

Figure 79: Transitioning - 3D



As mentioned above, the template insertion is not valid for all situations. Fig. 80 shows a “non-convex” transitioning. Here, it is obvious to insert the 2D templates and then try to find a 3D hexahedral mesh that matches that surface. As outlined in chapter. 1, this is not possible since the number of quadrilateral faces is odd.

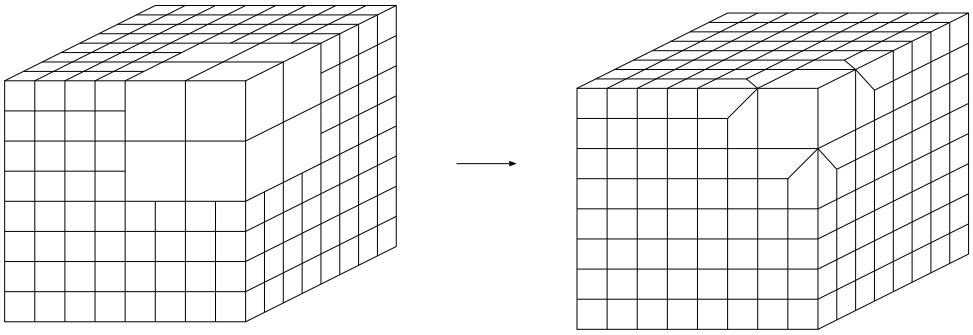


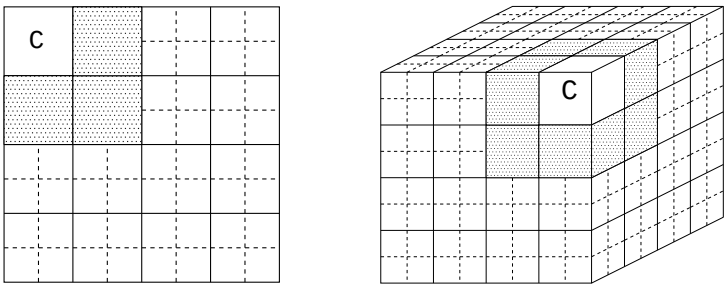
Figure 80: Non-convex transitioning

There seems to be no straightforward way to extend the 2D algorithm to the 3D case. In the next chapter, we will take a different look at the problem, and will come out with a solution for the non-convex transitioning problem and an algorithm for the construction of the conforming hull.

5.8 Mesh generation based on directional refinement

To get started, we take a closer look at the situation in which the template insertion technique fails. Fig. 81 shows the transitioning problem in 2D and 3D: A non-convex region has to be refined, quadrant (octant) C must not be split, and the transitioning must take place in the region between.

Figure 81: Non-convex transitioning problem



We start with an analysis of the 2D case, and use the spatial twist continuum introduced in chapter. 4 as a tool. The STC of the 4×4-mesh in fig. 81 is itself a 4×4-mesh. Refining the mesh is equivalent to the insertion of additional chords in the STC, and the idea is to do this independently for the x- and y-directions. For x-refinement, two straight chords c_2 and c_3 are inserted into the STC (Fig. 82a). Since we want to keep a coarse mesh in the left upper region, we insert a U-shaped chord c_1 on the left.

The insertion of chords in the STC corresponds to the insertion of element rows in the STC. Fig. 82b shows the resulting mesh (the dual of the STC), with the newly inserted elements marked grey.

We can now use the same technique for refining the mesh in the y-direction. As shown in fig. 83a, two straight-line rows c_2 and c_3 that are parallel to the x-axis are inserted in the STC. In order to avoid an overall refined mesh, the U-shaped chord c_1 is inserted in the upper half. Fig. 83b shows the resulting quadrilateral mesh, with the newly inserted buffer layers marked grey.

Figure 82: Refinement in the x-direction

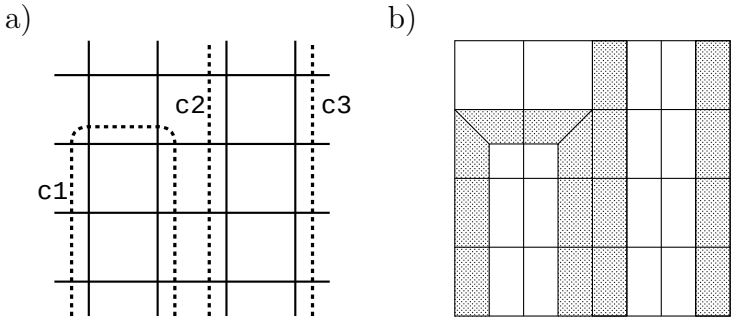
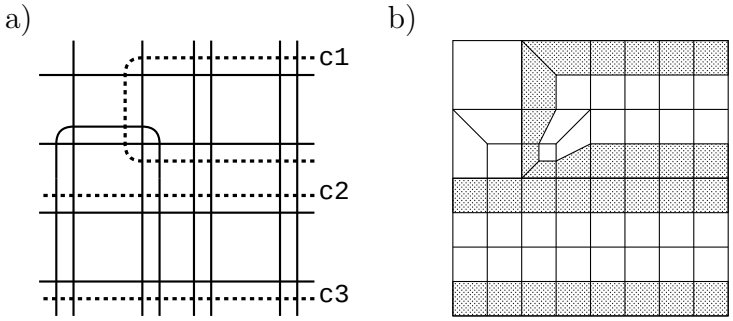
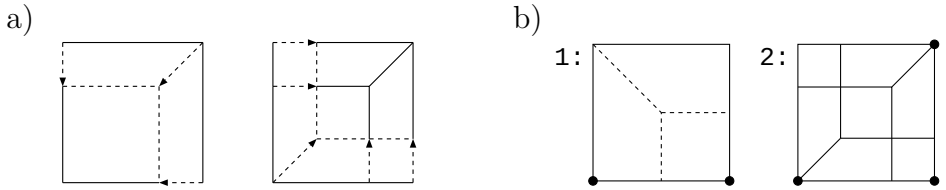


Figure 83: Refinement in the y-direction



If we interpret the refined mesh as the result of a template insertion process, we arrive at the new set of templates shown in fig. 84 (fig. 84a shows how template 2 is derived by succesive insertion of buffer layers). Note that if template 2 is used for the 3D example of fig. 80, the resulting surface has an even number of faces so that a volume mesh exists [Mitchell 1996].

Figure 84: a) Buffer layer insertion b) New set of templates

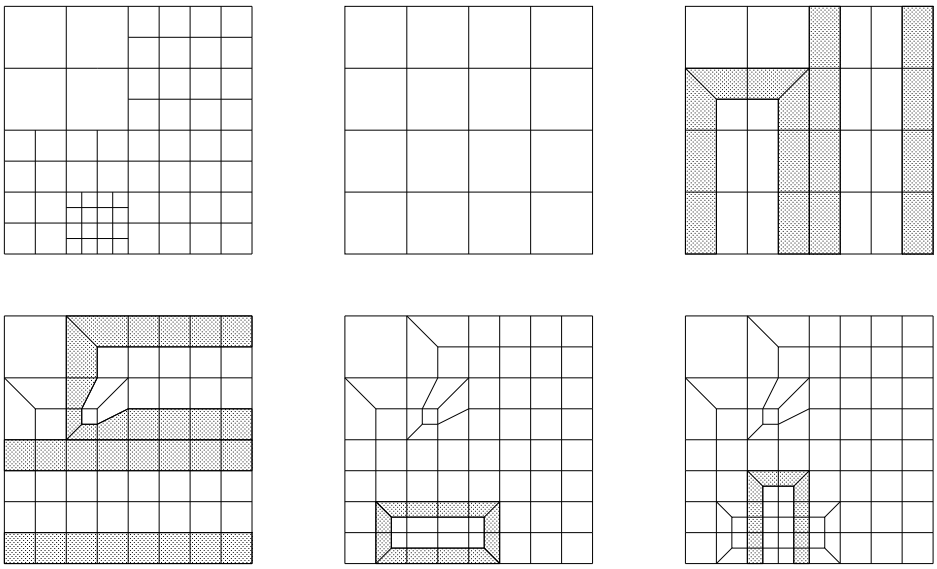


The approach presented above is a new algorithm for quadrilateral mesh refinement. We will call it *directional refinement*. Here, the quadtree serves for the definition of the element size, but is not used as a starting point of the algorithm. An example is shown in fig. 85: In order to obtain the mesh specified by the 2-level quadtree, a coarse mesh is refined by successive insertion of buffer layers. For each level, the required buffer layers are inserted, first in the x-, then in the y-direction. Each insertion operation does not introduce hanging nodes into the mesh, so that the resulting mesh is conforming.

The strong point of directional refinement is that it also generalizes for the 3D case. In particular, all refinement request can be processed, as we will see – non-convex situations pose no particular problem.

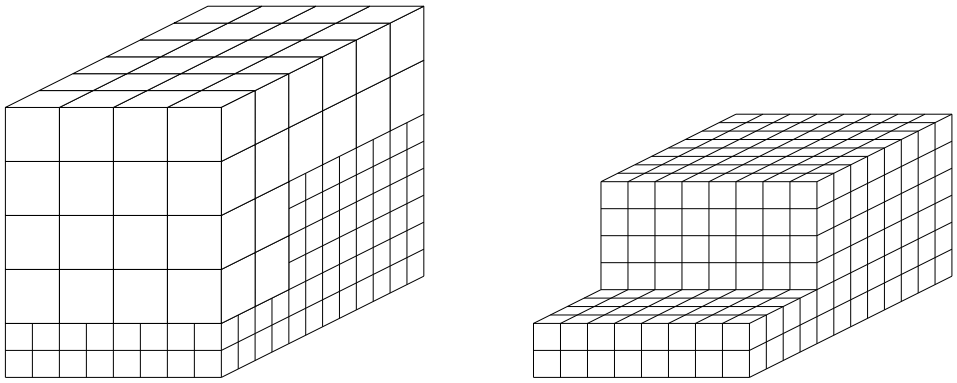
The definitions of quadrant level, node level and balanced octrees can be easily generalized for 3D. For hexahedral mesh generation, an initial octree is built, which is then

Figure 85: Adaptive mesh generation by directional refinement



balanced. It remains to derive the conforming hull; this will be explained for the example of fig. 86.

Figure 86: Balanced octree

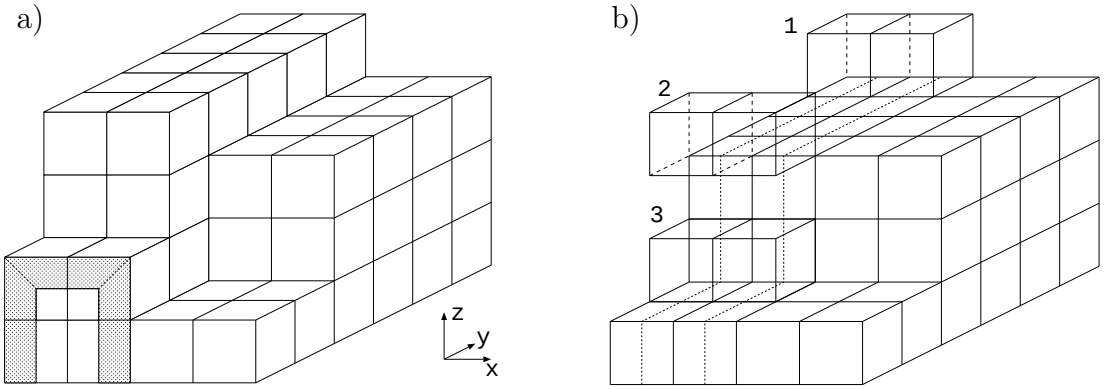


We start refining a conforming coarse mesh in the x-direction. As shown in fig. 87a, a U-shaped buffer layer is inserted for the two leftmost layers of the coarse mesh. The buffer layer can be seen as a 2D pattern which is swept through the mesh. Note that the resulting mesh is conforming.

While there are only 2 different transition patterns in 2D, in 3D there are 4 (fig. 87b). The buffer layer is inserted as follows:

- In case 1, the bottom faces adjacent to edge e have to be refined in the x-direction. For each node that is not adjacent to e (the marked nodes in fig. 88) a new node is generated so that a buffer layer of 4 elements can be constructed (marked grey in fig. 88).
- In case 2, only the back edges at node P must be split, which is achieved by the insertion of a buffer layer for the 6 faces that are not adjacent to P.

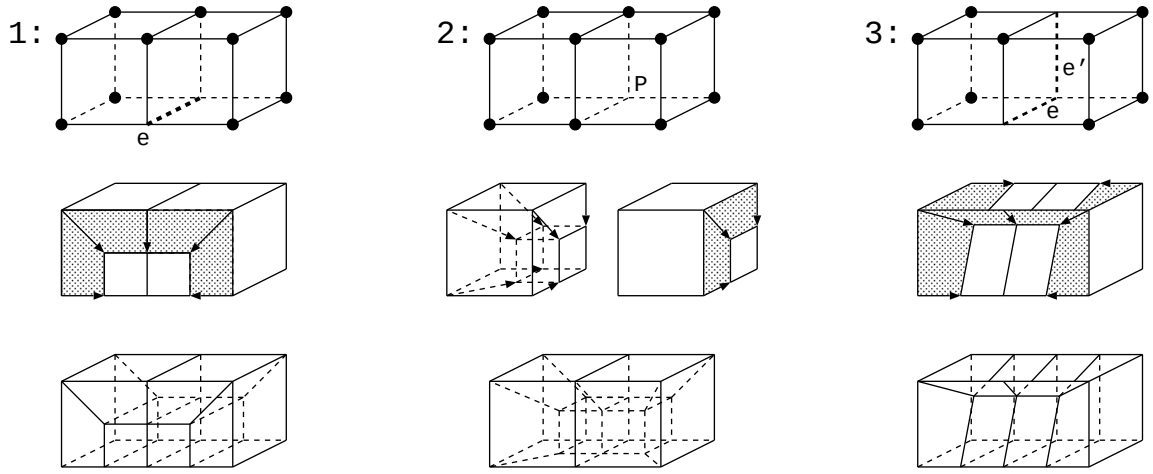
Figure 87: Refinement in x-direction



- In case 3, the back and bottom faces at the edges e and e' must be split. This is achieved by the insertion of two elements at the left and right face and by adding two dangling faces.
- In case 4, an element is split in half. This step poses no problems and is not discussed further.

Note that the patterns at the common faces match, so that the resulting mesh is conforming.

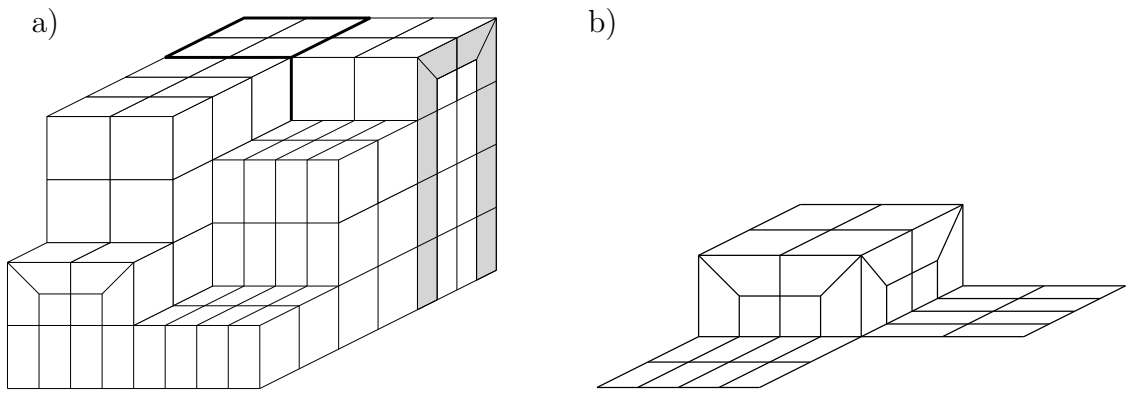
Figure 88: Refinement in x-direction - details



In the same way, one can insert buffer layers into the x-refined mesh to achieve a refinement in y-direction. The resulting transition patterns become more complicated, and we will look at some special cases only (the technique works in all situations). As in x-refinement, one can identify 4 cases. Fig. 89a shows an example where a 1-refinement in the x-direction "meets" a 1-refinement in the y-direction (cf. fig. 79).

The refinement in the x-direction (case 1) led to the configuration shown in fig. 90a. For the marked nodes, interior nodes are defined, and a buffer layer is created at the exterior faces of the configuration (fig. 90b). The buffer layer is marked grey in fig. 90 (element quality can be enhanced by laplacian smoothing). Note that the pattern in the

Figure 89: Refinement in y-direction



x-direction differs from the pattern in the y-direction. Fig. 91 shows the result of the x- and y-refinement for element q in fig. 90a.

Figure 90: Refinement in y-direction - details

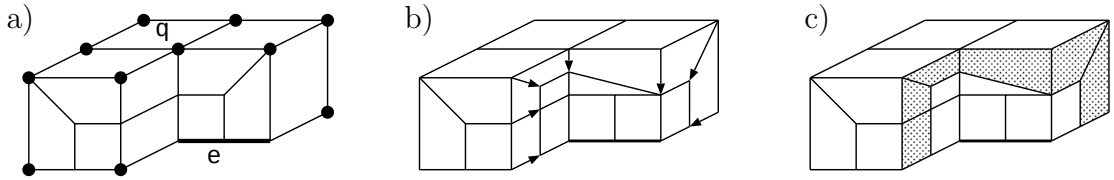
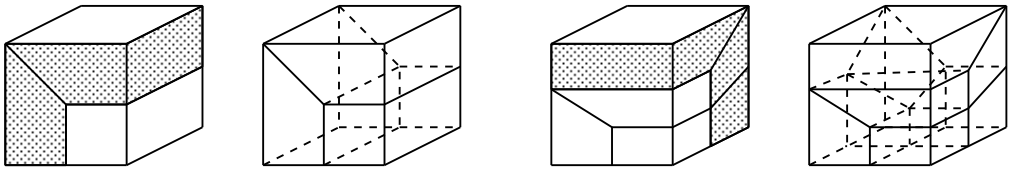


Figure 91: Further details



Having completed the refinement in y-direction (fig. 92a), buffer layers in z-direction are inserted into the mesh. This completes the refinement process. The result is shown in fig. 92a (note that the surface has the same structure as a 2D-refined meshed).

The local transition pattern is determined by the application of the insertion operators 1, 2 and 3 from fig. 88 and can be characterized by the tripel $(I_x, I_y, I_z) \in \{0,1,2,3\}^3$ which indicates the sequence of refinement operators (0 indicates that no refinement is done). In the example of fig. 89, the sequence is $(1,1,0)$. It is easy to see that the refinement sequences fall into one of the following classes:

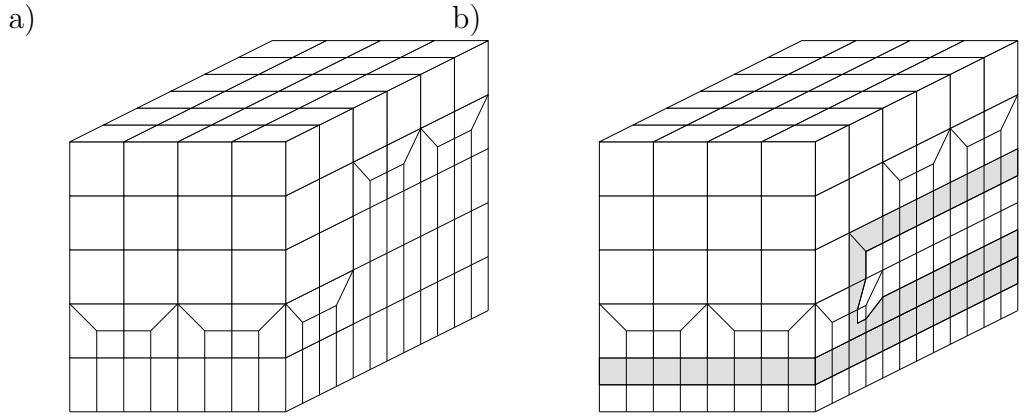
(i) One 2-, two 0-operators

(ii) Two 1-, one 0-operator

(iii) One 3-, two 1-operators

(iv) Three 3-operators

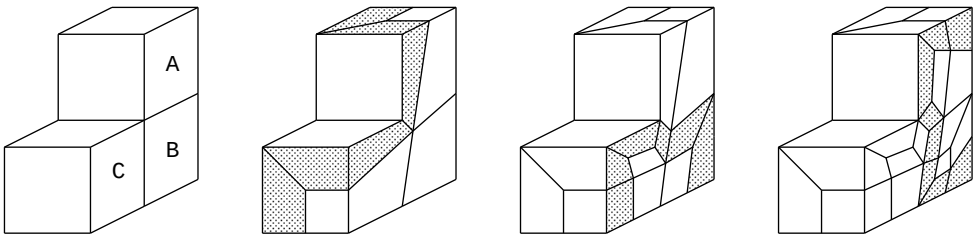
Figure 92: Refinement in y- and z-direction



The number of combinations is 10. In a template-insertion algorithm, one would have to construct a template for each combination – not a very elegant way to do the refinement; directional refinement is clearly superior.

An example of how the buffer layers are inserted is shown in fig. 93. The refinement sequences for the elements A, B and C are $(1, 0, 1)$, $(3, 1, 1)$ and $(1, 1, 0)$.

Figure 93: "Concave" refinement



In summary, the following algorithm generates an adapted hexahedral mesh:

- Construct a background octree T_b.
- Balance T_b so that the conditions of def. 7 hold.
- Generate a $4 \times 4 \times 4$-structured hexahedral mesh M.
- Starting with level 3, insert buffer layers into M as indicated by T_b.

Fig. 94a shows the initial mesh mesh that was generated for a mechanical part. In order to resolve the small features of the geometry, only refinement operator 1 had to be used. Fig. 94b shows how the mesh is adapted to the geometry. This step, done with the isomorphism technique, is equivalent to the insertion of a layer of elements (buffer layer) at the object boundary.

Fig. 95 shows a mesh that has been generated for the simulation of flow around a Peugeot car. The length of the problem domain is about 20m, while the distance between the car and the ground is only 20 cm. This example demonstrates the strength of octree-based methods: The ability to adapt the element size to the geometry. A 12-level-octree has been generated for the example, resulting in a mesh of less than 2000 elements.

Figure 94: Mesh for a mechanical part

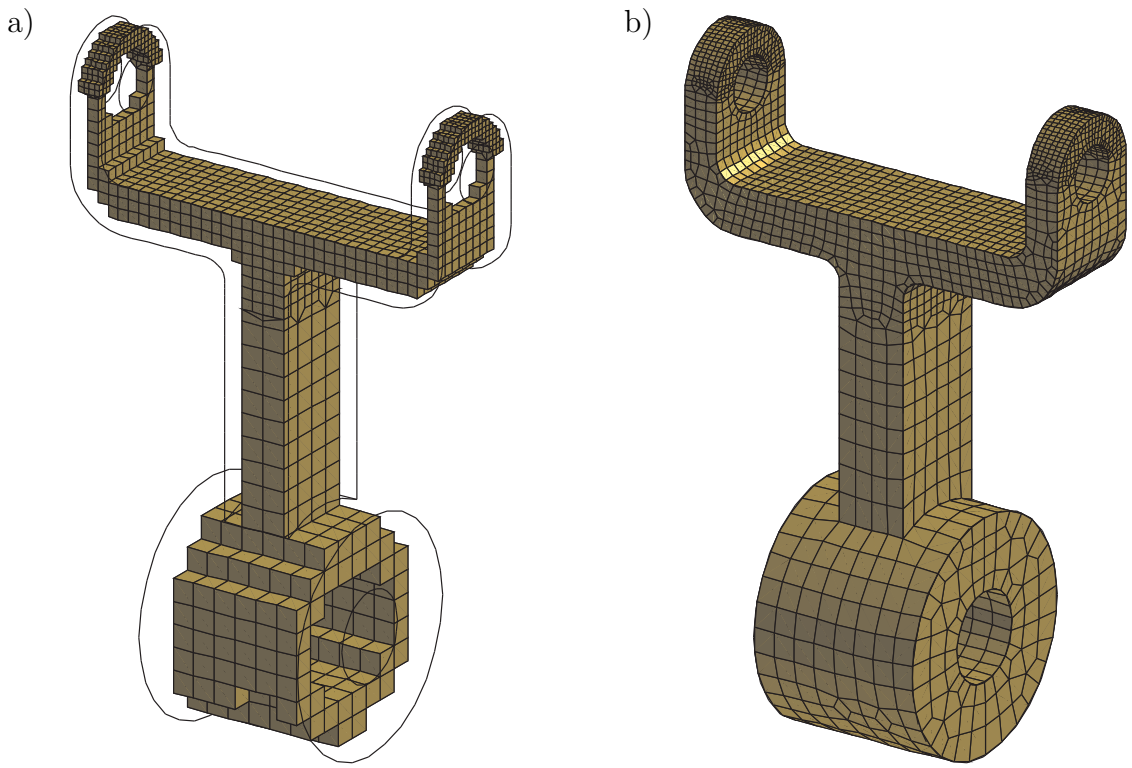


Figure 95: Example mesh (external flow)

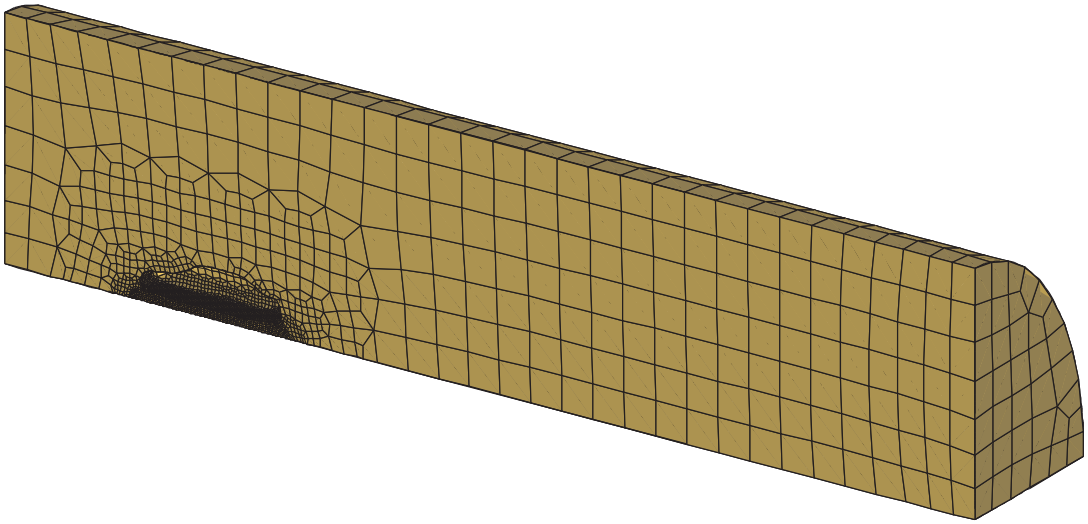
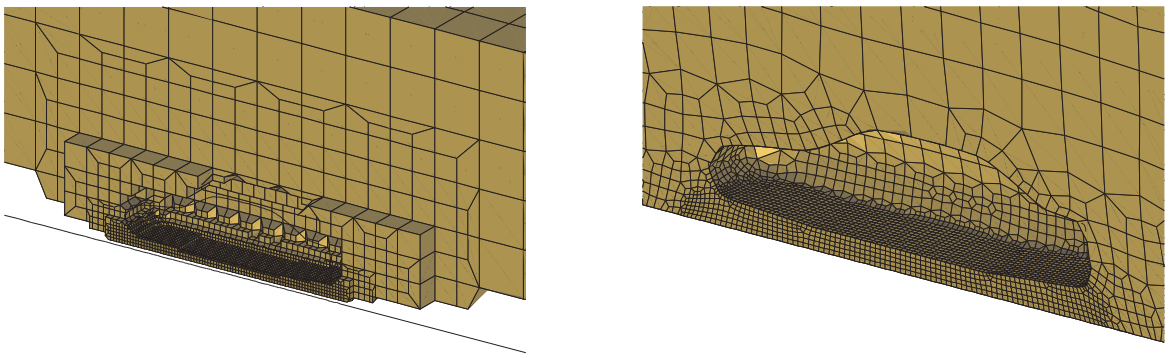


Fig. 96 reveals some details. One can easily add more detail to the mesh, e.g. generate smaller elements in the region near the bumper.

Figure 96: Detail



The example also shows a weak point of the method: The mesh structure at the boundary is determined by the structure of the volume mesh, so that a structured mesh is only generated in regions where the object boundary is parallel to one of the coordinate axes. A way to overcome this drawback is to choose a starting octant with curved boundaries that is aligned to the geometry.

6 Other Approaches

Advancing-front type methods are very popular for the generation of tetrahedral element meshes: First a mesh of triangles is generated for the surface, then a volume mesh is generated layer by layer. This allows the control of mesh quality near the boundary, internal faces can be represented in the mesh, and the method can be parallelized.

An adavancing-front type algorithm for the generation of quadrilateral element meshes was proposed in [Blacker and Stephenson 1991]. Fig. 97 gives an idea of how it works: Starting from a boundary discretization, the interior is "paved" with quadrilaterals layer by layer. If the layers overlap, a "seaming" procedure is invoked, and the procedure is repeated until the remaining cavities have been filled.

The paving algorithm is probably the best mesh generator for quadrilateral element meshes. It generates meshes of high-quality elements with an acceptable number of elements. Irregular nodes (interior nodes with other than four elements adjacent) are more likely to be found in the interior than close the boundary. The algorithm is very complex and not easy to implement.

An attempt to develop a three-dimensional version, the plastering algorithm, has been made in the CUBIT project. Starting from a quadrilateral discretisation of the object boundary, layers of hexahedral elements are generated in the volume (fig. 98). The number of different cases to be considered is far greater than in the 2D case. Degenerated elements (wedges) are removed by propagating them through the mesh [Blacker 1993].

Unfortunately, it turned out impossible to develop a robust algorithm. The problems arise when two fronts intersect. In the 2D case, one can glue ("seam") the fronts if the change in element size is not to large. This is not sufficient in 3D, since these surface

Figure 97: Paving algorithm

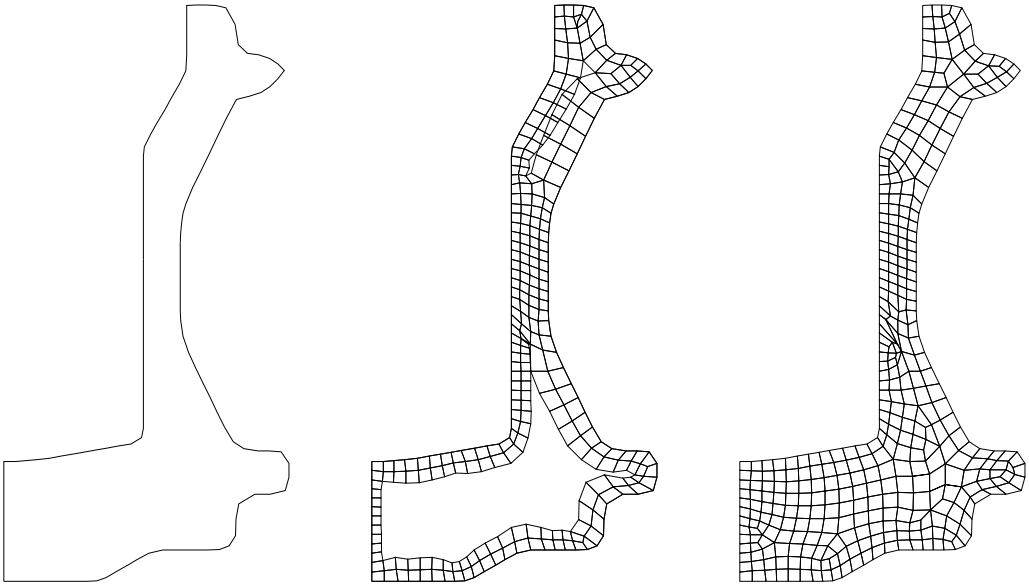
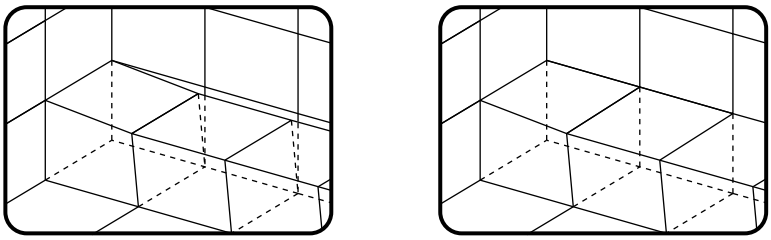


Figure 98: Plastering algorithm



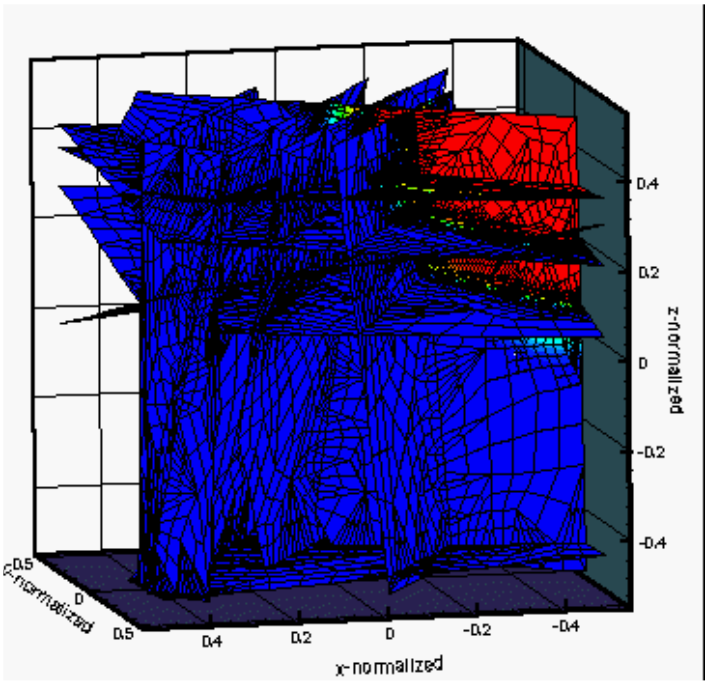
meshes must be isomorphic in order to seam them. This condition is unlikely to hold for practical problems. Another problem is that the cavities cannot be meshed in every case. Fig. 2 shows an example (the cavity can be reduced to an octahedron, a non-meshable object in the sense that a valid hex mesh that matches the octahedron surface has not yet been found).

It turns out that generating a hex mesh from a surface discretization is hard to realize if the decisions are made purely on local information. So the original idea was rejected, and global information was incorporated using the concept of the dual described in chapter 4.

An algorithm for the generation of hexahedral element meshes for very complicated domains (geological structures with internal boundaries) was proposed by Taniguchi [Taniguchi 1996]. His approach is similar to Armstrong's algorithm in that he decomposes the domain into simple subvolumes (tetrahedra, pentahedra, etc.) that are then meshed separately. The method is based on Delaunay triangulation, and, therefore can be applied for arbitrary convex domains which consist of a set of convex subdomains that are surrounded by fracture planes. Fig. 99 shows a mesh generated for the simulation of groundwater flow; for simulations like this it is very important that the boundaries between different layers of material are present in the mesh. The method has recently been extended [Kasper et. al. 1998].

A similar method for hexahedral element meshing of mechanical parts was proposed

Figure 99: Mesh for a geological structure with internal boundaries



by Sakurai [Shih and Sakurai 1996] (volume decomposition method). Also notable is the work of Shang-Sheng Liu [Liu 1996]; he tries to integrate the mesh generation into a solid modeling environment, an approach that is attractive particularily for mechanical engineering CAD systems.

So far we have concentrated on meshing strategies that can be applied both in two and three dimensions. There are, however, strategies for quadrilateral element mesh generation that cannot be extended to the 3D case. Two of these shall be discussed briefly.

The block decomposition approach used by Armstrong poses far fewer problems in 2D than in 3D. Whereas in 3D one must take care to generate subvolumes that can be split up into hexahedra, this is not really a problem in 2D, since every polygon with an even number of edges can be meshed with quadrilateral elements. So, there is much more room for finding a good partitioning strategy. An algorithm of this type is describe by Nowottny [Nowottny 1997] (fig. 100). First a multiply connected region is decomposed topologically. Then a sequence of optimal cuts is inserted until sufficiently small subregions have been generated. The subregions are meshed by using an advancing front algorithm.

The example of fig. 101 demonstrates the potential of the method. The strategy works in 2D since a suffienctly large polygon can always be split into two meshable subpolygons. This does not hold in 3D (fig. 2), and thus an extension of this strategy seems difficult to realized. Nowottny investigates a modified decomposition method.

Another approach for the generation of quadrilateral element meshes was proposed in [Shimada 1994]: First one generates a triangular mesh with an even number of elements; pairs of triangles are then combined to quadrilaterals until no triangles remain.

This approach is very elegant since it allows the use of the work done on triangulation

Figure 100: Quadrilateral mesh generation by geometrically optimized domain decomposition

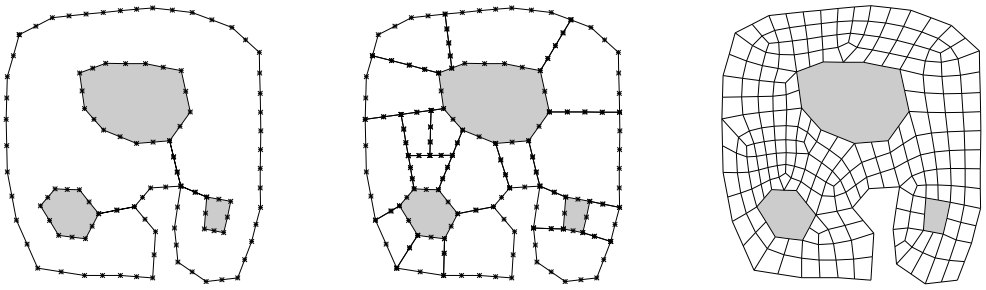
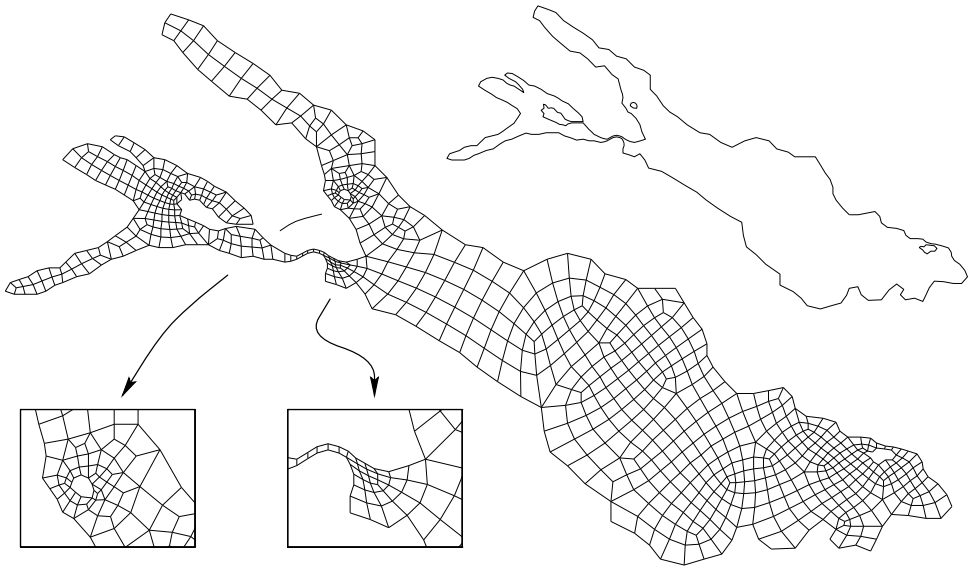


Figure 101: Mesh of Lake Constance



algorithms. Especially obtaining graded meshes or meshes for geometries with internal boundaries is straightforward using this approach. Unfortunately, it cannot be used for 3D since combining tetrahedra into hexahedra is not possible except for tet meshes with a very regular structure.

7 Acknowledgements

The work benefited from the support of the following people: Pictures were contributed from C. Armstrong (Queens University Belfast), M. Delanaye (Numeca SA), M. Müller-Hannemann (TU Berlin), H. Kasper (Universität Hannover), T. Taniguchi (Okayama University) and D. Nowottny. M. Schneider (MAGMA Gießereitechnologie GmbH) helped in translating the text. The author acknowledges their support of the report.

References

[Blacker and Stephenson 1991] T.D. Blacker, M.B. Stephenson (1991): Paving: a new approach to automated quadrilateral mesh generation. Int. Jou. Num. Meth. Eng. 32 (1991), pp. 811-847.

[Blacker 1993] T.D. Blacker, R.J. Meyers (1993): Seams and wedges in Plastering: a 3D hexahedral mesh generation algorithm. Engineering with Computers 9, pp. 83-93.

[Brodersen et al. 1996] O. Brodersen, M. Hepperle, A. Ronzheimer, C.-C. Rossow, B. Schöning: The Parametric Grid Generation System MegaCads. Proc. of the 5th Intern. Conf. on Numerical Grid Generation in Computational Field Simulations. Ed. B.K. Soni, J.F. Thompson, J. Häuser, P. Eiseman, NSF, Mississippi, 1996, pp. 353-362.

[Calvo2000] N. Calvo, Y. Idelsohn: All-hexahedral element meshing: Generation of the dual mesh by recurrent subdivision. Computer Methods in Applied Mechanics and Engineering 182 (3-4) pp. 371-378 (2000).

[Carbonera] C. Carbonera: Constrained mesh generation.
`http://www-users.informatik.rwth-aachen.de/~roberts/SchPyr/index.html`

[Dhondt 1999] G. Dhondt: Unstructured 20-node brick element meshing. Proceedings 8th International Meshing Roundtable, October 10-13, South Lake Tahoe, California, U.S.A., 369-376 (1999).

[George 1991] P.L. George: Automatic mesh generation: Applications to Finite Element Methods. John Wiley & Sons.

[Müller-Hannemann 1995] R. Möhring, M. Müller-Hannemann, K. Weihe: Using Network Flows For Surface Modeling. Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 350-359.

[Holmes 1995] D. Holmes: Generalized Method of Decomposing Solid Geometry into Hexahedron Finite Elements. Proceedings 4th International Meshing Roundtable, Sandia National Laboratories, pp. 141-152.

[Ives 1995] D. Ives: Geometric Grid Generation. Surface Modeling, Grid Generation, and Related Issues in Computational Fluid Dynamic (CFD) Solutions. Proceedings NASA-conference, Cleveland, Ohio, NASA CP-3291.

[Kasper et. al. 1998] H. Kasper, T. Rother, C. Thorenz, G. Kosakowski, T. Taniguchi and O. Kolditz: 3D Cad System for Numerical Analysis of Subsurface Flow and Transport. Proceedings 6th International Conference on Numerical Grid Generation in Computational Field Simulations, London, 1998.

[Knupp 1995] P. Knupp, S. Steinberg: Fundamentals of Grid Generation. CRC Press, ISBN 0-8493-8987-9.

[Liu 1996] S.-S. Liu, R. Gadh: Basic LOgical Bulk shapes (BLOBs) for finite element hexahedral mesh generation. Proceedings 5th International Meshing Roundtable.

[Müller-Hannemann 1999] Matthias Müller-Hannemann: Hexahedral Mesh generation by Successive Dual Cycle Elimination. Engineering with Computers 15, pp. 269-279, (1999)

[Mitchell 1996] S.A. Mitchell: A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume. Proceedings STACS'96, Grenoble.

[Murdock et al. 1997] P. Murdock, S.E. Benzley, T.D. Blacker and S.A. Mitchell: The spatial twist continuum: A connectivity based method for representing and contructing all-hexahedral finite element meshes. Finite elements in analysis and design 28, pp. 137-149 (1997).

[Nowottny 1997] D. Nowottny: Quadrilateral Mesh Generation via Geometrically Optimized Domain Decomposition. Proc. 6th International Meshing Roundtable, Park City, Utah, pp. 309-320.

[Owen 1996] S. Owen: Meshing Research Corner. Literature database, URL `http://www.ce.cmu.edu/~sowen/mesh.html`

[Preparata and Shamos 1985] Computational Geometry: An Introduction. Springer Verlag, New York, pp. 24-26 (1985).

[Price, Armstrong, Sabin 1995] M.A. Price, C.G. Armstrong, M.A. Sabin: Hexahedral Mesh Generation by Medial Axis Subdivision: I. Solids with convex edges. Int. Jou. Num. Meth. Eng. 38, pp. 3335-3359.

[Price and Armstrong 1997] M.A. Price, C.G. Armstrong: Hexahedral Mesh Generation by Medial Axis Subdivision: II. Solids with Flat and Concave Edges. Int. Jou. Num. Meth. Eng. 40, pp. 111-136.

[Sabin 1991] M. Sabin (1991): Criteria for comparison of automatic mesh generation methods. Adv. Eng. Softw. 13, pp. 220-225.

[Schneiders 1996a] R. Schneiders: A grid-based algorithm for the generation of hexahedral element meshes. Engineering with Computers 12, pp. 168-177.

[Schneiders 1996b] R. Schneiders, R. Schindler, F. Weiler: Octree-based Generation of Hexahedral Element Meshes. Proc. 5th International Meshing Roundtable, Sandia National Laboratories, pp.205-216.

[Schneiders 1996c] R. Schneiders: Refining Quadrilateral and Hexahedral Element Meshes. Proc. 5th International Conference on Numerical Grid Generation in Computational Field Simulations, pp. 699-708.

[Schneiders 1996d] R. Schneiders: Mesh Generation and Grid Generation on the Web. `http://www-users.informatik.rwth-aachen.de/~roberts/meshgeneration.html`.

[Schneiders 1998] Octree-based hexahedral mesh generation. To appear in Journal of Computational Geometry and Applications, special issue on mesh generation (1998).

[Shewchuk 1998] J.R. Shewchuk: A Condition Guaranteeing the Existence of Higher-Dimensional Constrained Delaunay Triangulations. Submitted to the Fourteenth Annual Symposium on Computational Geometry (1998).

[Shih and Sakurai 1996] B.-Y. Shih, H. Sakurai: Automated Hexahedral Mesh Generation by Swept Volume Decomposition and Recomposition. Proceedings 5th International Meshing Roundtable.

[Shimada 1994] K. Shimada and T. Itoh: Automated Conversion of 2D Triangular Meshes into Quadrilateral Meshes. Proceedings International Conference on Computational Engineering Science.

[Smith 1996] R.J. Smith, M.A. Leschziner: A Novel Approach to Engineering Computations for Complex Aerodynamic Flows. Proceedings 5th International Conference on Numerical Grid Generation in Computational Field Simulations, pp. 709-716.

[Taghavi 1994] R. Taghavi: Automatic, parallel and fault tolerant mesh generation from CAD on Cray research supercomputers. Proc. CUG Conference, Tours, France.

[Tam and Armstrong 1993] T.K.H Tam, C.G Armstrong: Finite element mesh control by integer programming. Int. Jou. Num. Meth. Eng. 36, pp. 2581-2605.

[Taniguchi 1996] T. Taniguchi: New Concept of Hexahedral Mesh Generation for Arbitrary 3D Domain – Block Degeneration Method. Proc. 5th International Conference on Numerical Grid Generation in Computational Field Simulations, pp. 671-678.

[Tautges 1996] T.J. Tautges, S. Mitchell: Progress report on the whisker weaving all-hexahedral meshing algorithm. Proceedings 5th International Conference on Numerical Grid Generation in Computational Field Simulations, pp. 659-670 (1996).

[Tchon 1997] K.-F. Tchon, C. Hirsch, R. Schneiders: Octree-Based Hexahedral Mesh Generation for Viscous Flow Simulations. Proceedings 13th AIAA Computational Fluid Dynamics Conference, Snowmass, CO.

[Thompson 1985] J.F. Thompson, Z.U.A. Warsi and C.W. Mastin (1985): Numerical Grid Generation: Foundations and Applications. North Holland

[Thompson 1999] J.F. Thompson, B. Soni and N. Weatherhill (1999): Handbook of Grid Generation. CRC Press, ISBN 0-8493-2687-7.

[Turkkiyah 1995] G.M. Turkkiyah, M.A. Ganter, Duane W. Storti, Hao Chen: Skeleton-Based Hexahedral Finite Element Mesh Generation of General 3D Solids. Proceedings 4th International Meshing Roundtable, Sandia National Laboratories, late addition.