NETGEN - An Advancing Front 2D/3D-Mesh Generator Based on Abstract Rules

Joachim Schöberl*

Abstract

In this paper, the algorithms of the automatic mesh generator NETGEN are described. The domain is provided by a Constructive Solid Geometry (CSG). The whole task of 3D mesh generation splits into four subproblems of special point calculation, edge following, surface meshing and finally volume mesh generation. Surface and volume mesh generation are based on the advancing front method. Emphasis is given to the abstract structure of the element generation rules. Several techniques of mesh optimization are tested and quality plots are presented.

1 Introduction

Many engineering disciplines require partial differential equation (pde) modeling. Two of the most popular methods to treat pdes numerically are the finite element method and the finite volume method. Both require partitioning the domain of interest into a set of simple domains, the elements. In two dimensions triangles and quadrilaterals are used, in three dimensions tetrahedra, hexahedra and pentahedra are the most popular elements. It is desirable to perform this partitioning automatically. There are several approaches working well in the plane, but cause a lot of difficulties in three dimensions. Overviews of mesh generation algorithms are given in [1], [13].

One of the most popular methods is the advancing front technique, see e.g. [10], [14], [9], [8], [6]. The input to this method is a boundary mesh. Starting with the original boundary, element by element is cut off to reduce the domain iteratively. The state of the algorithm is always represented by the advancing boundary front. In several papers, rules for element generation are presented. We want to emphasize, how to handle these rules by means of an abstract rule description. The advancing front method can be applied for plane, for surface as well as for three dimensional mesh generation. Our approach to rule application leads to a highly unified code for these three kinds of mesh generation.

Advancing front mesh generation starts with a given boundary mesh. In 3D, boundary mesh generation is also a non-trivial task. It can be solved by an advancing front surface mesh generator starting at the boundaries of the smooth pieces of the surface, namely the edges. Edge finding can be split into calculation of initial points and implicit curve following. Depending on the geometric model, initial point calculation can be complicated, too. We use the Constructive Solid Geometry (CSG) - model, defining complicated solids by the Boolean operations applied to simple primitives [11]. For this model we have to solve many nonlinear equations in three variables to calculate initial points for edge detection. The whole mesh generation problem is sketched in Figure 1.

*This work is supported by the Austrian Science Fund - 'Fonds zur Förderung der wissenschaftlichen Forschung' - under project P 10643-TEC

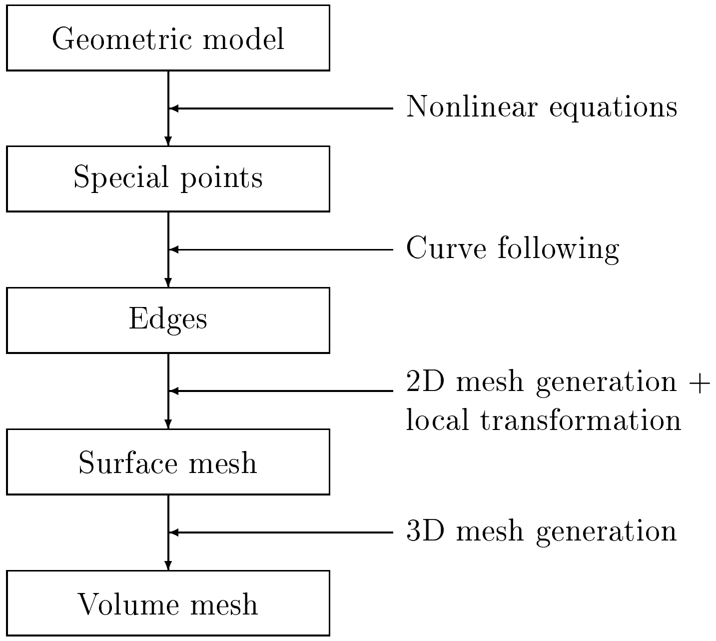

Figure 1: Subproblems in mesh generation

The paper is organized accordingly to the several sub-problems. In §2 the geometric model in use is described. The problem of calculating corners and more special points is described in §3, edge detection in §4. Our implementation of the advancing front method for plane, surface and volume mesh generation is explained in §5. The used mesh optimization strategies are discussed in §6. In §7 examples are given and in §8 the current work is summarized.

2 Geometric Modeling

The wide field of computational geometry provides several possibilities for geometric modeling [11]. The different models have complementary properties with respect to ease of solid description and ease of mesh generation algorithms.

The Constructive Solid Geometry (CSG) - model uses smooth primitives like cylinders and spheres to build more complex solids by the Boolean operations. In the computer, the solid is represented by a binary tree. The leaves are the primitives, and the nodes are the operations intersection, union and complement. A lot of mechanical production parts can be described by this model very simply, but it takes some numerical effort to find the corners and edges.

Explicit surface description, e.g. by spline patches, is very well suited for smooth surfaces like car bodies and airplane wings [7]. It is difficult to describe geometries with cutting edges, but mesh generation for explicit surfaces can be much simpler. The flexibility of both methods can be combined by using CSG primitives described by spline patches.

To apply mesh generation algorithms for the CSG model, some operations have to be implemented for every class of primitives. For globally convergent algorithms to find the corners we need the test 'Is a given cube contained in the primitive ?' The possible answers are 'no', 'partially' and 'yes'. For edge detection as well as surface meshing we need local quantities like the normal vector to the smooth boundary of a primitive.

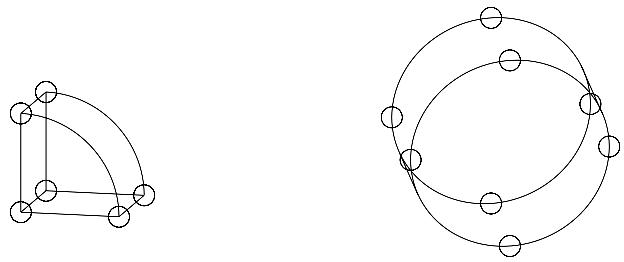

Figure 2: Examples for points of intersection and points of extreme coordinates

3 Special Point Calculation

The first step in mesh generation is the calculation of special points. For reasons of numerical approximation it is important to place mesh nodes at the corners of the geometry. These 'points of intersection' are points, where at least three surfaces intersect (Figure 2). The computation of them is explained in §3.1.

Starting at these points, one can follow the edges. But not every edge is terminated by two points of intersection. E.g., a cylinder has closed edges (Figure 2). To get uniquely defined points on these curves one can choose the points with minimal or maximal x, y or z coordinates. The computation of the 'points of extreme coordinates' is described in §3.2. There are more types of special points arising from some kind of degeneration. These points are discussed in §3.3. Together with the special points, tangents to the edges are calculated.

3.1 Points of intersection

Points of intersection are points, where at least three surfaces intersect. For implicitly defined surfaces these are the roots of the nonlinear system

$$\begin{aligned} f_1(x,y,z) &= 0, \\ f_2(x,y,z) &= 0, \\ f_3(x,y,z) &= 0. \end{aligned} \tag{1}$$

Robust and flexible solving procedures for this problem are bisection algorithms based on geometric tests. We start with a cube containing the whole solid. This cube is cut into eight cubes of half the size recursively, until a necessary criterion to have a point of intersection in the cube is not fulfilled anymore or the prescribed precision is reached. A very cheap criterion is the number of surfaces cutting the cube, which has to be at least three. The evolution of a bisection algorithm for a 2D example is given in Figure 3. If the point of intersection is regular, the bisection algorithm converges with linear rate.

3.1.1 Localization

The bisection algorithm uses only information of the solid contained in a cube. If we use a CSG model, the solid can be reduced to a local model in the cube by a recursive algorithm on the binary solid tree. The local model is **0** , if the cube is outside of the solid, **S**, if the cube is partially contained in the solid, or **1**, if the cube is fully contained in the solid. The state **S** is described by a smaller CSG model again. A primitive has to provide a function to determine the relation of a cube to the solid. For the Boolean operations, the localization can be expressed by the localization of the operands:

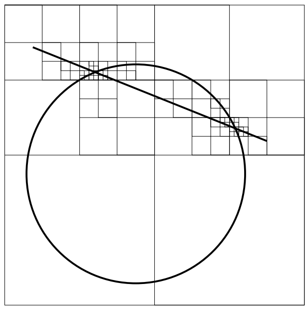

Figure 3: Bisection algorithm

$\cap$	**0**	S_1	**1**
0	**0**	**0**	**0**
S_2	**0**	$\mathsf{S}_1 \cap \mathsf{S}_2$	S_2
1	**0**	S_1	**1**

$\cup$	**0**	S_1	**1**
0	**0**	S_1	**1**
S_2	S_2	$\mathsf{S}_1 \cup \mathsf{S}_2$	**1**
1	**1**	**1**	**1**

	$\backslash$
0	**1**
S	$\backslash\mathsf{S}$
1	**0**

A minimal number of three primitives in the local model is used as necessary condition in the bisection algorithm.

3.1.2 Newton's method

Let us assume, we can evaluate the functions f_i as well as their derivatives. If we are close to a root of the nonlinear system (1), we can use the locally quadratic convergent Newton method to calculate the coordinates in just a few steps within machine precision [12]. With $F(x) := (f_1(x,y,z), f_2(x,y,z), f_3(x,y,z))$ and a given initial point x^0 Newton's method is defined by

$$x^{k+1} = x^k - \left(F'\left(x^k\right)\right)^{-1} F\left(x^k\right).$$

For (1), the geometric interpretation of Newton's method is as following. In the limit, the components of $F(x)$ measure the distance of x to the corresponding surface, and the columns of the matrix F' are scalar multiples of the normal vectors of the surfaces. The new point x^{k+1} is the point of intersection of the three tangential plains. We need the regularity of the solution to apply Newton's method.

We see, which operations a primitive has to provide. In a small environment of the surface, we need a linear approximation to an implicit representation of the surface. For explicitly given surfaces we can define an implicit function by

$$f(x) = (x - P(x), n(P(x))), \tag{2}$$

where $P(x)$ is the projection of x onto the surface, and n is the normal vector to the surface. The derivative of f is

$$\nabla f(x) = n(P(x)).$$

The non-trivial step for explicit surfaces is the implementation of the projection $P(x)$. For spline surfaces this projection can be implemented fast and robust by using the convex hull property and zoom-in algorithms [7].

Before starting Newton's method one has to check whether it converges to a unique solution. This test is given in Kantorovich's theorem [12]:

Let $B(x^0, r)$ be a sphere containing the current box.
Assume that $F'(x^0)$ is regular and the conditions

$$\begin{aligned} \|F'(y) - F'(x)\| &\leq \gamma\|y - x\| \qquad \forall x, y \in B(x^0, r), \\ \|F'\left(x^0\right)^{-1}\| &\leq \beta, \\ \|F'\left(x^0\right)^{-1} F\left(x^0\right)\| &\leq \eta, \end{aligned}$$

are fulfilled such that

$$\alpha := \beta\gamma\eta < \frac{1}{2}, \tag{3}$$

$$t_* := \frac{1}{\beta\gamma}\left(1 - \sqrt{1 - 2\alpha}\right) < r. \tag{4}$$

Then Newton's method with initial point x^0 is well defined and x^k converges to a root of F in $B(x^0, t_*)$. The root is unique in $B(x^0, t_{**})$ with

$$t_{**} = \frac{1}{\beta\gamma}\left(1 + \sqrt{1 - 2\alpha}\right).$$

In a small environment of a regular solution the constants β and γ are bounded. By further subdivision we can reduce η until (3) and (4) are fulfilled and the solution is unique in the bisection interval.

To implement the convergence test we have to evaluate the constants β, γ and η. For this we need in addition to the implicit function value and its derivative an estimate for the local Lipschitz - constant γ of ∇f. For the implicit function f defined in (2), the bound can be calculated by means of the main curvature κ. In the $(2\kappa)^{-1}$ environment of the surface the Hessian $\nabla^2 f$ is bounded by 2κ.

3.2 Points of extreme coordinates

As mentioned above, we also want to find points of minimal and maximal x, y and z coordinates on an edge. The problem of finding the maximal x coordinate on the edge defined by the intersection of the surfaces f_1 and f_2 is the constraint maximization problem

$$\max_{f_1 = f_2 = 0} x.$$

The corresponding Lagrangian for this problem is

$$\mathcal{L}(x, y, z, \lambda_1, \lambda_2) = x + \lambda_1 f_1(x, y, z) + \lambda_2 f_2(x, y, z).$$

Necessary conditions for points of extreme coordinates are the Kuhn-Tucker conditions of first order

$$\begin{aligned} \nabla_{(x,y,z)}\mathcal{L} = (1, 0, 0)^t + \lambda_1 \nabla f_1 + \lambda_2 \nabla f_2 &= 0, \\ f_1 &= 0, \\ f_2 &= 0. \end{aligned}$$

By the elimination of the Lagrange multipliers we get the non-linear 3×3 system

$$\begin{aligned} f_1 &= 0, \\ f_2 &= 0, \\ f_3 := (\nabla f_1 \times \nabla f_2)_x &= 0. \end{aligned} \tag{5}$$

For degenerated edges, e.g., if a sphere is put on top of a cylinder, the third equation $f_3 = 0$ is fulfilled on the whole edge. The following algorithms are not able to handle degenerated edges, so we demand edge regularity in the form of

$$\|\nabla f_1 \times \nabla f_2\| \geq \varepsilon_1 \|\nabla f_1\| \, \|\nabla f_2\|, \tag{6}$$

with the application specific, small parameter ε_1. If the edge is parallel to the y-z plane, then every point on that edge fulfills the necessary Kuhn-Tucker conditions of first order. So we demand the numerically sufficient Kuhn-Tucker condition of second order

$$s^T \nabla^2 \mathcal{L}\,(x, \lambda)\, s \geq \varepsilon_2 \|s\|^2 \quad \forall s \in \overline{C}. \tag{7}$$

The tangential cone $\overline{C}$ is the one dimensional space spanned by $\nabla f_1 \times \nabla f_2$. The sufficient Kuhn-Tucker condition ensures isolated solutions.

3.2.1 Bisection criteria

To calculate the points of extreme coordinates, the first step is again a bisection algorithm. A necessary condition is that there are at least two surfaces in the localization of the solid. To make use of the third equation $f_3 = 0$, we apply the mean value theorem $\|f\,(x) - f\,(x^*)\,\| \leq \|x - x^*\| \sup_{\xi \in [x, x^*]} \|f'\,(\xi)\,\|$. A necessary condition for a solution in the ball $B(c, r)$ containing the bisection cube is

$$|f_3(c)| \leq r \sup_{\xi \in B(c,r)} \|\nabla f_3(\xi)\|. \tag{8}$$

This can be estimated by the computable bound

$$\|\nabla f_3\| \leq \left(\|\nabla^2 f_1\| \|\nabla f_2\| + \|\nabla^2 f_2\| \|\nabla f_1\|\right).$$

These necessary conditions are only sufficient, if the edge is regular according to (6) and the second order Kuhn-Tucker condition (7) is fulfilled. Both conditions are specified for the solution of (5), but they can be checked only for a given point in the bisection cube. While we can give necessary conditions to (6) in terms of bounds for second order derivatives, we need third order derivatives to give necessary conditions to (7). Therefore we perform the bisection algorithm until the cube is 'small' in comparison to geometric details and check condition (7) in the center of the cube.

If we are close to an edge, we want to switch to Newton's method again. Due to the fact just mentioned, we cannot apply Kantorovich's theorem. For reasons of numerical approximation, it is important to solve the first two equations of (5). The third one is used only to get a finite number of points on the edges. So we apply Newton's method to $f_1(x) = 0, f_2(x) = 0, s^T(x - c) = 0$ with the approximation $s = \nabla f_1(c) \times \nabla f_2(c)$ of the tangential vector evaluated in the center c of the cube. For this equation, we can apply Kantorovich's theorem using just bounds for second order derivatives. By defining an application specific, minimal distance for special points, we get one numerical solution per point of extreme coordinate.

Although the described algorithm cannot handle degenerated edges formed by two surfaces, we can handle solids with degenerated edges by using a third surface belonging to the same edge. E.g., if we put a sphere on top of a cylinder, we already need the plane terminating the cylinder. While the algorithm applied to the first two surfaces stops because of (6), it calculates the proper point, if it is applied to the third surface together with one of the first two.

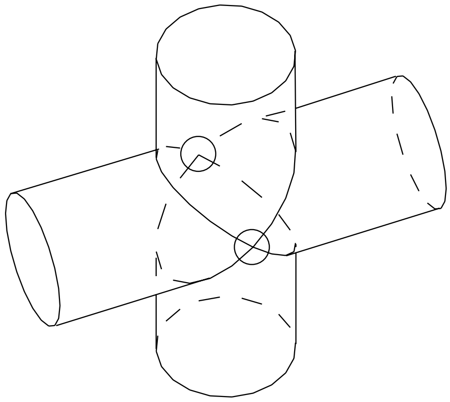

Figure 4: Two intersecting cylinders

3.3 Degenerated Points

A further kind of a special point is a degenerated point specified by

$$\begin{aligned} f_1 &= 0, \\ f_2 &= 0, \\ \nabla f_1 \times \nabla f_2 &= 0. \end{aligned} \tag{9}$$

This means that the tangential planes are parallel in a point of intersection. We are interested in isolated degenerated points, like seen in Figure 4, only. Degenerated edges fulfill equation (9) on the whole edge.

The classification of degenerated points needs a second order approximation. Therefore we introduce a local coordinate system spanned by the orthogonal unit vectors e_1, e_2 and e_3, such that e_3 is parallel to the common normal vector of f_1 and f_2. The surfaces f_1 and f_2 are approximated by the two, pure quadratic graphs

$$\xi_3^{(i)} = \frac{1}{2}(\xi_1\ \xi_2)A^{(i)} \begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix}, \quad i = 1, 2.$$

The matrices $A^{(i)}$ are given by the implicit function theorem as

$$A^{(i)} = \left[\frac{e_k^T \nabla^2 f_i e_l}{e_3^T \nabla f} \right]_{k,l=1,2}.$$

The intersection of the surfaces is approximated by the intersection of the approximations $\xi_3^{(1)} = \xi_3^{(2)}$. The intersection is classified by the eigenvalues of $A^{(1)} - A^{(2)}$. If both have the same sign, i.e., $\det\left(A^{(1)} - A^{(2)}\right)$ is positive, the intersection is only one point. If at least one eigenvalue is 0, the degeneration is of higher order, like in the case of a degenerated edge. If there are two eigenvalues of opposite sign, i.e., the determinant is negative, then two edges intersect in this point. The tangential vectors of both edges are given by the non-trivial solutions of the quadratic equation

$$(\xi_1\ \xi_2)\left(A^{(1)} - A^{(2)}\right)\begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix} = 0.$$

The bisection criteria for degenerated point calculation are analogous to the criteria for points of extreme coordinates. Only condition (8) is extended to all three components and the tests for non-degeneration differ. We check, whether $\det\left(A^{(1)} - A^{(2)}\right) \leq -\varepsilon$ in the center of the bisection cube is fulfilled.

4 Calculating Edges

The second step after special point calculation is edge detection. If we speak of a special point now, we mean a tuple (p_i, t_i) of a geometric point p_i together with the tangential vector t_i pointing in the direction of an edge starting in that point. There are always several special points within one geometric point.

For edge detection one has to choose an initial point for every edge and has to follow it, until the corresponding terminal point is reached. Then the edge is subdivided into segments of the demanded mesh size as good as possible. Selecting initial and terminal points is explained in §4.1 and edge following in §4.2.

4.1 Selecting Start Points

We start with two sets of special points, with the set of unconditional special points S^u and the set of conditional special points S^c. Unconditional points have to appear in the generated mesh, while conditional points are only used, if the edge would not be found without them. Points of intersection are of unconditional type, while points of extreme coordinates are conditional special points.

As long as S^u is non-empty, we choose one point from S^u as initial point on an edge. Otherwise, if only S^c is non-empty, we choose one of them. All special points within this geometric point are moved from S^c to S^u.

Starting in tangential direction, we follow the edge, until we reach the corresponding terminal point on the edge contained in S^u. If there are some conditional special points on the edge, they are removed from S^c.

When we finished one edge, the initial point and the final point are removed from S^u. As long as S^u or S^c are non-empty, we go on with the next edge.

4.2 Following Curves

Implicit curve following is mainly investigated for homotopy methods to solve nonlinear equations [12]. The principle is to start at a given point x^0 on the curve, and follow the curve in small steps until some terminal point on the curve is reached. The search is done by a predictor - corrector method. In the current point x^k, we compute the unit tangential vector t^k to the curve. Up to a scalar factor, it is the vector product of the normal vectors of the two surfaces defining the edge. As predictor, we use the point

$$\tilde{x}^{k+1} = x^k + \tau_k t^k$$

with the adaptively controlled step-length τ_k. This point is projected back onto the curve by the reduced Newton method to get the new value x^{k+1}.

On smooth curves, we have

$$\|x^{k+1} - \tilde{x}^{k+1}\| = O\left(\|\tilde{x}^{k+1} - x^k\|^2\right),$$

which can be used for adaptive step-length control. As long as the relation

$$\|x^{k+1} - \tilde{x}^{k+1}\| \leq c\|\tilde{x}^{k+1} - x^k\| \tag{10}$$

is not fulfilled, we halve the step-length τ_k and test a new $\tilde{x}^{k+1}$. Because the left hand side depends quadratically on the right hand side, (10) will be fulfilled for a sufficiently small τ_k. A proper choice is, e.g. $c = 0.1$. If condition (10) is fulfilled with $c/4$, we try with a doubled step-length for the next step.

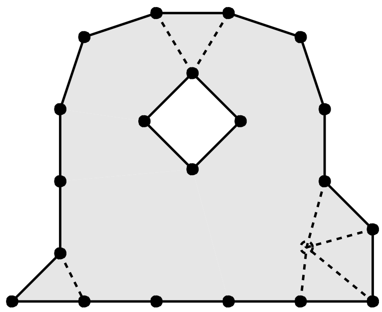

Figure 5: Example meshing problem

The points along the curve are stored in a list. As we have reached the terminal point, this approximative curve is subdivided into the segments of the prescribed grid-size as good as possible. The points generated are projected onto the exact edge.

5 Surface and Volume Mesh Generation

For a human being, 2D mesh generation is a boring, but trivial task. Let us analyze, how *we* might solve the following problem. Some plane domain is given in terms of a boundary mesh consisting of line segments. The goal is to fill the area with nearly equilateral triangles (see Figure 5). One might cut off the corner on the left hand side by one triangle. On the right hand side, one might fill in three triangles as sketched in the Figure 5. The new point should be chosen such that the shape of the triangles is optimal. On the top we can connect to the inner boundary by one triangle. We go on until the whole domain is meshed. But, how do we decide where to put a triangle? We recognize a specific image formed by the boundary elements and decide to cut off one or more triangles simultaneously. That is the task we have to teach the computer: If the boundary looks like a certain image, then cut off these triangles and get that new boundary.

This algorithm is the well known advancing front method, see e.g. [10], [14], [9], [8], [6]. The action performed for a certain boundary image is described by geometric rules. Our version of the advancing front method differs from others in the way, how the rules are applied. The approach is to separate the concrete rules form the rule application code. The algorithm has to check rules stored in data structures. Therefore the code complexity is independent of the number of rules. The algorithm is complicated, but well defined and can be, at least theoretically, implemented failsafe. Especially in 3D, the choice of the concrete rules is based on heuristics, which is put into an easily maintainable rule description database.

We proceed as follows. First, we describe the overall algorithm, we continue with the abstract rule description and the rule application algorithm. Finally, we concentrate on the extensions for surface and volume mesh generation.

5.1 The overall algorithm

The whole advancing front algorithm is stated in Figure 6. The input data to the mesh generator is the boundary description. It consists of the vector of n_P points

$$X = (X_1, \dots, X_{n_P}) \in \mathbf{R}^{D \times n_P},$$

```
load boundary mesh (starting front)
initialize quality classes
while front is not empty
        choose base-element from front
        get environment of base-element
        transform to local coordinate system
        test for applicable rules
        if a rule is applicable
                generate new points in local coordinates
                transform new points to global coordinates
                store new inner elements
                update front
        else
                decrease quality class for base-element
```

Figure 6: Overall algorithm

where $D = 2$ for plane meshing and $D = 3$ for surface and volume meshing, and the vector of n_{BE} boundary elements (as usual, $\mathbf{N}_n$ denotes the set of natural numbers $\leq n$)

$$R = (R_1, \ldots, R_{n_{BE}}) \in \mathbf{N}_{n_P}^{d \times n_{BE}}.$$

A simplicial boundary element is identified with the d indices of the corner points. For plane and surface meshing we have boundary segments ($d = 2$), for volume meshing the boundary elements are triangles ($d = 3$).

To every boundary element, the quality class, a natural number, is associated. It is initialized to one, which means highest quality.

The state of the preceding algorithm is always represented by the current boundary element vector. The algorithm performs as long as the length of the vector is positive.

We choose one of the boundary elements minimizing the criterion

$$\text{quality class} + \text{distance to boundary}.$$

This element will be called *base-element*. The distance to the boundary is determined by the minimal number of inner elements needed to draw an open path from the element to an arbitrary element of the original boundary. This distance term ensures a nearly uniform growing of the mesh over the whole boundary. The quality term prefers elements, on which better fitting rules can be applied.

The examination of the boundary is a local process. For the following steps, we have to know the environment of the base-element. This environment consists of all boundary elements closer to the base-element than a prescribed radius, and the accompanying corner points. Depending on the rules, the radius is chosen between three and five times the desired edge-length parameter h. This step is a geometric search process, it defines the asymptotic complexity of the whole algorithm. While one has to test each point of the current boundary in a straight forward approach, a quadtree or octree search tree for 2D or 3D, respectively, reduces the complexity of point searches to a logarithmic behavior. By the use of an alternating digital tree (ADT), also boundary elements can be found fast [2]. The

n_{LP} points and the n_{LBE} boundary elements in the environment are marked by the vectors l^P and l^E, respectively

$$X_{l^P(i)}, \quad 1 \leq i \leq n_{LP}, \qquad R_{l^E(i)}, \quad 1 \leq i \leq n_{LBE}.$$

Without loss of generality, we let $R_{l^E(1)}$ be the base-element and $X_{l^P(1)}$, $X_{l^P(2)}$ and, for the 3D case, $X_{l^P(3)}$, are the corners of the base-element. For the rest of the loop, we only have to work with a bounded number of points and boundary elements.

To simplify the following steps, we transform the n_{LP} points of the environment to a local coordinate system

$$x_i = F(X_{l^P(i)}), \qquad 1 \leq i \leq n_{LP}.$$

The coordinate transformation F is chosen such that $x_1 = 0$, x_2 is a point on the local ξ_1-axis and, for the 3D case, x_3 is a point in the local ξ_1-ξ_2-plane. If we choose unit-length in the local coordinate system corresponding to the desired edge-length h, the edge-length of well sized elements in local coordinates is about one. The various possibilities of the coordinate transformation will be discussed in Section 5.4. The n_{LBE} boundary elements in the environment are described by local boundary elements r_i using local point indices

$$l^P(r_{i,j}) = R_{l^E(i),j}, \quad 1 \leq i \leq n_{LBE},\ 1 \leq j \leq d$$

To the local boundary description the rule testing algorithm is applied. This key-step of our approach is described in Section 5.3. Here we only mention, that the tolerances of the rules are given by the quality class of the base-element. A high quality allows only good elements, which can make any rule application impossible.

If an applicable rule is found, it describes where to generate new points in local coordinates.

The n_{NP} new points $y_1, \ldots, y_{n_{NP}}$ are transformed to global coordinates and appended to the point vector

$$X_{n_P+i} = F^{-1}(y_i), \quad 1 \leq i \leq n_{NP}.$$

The rule describes the creation of new inner elements. After converting the local to global point indices, the elements are stored in the element list.

The rule also prescribes the necessary changes in the advancing front. Some boundary elements must be deleted, some others are added to the global boundary element vector R.

If no rule is applicable for the base element, its quality class is decreased. This enables either the application of a worse fitting rule later on, or a rule applied to a neighbor boundary element also removes the current one.

5.2 Abstract rule description

Next, the abstract description for 2D and 3D element generation rules is explained. Due to the coordinate transformation, 2D and surface meshing can be handled by the same rules. A rule is always specified in reference position, such that the elements have optimal shape. A rule involves n_{EP} existing points given by the vector x^R.

$$x^R = (x_1^R, \ldots x_{n_{EP}}^R) \in \mathbf{R}^{d \times n_{EP}}.$$

The connection of these points by elements is specified by the vector of existing boundary elements

$$r^R = (r_1^R, \ldots, r_{n_{EBE}}^R) \in \mathbf{N}_{n_{EP}}^{d \times n_{EBE}}.$$

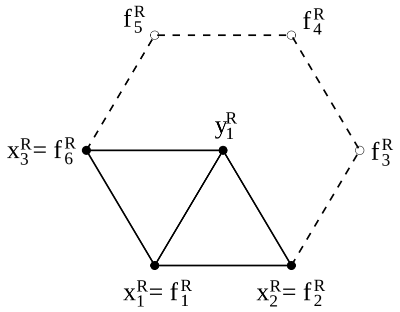

Figure 7: Example for old points, new point and free-zone, base element is (1, 2)

Some rules generate new points whose coordinates are given by the vector

$$y^R = (y_1^R, \ldots, y_{n_{NP}}^R) \in \mathbf{R}^{d \times n_{NP}},$$

and new boundary elements connecting existing and new points

$$s^R = (s_1^R, \ldots, s_{n_{NBE}}^R) \in \mathbf{N}_{n_{EP}+n_{NP}}^{d \times n_{NBE}}.$$

The corner with index i, $1 \leq i \leq n_{EP} + n_{NP}$ of the boundary element is x_i for $i \leq n_{EP}$, or $y_{i-n_{EP}}$, otherwise. The generation of n_{NE} inner elements is specified by the vector

$$t^R = (t_1^R, \ldots, t_{n_{NE}}^R) \in \mathbf{N}_{n_{EP}+n_{NP}}^{(d+1) \times n_{NE}}.$$

To be able to apply a rule, we need some free space containing no existing point and no existing boundary element. We define the free-zone by the convex hull of the n_{FP} points f_i^R combined to the vector

$$f^R = (f_1^R, \ldots, f_{n_{FP}}^R) \in \mathbf{R}^{d \times n_{FP}}.$$

An example for a rule filling a corner of 120 degree by two triangles is shown in Figure 7. This example is described by the data structure

$$\begin{aligned}
x^R &= ((0,0),(1,0),(-0.5,0.866)),\\
y^R &= ((0.5,0.866)),\\
r^R &= ((1,2),(3,1)),\\
s^R &= ((3,4),(4,2)),\\
t^R &= ((1,2,4),(1,4,3)),\\
f^R &= ((0,0),(1,0),(1.5,0.866),(1,1.732),(0,1.732),(-0.5,0.866)),
\end{aligned}$$

with constants $n_{EP} = 3$, $n_{NP} = 1$, $n_{EBE} = 2$, $n_{NBE} = 2$, $n_{NE} = 2$, $n_{FP} = 6$. Without loss of generality, we can always set $x_1^R = 0$, $x_2^R = (x_{2,1}^R, 0)$ and $r_1^R = (1, 2)$ for the 2D case and $x_1^R = 0$, $x_2^R = (x_{2,1}^R, 0, 0)$, $x_3^R = (x_{3,1}^R, x_{3,2}^R, 0)$ and $r_1^R = (1, 2, 3)$ for the 3D case.

The rules are specified for elements of optimal shape. But in a real situation, we will not have a corner of exact 120 degree. The real existing points will deviate form the reference position x^R. Let us denote the real existing points by x'. We define the deviation $u \in \mathbf{R}^{d \times n_{EP}}$ by

$$u = x' - x^R.$$

A deviation of the existing points should also lead to a movement of the new points. The simplest case is given by a liner mapping of movements. We generate new points at the position

$$y' = y^R + L_y u.$$

Analogously, the corners of the free-zone f^R are moved to f' by the linear mapping L_f

$$f' = f^R + L_f u.$$

For the 120 degree corner, the linear mapping

$$L_y u = -u_1 + u_2 + u_3$$

leads to two congruent, isosceles triangles. The linear mapping for the corners of the free-zone is constructed as follows. The points f_1', f_2' and f_6' have to match the points x_1', x_2' and x_3'. The other three are mirrored at the new point y_1'.

$$\begin{aligned}
f_1' - f_1^R &= u_1, \\
f_2' - f_2^R &= u_2, \\
f_3' - f_3^R &= 2L_y u - u_3 = -2u_1 + 2u_2 + u_3, \\
f_4' - f_4^R &= 2L_y u - u_1 = -3u_1 + 2u_2 + 2u_3, \\
f_5' - f_5^R &= 2L_y u - u_2 = -2u_1 + u_2 + 2u_3, \\
f_6' - f_6^R &= u_3.
\end{aligned}$$

Mention, it is not ensured to achieve properly oriented inner elements. In the case of wrong orientation, we reject the application of the rule.

5.3 The rule application algorithm

We need an algorithm applying rules defined by the data structure to a boundary image given in local coordinates. This is to find injective mappings

$$m^P : N_{n_{EP}} \to N_{n_{LP}}$$

and

$$m^E : N_{n_{EBE}} \to N_{n_{LBE}}$$

assigning each point (boundary element) in reference position a point (boundary element) in the environment of the base-element. In the 3D case, the three possible rotations of a triangle are handled by the mapping

$$m^O : N_{n_{EBE}} \to \mathbf{N}_3.$$

For the 2D case, we set $m^O = 0$. The mapping m^P defines the vector x' of local points in reference numbering by

$$x' = (x_{m^P(i)})_{1 \le i \le n_{EP}},$$

it also specifies the deviation u, the position of the new points in local coordinates y' and the corner points of the free-zone f'.

If the restrictions listed below are fulfilled, we can apply that rule.

- The quality class of the base-element must be worse then a prescribed value. This enables rules necessary for the termination of the algorithm, but which should be avoided if possible.

- Because of the compatible placement of the local coordinate system and the position of boundary element one in reference position we can set

$$m^E(1) = 1.$$

- The boundary in the rule description has to be topologically equivalent to its image in the local boundary, which means

$$m^P(r^R_{i,j}) = r_{m^E(i),j+m^O(i)}, \quad 1 \le i \le n_{NBE},\ 1 \le j \le d. \tag{11}$$

  To avoid additional complication, we assume to take the corner index modulo d.

- The point deviation must not exceed a limit depending on the quality class

$$|u_i| \le f^d(qual.\,cl.),$$

  where $f^d(.)$ is a monotone increasing function such that $f(x) \to \infty$ as $x \to \infty$. For example, we use $f(x) = 0.2x^2$.

- Each local boundary element not contained in the image of m^E must not intersect the free-zone.

- And finally, each generated inner element t_i must be properly oriented and the element error functional $E(t_i)$ measuring the well-shapedness of the element must be below a limit depending on the quality class

$$E(t_i) \le f^s(qual.\,cl.), \quad 1 \le i \le NE.$$

  Element error functionals will be discussed in Section 6. For example, we chose $f^s(x) = x$.

To construct the mappings m^P, m^E and m^O we essentially have to test each of the ${n_{LP}}^{n_{EP}} \times {n_{LBE}}^{n_{EBE}} \times 3^{n_{EBE}}$ possibilities. In an average 3D case we have $n_{EP} = 5$, $n_{EBE} = 3$, $n_{LP} = 30$ and $n_{LBE} = 50$ leading to $8.2 \cdot 10^{13}$ possibilities.

But by a good ordering of the trials, we can reduce the work to a realistic amount. It is advantageous to start with the mapping of elements. The algorithm performing this task for the 2D case is stated in Figure 8. It simulates n_{EBE} nested loops from 1 to n_{LBE}, but the loop for element ei is only started, if the elements $1, \ldots, ei-1$ are compatible mapped. Compatible is understood in the sense of (11).

Every loop mapping a boundary element connected to an already mapped one will find only one (or, in exceptional cases a few) hits. Only separated boundary elements in the rule description will produce nearly n_{LBE} hits. This gives a complexity of $O(n_{EBE} \times n_{LBE})$ for connected boundary elements and an additional factor n_{LBE} for each separated group.

The mapping of most points in the rule description is determined by the mapping of elements. Only the mapping of separated points in the rule description must be tried for each local point.

5.4 Possibilities of the transformation

The difference between plane and surface mesh generation is essentially contained in the coordinate transformations $F : S \cap U_{b.e.} \to \mathbf{R}^2$ used in the overall algorithm (Figure 6). It maps the environment $U_{b.e.}$ of the base-element from the surface S into the plane. In the plane the 2D rules are applied. If new points must be inserted, they have to be mapped by

```
ei = 2, m^E(1) = 1, m^E(2) = 0
while ei ≥ 2
    while m^E(ei) < n_LBE
        increment m^E(ei)
        if m^E(ei) compatible to m^E(1), ..., m^E(ei − 1)
            if ei < n_EBE
                increment ei
                m^E(ei) = 0
            else
                all elements are compatibly mapped !!!
    decrement ei
```

Figure 8: Element mapping algorithm

F^{-1} from the plane to the surface. The whole task is sketched in Figure 9. Because we can choose individual, local transformations, we do not have the problems of global coordinate transformations as non-uniform point distribution and singularities.

For small edge-length h compared to the radius of curvature, the transformation can be implemented numerically using the surface-operations defined in Section 3. For an edge-length of about the curvature radius, individual bijective transformations have to be implemented for every class of surface.

Beside surface mesh generation, the local coordinate transformation can be used for other mesh generation variants:

- Graded mesh generation can be achieved by varying the unit-length in the local coordinate system.
- (Local) anisotropic meshes can be produced by anisotropic coordinate transformations.

5.5 The rules in 2D

The nine rules used for 2D are drawn in Figure 10. The first seven rules have a free-zone to ensure enough space for further, well shaped elements. The last two ensure the termination of the algorithm. They have enough free space to build a new triangle, but do not take care of the remaining rest domain. To avoid the application of the last two rules at the beginning, the minimal quality class of them is set to a higher value (e.g. 5).

A dead lock cannot appear, because in 2D it is always possible to cut off a triangle without inserting a new point by one of the last three rules.

5.6 Closure for 3D mesh generation

In 3D, it is not always possible to cut off one element without inserting new points. These boundaries can cause difficulties to the advancing front method. An example taken from [8] is a pentahedron, which cannot be dissected into tetrahedra without inserting a point in the inner, see Figure 11.

We could not define rules for a robust handling of these type of geometries. Therefore it is necessary to recognize the problem, and to have an alternative algorithm. The problem

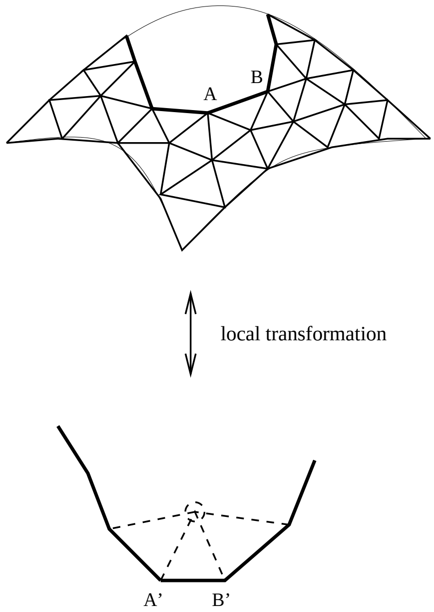

Figure 9: Surface mesh generation

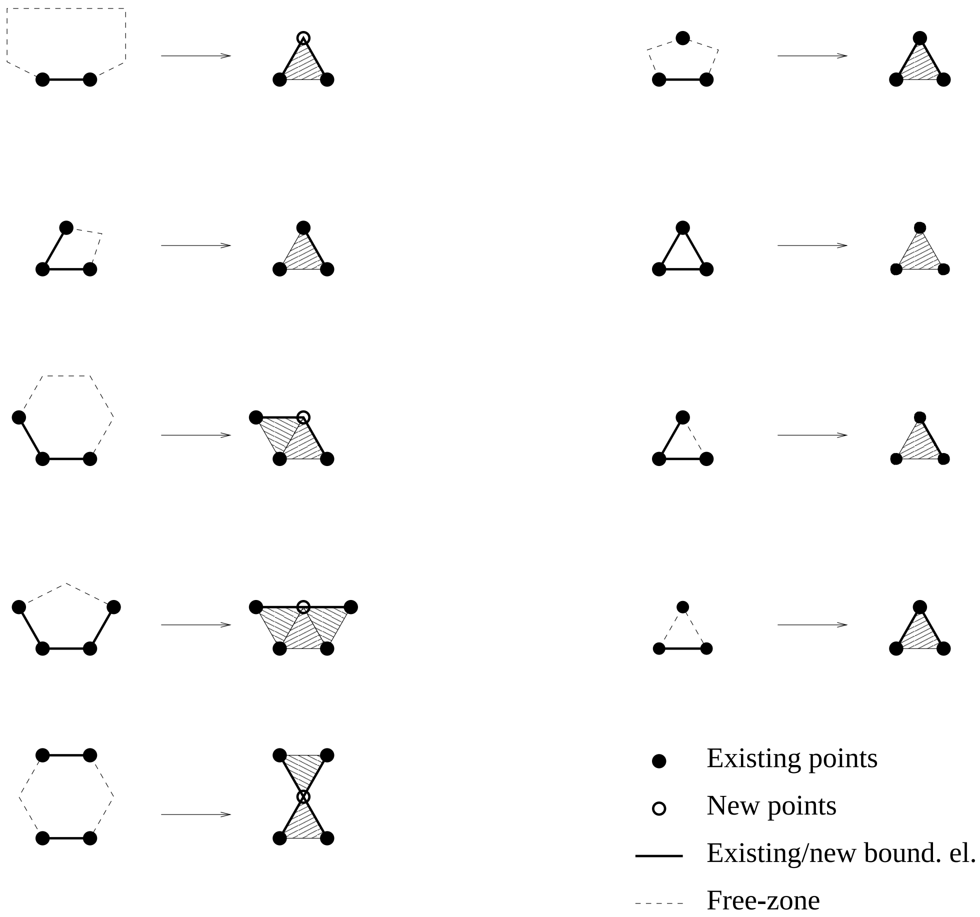

Figure 10: Rules for 2D

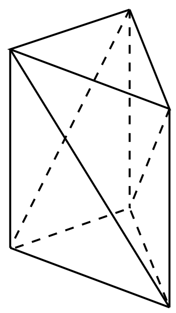

Figure 11: Example for a difficult boundary

is identified, if we cannot apply any rule up to a certain quality class. The alternative algorithm has to insert one point in the inner of the rest domain, such that elements can be built to this new point.

Experiments have shown, that in most of the exceptional cases star shaped rest domains can be achieved. For these domains, we can find an inner point, which can be connected to each boundary triangle by a tetrahedron. A star point x has to be in front of each boundary element of one rest domain, which means

$$n_i^T(x - X_{R_{i,1}}) \leq 0$$

has to be fulfilled for all boundary elements R_i with outer normal vectors n_i. The solution x is calculated by the Simplex method applied to the linear optimization problem

$$\min_{x \in \mathbf{R}^3} \max_i n_i^T(x - X_{R_{i,1}}).$$

6 Mesh Optimization

The quality of the generated mesh can be improved dramatically by several techniques of mesh optimization [3]. We distinguish between metric optimization, where points are moved to increase the quality, and topological changes to the mesh, where points are reconnected by different elements.

To quantify the quality of the mesh, we need an error functional for the elements. Because the standard finite element functional maximal edge-length over in-radius is expensive to calculate for 3D, we use a cheaper one dominating the standard functional. For a triangular element T we use the term

$$E(T) = \frac{\sqrt{3}}{36} \frac{(\sum_i l_i)^2}{Area} + \sum_i \left(\frac{l_i}{h} + \frac{h}{l_i} - 2 \right),$$

for a tetrahedral element we use

$$E(T) = \frac{1}{6^4\sqrt{2}} \frac{(\sum_i l_i)^3}{Vol} + \sum_i \left(\frac{l_i}{h} + \frac{h}{l_i} - 2 \right),$$

with the 3 or 6 edge lengths l_i, respectively. The constants are chosen such that an equilateral element has error 1. The first term of each functional penalizes flat elements, the second one too large or too small elements. As one element degenerates, the functional tends to infinity. If the orientation of the element nodes is negative, we set the error to infinity, too. The total error functional of the mesh $\mathcal{T}$ is the sum of the element errors

$$E(\mathcal{T}) = \sum_{T \in \mathcal{T}} E(T).$$

6.1 Metric optimization

In two dimensions, Laplacian smoothing is the standard mesh improvement algorithm. The inner points are moved into the center of gravity of the neighbors iteratively. Laplacian smoothing cannot be applied for surfaces, and it does not work well in 3D. Theoretically, the optimal topological equivalent mesh can be calculated by minimizing the error functional $E(\mathcal{T})$. A global minimization is too expensive, but we can apply point-wise relaxation of the error functional. Therefore we have to solve a set of minimization problems of maximal size three. We use the BFGS method, a member of the Broyden family (see [4]). It is globally convergent with locally quadratic convergence rate, although it does not need second derivatives.

One step of the relaxation method for an inner point is to solve the three dimensional problem

$$\min_{x \in \mathbf{R}^3} E_i(\mathcal{T}, x),$$

where $E_i(\mathcal{T}, x)$ is the global error functional $E(\mathcal{T})$ with the point X_i moved to x. Of course, only the elements connected to X_i have to be used in the optimization procedure. Because we start with a valid finite element mesh and the error of a wrong oriented element is defined as infinity, the mesh minimizing E_i is a valid mesh.

Points in the surfaces are optimized by the two dimensional problem

$$\min_{t \in \mathbf{R}^2} E_i(\mathcal{T}, F^{-1}(t)),$$

where F is the local transformation from the surface to the two dimensional parameter set. The point X_i is moved to $F^{-1}(t)$. Analogous points on edges are optimized by the solution of

$$\min_{t \in \mathbf{R}} E_i(\mathcal{T}, G^{-1}(t)),$$

with the local transformation G from the edge into the parameter interval.

6.2 Topological optimization

When point movement cannot increase the mesh quality anymore, some changes in the mesh topology may help a lot. Therefore a few elements are removed and the points are connected in a new manner. This actions may be described by local rules also.

The simplest technique working in two as well as in three dimensions is point collapsing (Figure 12). If the mesh quality increases, if two points are collapsed in the center and the elements between are removed, we perform this change to the mesh.

The common technique in two dimensions is edge swapping [5]. Often two flat triangles can be improved by swapping the common edge (Figure 12). We get two new triangles connecting the same four points in a different manner. To generalize edge swapping to three dimensions, we have to distinguish different cases. Five points can be connected by two as well as by three tetrahedra, see Figure 13. The optimizer can select the better one. Six points forming an octahedron can be cut into four tetrahedra in three different manners, where we

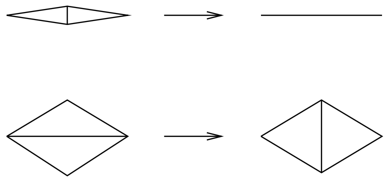

Figure 12: Point collapsing and edge swapping

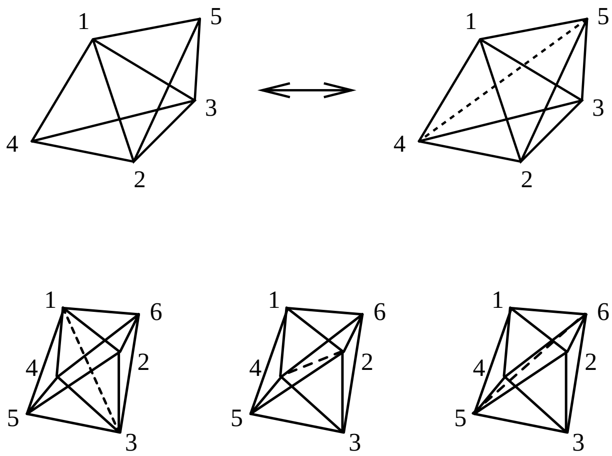

Figure 13: Face swapping in 3D

can choose the best one. The optimal connection is a valid mesh, because otherwise at least one tetrahedron with wrong orientation would occur and therefore the error functional is infinite.

7 Examples

Finally, we give several examples showing the performance of the mesh generator. The first example, called 'Cube and Spheres', shows a cube intersected by a sphere and by the complement of a sphere. The information given to the mesh generator consists of the text file shown in Figure 14 describing the geometry, and the desired mesh size parameter.

The 'plane' command defines half-spaces by a point in the plane and the outer normal vector. Spheres are defined by the center and the radius. By the Boolean operations, more complex objects can be formed and assigned to named solids. The solid called 'all' is the main object used by NETGEN.

The edges as well as the surface mesh of this example are drawn in Figure 15. First the convergence behavior of the bisection algorithm for special point calculation is investigated. There are 24 points of intersection (three in each of the eight corners) and also 24 points of extreme coordinates (four on each of the six circles). The number of cubes per level used for point of intersection and point of extreme coordinates calculation are shown in Table 1 and 2, respectively. We can see a bounded number independent of the level. The final mesh according to an edge-length parameter $h = 10$ consists of 1418 surface triangles, 1898 tetrahedra and 813 points. The classification of the elements with respect to the quality measure $E(T)$ is drawn in Figure 16. The left hand side shows the quality before, the right hand side after optimization. CPU times on a SUN Ultra 1 workstation for the individual

```
solid cube =
          plane (0, 0, 0; 0, 0, -1)
      and plane (0, 0, 0; 0, -1, 0)
      and plane (0, 0, 0; -1, 0, 0)
      and plane (100, 100, 100; 0, 0, 1)
      and plane (100, 100, 100; 0, 1, 0)
      and plane (100, 100, 100; 1, 0, 0);
solid all =
          cube
      and sphere (50, 50, 50; 75)
      and not sphere (50, 50, 50; 60);
```

Figure 14: Input to 'Cube and Spheres'

Level	1	2	3	4	5	6	7	8	9	10	11
Intervals	1	7	19	56	200	664	1712	1824	1896	1128	384

Table 1: Cubes per level for point of intersection calculation

subproblems are given in Table 3.

To apply efficient geometric multigrid methods, we require a coarse mesh, which will be hierarchically refined. Our version of the advancing front method is well suited to the generation of very coarse meshes, because by the abstract rules many special cases can be handled without coding. For the same object, a mesh with edge-length $h = 50$ was generated consisting only of 124 surface triangles, 74 tetrahedra and 54 points.

A second example is the crankshaft drawn in Figure 17. The mesh consists of 592 surface elements, 853 volume elements and 335 mesh nodes. Figure 18 shows the quality classes of the volume elements. The total CPU time need for the generation of the mesh was 140 seconds. The final example is a surface mesh (Figure 19) of some Sculpture in St. Gallen.

8 Concluding Remarks

In this paper, we proposed the separation of element generation rules from the code. The functionality of the implemented code is precise specified, while all the heuristics of the rules is moved to an extern rule description. Tools for graphical editing of the rules may be developed. By means of these tools it should be possible to develop a proper set of rules for mixed tetrahedral, pentahedral and hexahedral mesh generation.

We can use the same rules for plane and surface mesh generation as well as for uniform, graded or anisotropic mesh generation. The distinction is contained in two transformations.

Level	1	2	3	4	5	6	7	8	9	10	11
Intervals	1	7	19	56	200	688	2120	2880	3048	2928	2976

Table 2: Cubes per level for point of extreme coordinates calculation

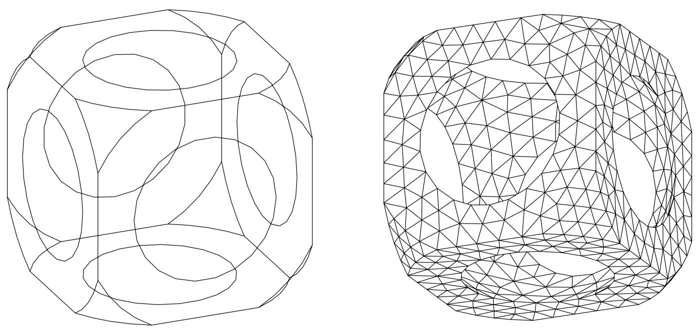

Figure 15: Cube and Spheres

Step	Points	Edges	Surface	Surf. Opt.	Volume	Vol. Opt.	total
Seconds	5	1	13	15	220	150	394

Table 3: CPU times for individual subproblems for 'Cube and Spheres'

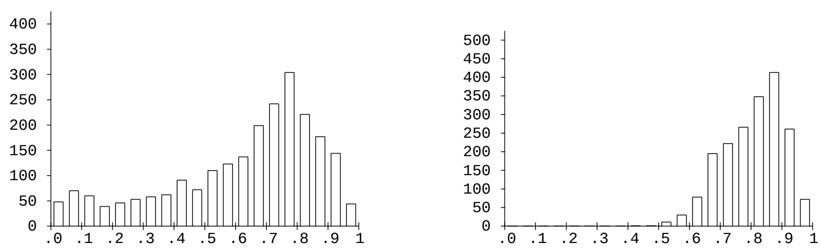

Figure 16: Quality classes for 'Cube and Spheres'

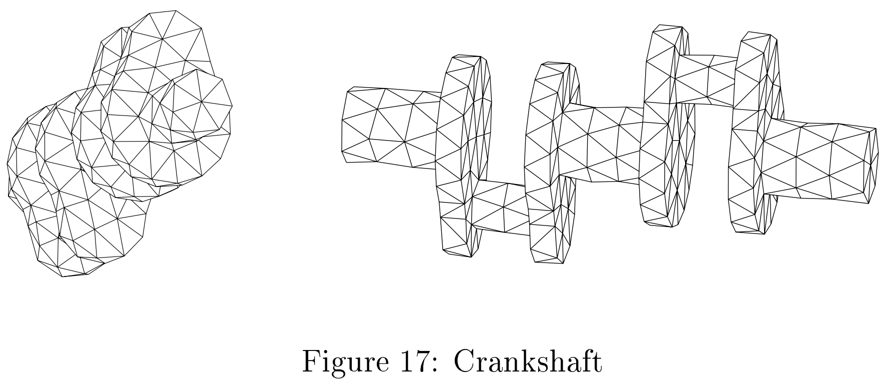

Figure 17: Crankshaft

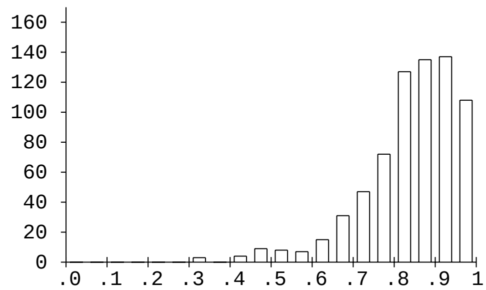

Figure 18: Quality classes for crankshaft

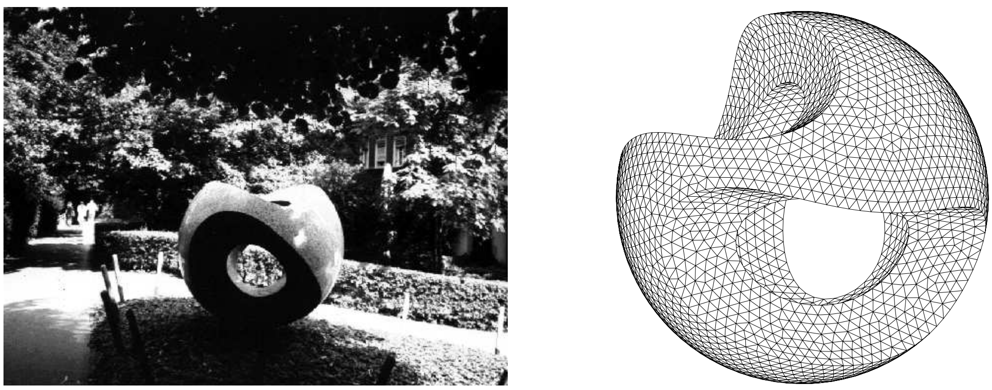

Figure 19: Sculpture in St. Gallen and surface mesh

At current time we have implemented only two dimensional graded mesh generation.

The corner-, edges- and surface mesh generation algorithms are based on abstract geometric primitives. Therefore new geometric objects can be implemented without changing the main code.

The software as well as the examples are available via anonymous ftp from the address:

ftp.numa.uni-linz.ac.at
/pub/software/netgen.tar.gz

References

[1] T. J. Baker. Developments and trends in three-dimensional mesh generation. *Appl. Numer. Math.*, 5:275–304, 1989.

[2] J. Bonet and J. Peraire. An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems. *Int. J. Numer. Methods Eng.*, 31:1–17, 1991.

[3] E. B. de l'Isle and P. L. George. Optimization of tetrahedral meshes. In I. Babuska, J. E. Flaherty, W. D. Henshaw, J. E. Hopcroft, J. E. Oliger, and T. Tezduyar, editors, *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, volume 75, pages 97–127. Springer, New York, 1995.

[4] R. Fletcher. *Practical Methods of Optimization, Volume 1: Unconstrained Optimization.* John Wiley & Sons, Chichester - New York - Brisbane - Toronto, 1980.

[5] W. H. Frey and D. A. Field. Mesh relaxation: A new technique for improving triangulations. *Int. J. Numer. Methods Eng.*, 31:1121–1133, 1991.

[6] P. L. George and E. Seveno. The advancing-front mesh generation method revisited. *Int. J. Numer. Methods Eng.*, 37:3605–3619, 1994.

[7] J. Hoschek and D. Lasser. *Grundlagen der geometrischen Datenverarbeitung.* Teubner, Stuttgart, 1989.

[8] H. Jin and R. I. Tanner. Generation of unstructured tetrahedral meshes by advancing front technique. *Int. J. Numer. Methods Eng.*, 36:1805–1823, 1993.

[9] B. P. Johnston and J. M. Sullivan, Jr. A normal offsetting technique for automatic mesh generation in three dimensions. *Int. J. Numer. Methods Eng.*, 36:1717–1734, 1993.

[10] S. H. Lo. A new mesh generation scheme for arbitrary planar domains. *Int. J. Numer. Methods Eng.*, 21:1403–1426, 1985.

[11] M. E. Mortenson. *Geometric Modeling.* John Wiley & Sons, New York, 1985.

[12] H. Schwetlick. *Numerische Lösung nichtlinearer Gleichungen.* Deutscher Verlag der Wissenschaften, Berlin, 1979.

[13] J. f. Thompson and N. P. Weatherill. Aspects of numerical grid generation: Current science and art. Technical report, NSF Engineering Research Center for Computational Field Simulation, Mississippi State University, and Department of Civil Engineering, University College of Swansea, UK, 1993.

[14] J. Z. Zhu, O. C. Zienkiewicz, E. Hinton, and J. Wu. A new approach to the development of automatic quadrilateral mesh generation. *Int. J. Numer. Methods Eng.*, 32:849–866, 1991.