An Automatic Mesh Generator using Geometric Rules for Two and Three Space Dimensions

Joachim Schöberl

Abstract

A kind of moving front technique mesh generator is presented. A given boundary mesh is extended into the domain in two or three dimensional space. The user can specify rules which describe the generation of the interior elements for a given boundary image.

1 Introduction

A lot of engineering disciplines require a partial differential equation modeling. Two of the most popular methods to treat these p.d.e.s numerically are the finite element method and the finite volume method. Both require a partitioning of the domain of interest into a set of simple domains, called (interior) elements. All elements together form the mesh. In the plane triangles and quadrilaterals are used, in three dimensional space tetrahedrons, hexahedrons and pentahedrons are the most popular elements. It is a desirable goal to do this partitioning automatically. There are several approaches which work quiet well in the plane but make a lot of difficulties in three dimensions. In [1] a wide bandwidth of algorithms is presented.

One of them is the moving front technique. The method requires that the boundary of the domain is given by a set of boundary elements. The method puts one layer of interior elements onto the boundary elements. So a smaller domain is received, which is reduced by the same principle until nothing is left to do. This reduction can be done for a whole layer at once. In [5] a description of this technique is given. In this paper a local variant is described. One step of the algorithm cuts off one (or a few) elements from the remaining domain.

An advancing front mesher has to decide which interior elements should be created for a given boundary image. This tests may be coded into a programme, but I guess this will be a long and error vulnerable programme, especially for the three dimensional case. The suggestion is to split the whole task into rules, which describe what to do in a certain situation and into a part of code, which has to apply these rules. The rules may be specified in a simple programming language or graphical tools can be implemented to edit the rules. It is described what these rules have to contain.

A full automatic three dimensional mesh generator must only use geometric information to build the mesh. In section 2 a few possibilities to geometric modeling are given. The first non trivial problem is to find special points of the solid. It should be a general method which can be applied to different geometric models. In section 3 a combination of bisection and Newtons method is presented. Starting at the special points the edges have to be detected. This task is shortly mentioned in 4. The main part of the paper deals with free meshing, which is used for surfaces and volume.

2 Geometric Modeling

The first step in mesh generation is to describe the solid in a mathematical way. Solid modeling is a wide area in research over the last decade and still very active. There are a lot of different approaches to describe a solid. We give two possibilities:

2.1 Algebraic Modeling

A lot of solids can be described by algebraic inequalities. E. g. the inequality

$$(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 \leq r^2$$

describes a sphere with center (c_x, c_y, c_z) and radius r. Also a cylinder with infinite length can be described by a second order algebraic inequality. A halfspace, e. g. $x \geq 0$, is given by a linear inequality. These solids, which are given by a simple inequality are called primitives. Out from these more complicated solids can be formed by the Boolean operations union, section and subtraction. A cylinder of finite length is the result of the section of a cylinder of infinite length with two halfspaces:

$$\begin{aligned} cyl &= \left\{y^2 + z^2 \leq 1\right\} \\ h_1 &= \{x \geq 0\} \\ h_2 &= \{x \leq 10\} \\ all &= cyl \cap h_1 \cap h_2 \end{aligned}$$

In a computer a solid is represented by a binary tree. The leaves are the primitives. It is simple to build solids from these primitives, but the usage is very restricted. It is difficult to plot these solids. Finding edges is to find the nullset of two algebraic equations in three variables. It is simple to check if a given point is inside or outside the solid.

2.2 Spline Modeling

Bezier splines and B - splines are widely used in Computer Aided Geometric Design (CAGD), e. g. for designing aircraft wings or bodyworks. A curve or a surface is locally approximated by a polynomial. As basis functions for Bezier splines Bernstein polynomials are used. The coefficients, called Bezier points for Bezier splines or d'Boor points for B - splines, are points in the space which have a geometric meaning. Also rational functions are used. With the help of them spheres and cylinders can be represented exactly. Because the surfaces are given explicitly, it is simple to plot them. But it is difficult to work with solids bounded by spline patches and implement an is-in - test. To implement operations on such solids two facts are important:

- The *convex hull property:* The spline surface (curve) is contained in the convex hull of the surrounding Bezier respectively. d'Boor points. So a simpler (and worse) approximation is given by a polyeder.

- *zoom in:* It is possible to dissect a patch in a few smaller patches of the same degree. So the approximation of the Bezier or d'Boor points becomes better.

A textbook about spline modeling is e. g. [4].

2.3 Operations on Solids

The mesh generator should be independent of the kind of surface representation used. Therefore the allowed operations are very limited. We will use a combination of global bisection algorithms and local Newton type algorithms. For the bisection algorithms one needs tests like ' is a given box inside a solid ? ' The possible answers are ' no ', ' yes ' or 'maybe'. The answer 'maybe' is always correct, but it is only allowed if the distance of the box to the boundary is of the order of the diameter of the box. This can be decided easily for algebraic primitives (e. g. if the distance from the center of the box to the center of a sphere is less than the radius of the sphere minus the half diameter of the box, than it is sure that the box is inside the sphere). For spline surfaces the test can be done for the d'Boor polyeder after a local zoom until the accuracy is of the demanded dimension. The result of the Boolean operations are the following:

$\cap$	no	yes	maybe
no	no	no	no
yes	no	yes	maybe
maybe	no	maybe	maybe

$\cup$	no	yes	maybe
no	no	yes	maybe
yes	yes	yes	yes
maybe	maybe	yes	maybe

$\setminus$	
no	yes
yes	no
maybe	maybe

The bisection type algorithms assures global convergence, but they are slow if one is interested in exact coordinates of crosspoints. If one is near to a crosspoint, it can be calculated fast and precise by Newton's method. The thing one has to check if one has a start point in the convergence area of Newton's method. This can be decided by the surface curvature and by the angles between the surfaces to intersect. For Newton's method one needs the normal vector to the surface and for the convergence test the curvature. These information are available for algebraic surfaces and are locally available for spline patches.

3 Special Points

The mesh generator meshes volumes from its boundary, the surfaces, it meshes surfaces from their boundaries, the edges and meshes edges from their boundaries, points, which we call special points. There are different types of special points. The most common are the points, where three surfaces intersect (see fig. 1). These points we call crosspoints. Their computation is explained in section 3.1. There are edges without natural start and endpoint (see fig. 2). To get unique defined points on these curves one can choose these with extremal x, y or z coordinates. They are called extremalpoints and their computation is shown in section 3.2. There are still more types of special points which came from degeneration in some sense. They are discussed in 3.3. In common with the special points tangents to the edges are calculated.

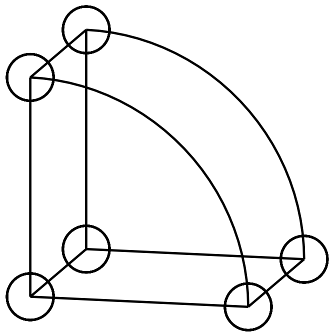

Figure 1: Examples for Crosspoints

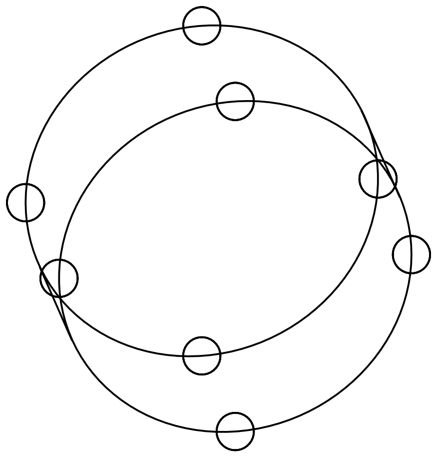

Figure 2: Examples for Extremalpoints

3.1 Crosspoints

Crosspoints are isolated points, where three surfaces intersect. For implicit given surfaces these are the isolated roots of the system

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

For parametric surfaces the crosspoints are given by the system

$$\begin{aligned} X_1(s_1,t_1) &= X_2(s_2,t_2) \\ X_1(s_1,t_1) &= X_3(s_3,t_3) \\ Y_1(s_1,t_1) &= Y_2(s_2,t_2) \\ Y_1(s_1,t_1) &= Y_3(s_3,t_3) \\ Z_1(s_1,t_1) &= Z_2(s_2,t_2) \\ Z_1(s_1,t_1) &= Z_3(s_3,t_3) \end{aligned} \tag{2}$$

For a combination of different types also equations can be specified. These are polynomial equations with three respectively six unknowns and as many equations. One has to find all roots. There are algebraic (see [6]) and also numerical (see [8]) algorithms for doing this, but they are complicated and computationally expensive.

Our approach uses more geometric information. We consider a box in which the whole solid is contained. Then we subdivide this box into eight boxes of the half size and so on for each new box. In each box we examine the corresponding part of the solid. We need a criterion to decide, whether there is a crosspoint in the box or not. If there is no crosspoint in it we can stop this branch of the recursive algorithm. If the tests are sharp enough this method finds all crosspoints with an arbitrary precision. But it is slow if one is interested in precise coordinates. It needs at least eight tests to get one more binary digit in the solution. If one is close to a solution and has information about the distance to the surfaces and normals to the surfaces one can start Newton's method to solve the equations above. To assure one is in the convergence area of Newton's method one needs information about the curvature of the surfaces.

3.1.1 Localization

To apply the above method one has to know the shape of the solid restricted to a box. Of course one could represent the local shape by the original solid tree, but this does not lead to a simple local model. The simple local model model should be of the type 'empty', full', or 'solid'. The localization can be done recursive:

- For a *primitive* the local model is 'empty' if the whole box is outside the primitive, it is 'full', if the whole box is included in the primitive, and it is 'solid', i. e. the primitive, if the box intersects the boundary or if we cannot decide.

- For the Boolean operations the localization can be expressed by the localization of the operands:

$\cap$	empty	full	solid 1
empty	empty	empty	empty
full	empty	full	solid 1
solid 2	empty	solid 2	solid 1 $\cap$ solid 2

$\cup$	empty	full	solid 1
empty	empty	full	solid 1
full	full	full	full
solid 2	solid 1	full	solid 1 $\cup$ solid 2

$\setminus$	
empty	full
full	empty
solid	$\setminus$ solid

If the box - in - solid tests fulfill Proposition ??? and the solid fulfills ??? than the local models of the successively refined boxes converge to the local model of the limit point.

3.1.2 Crosspoint - in - Box - Test

A necessary condition to have a crosspoint in a box is that their are at least three surfaces in the local model. If all existing crosspoints are regular in the following sense, then the number of boxes in each refinement level are limited by a constant. A crosspoint is regular, if the matrix J formed by the normal vectors to three intersecting surfaces is regular. E. g. the matrix is singular if two surfaces have the same normals. The upper bound for the number of boxes per level grows with the condition of this matrix. The computational cost of the bisection algorithm for regular crosspoints is not too bad, because each digit more in precision costs the same price. Some examples for the number of boxes per level are given in section 3.4. If a crosspoint is singular, the number of boxes fulfilling the necessary condition grows exponential with the refinement level. This inefficiency is due to the ill posedness of the equation. An arbitrary small perturbation in one of the functions can change the structure of the solution: It may disappear or it may split up into two crosspoints. A mathematical correction to this problem is to solve the equations $f_i = f_j = \det J = 0$ instead of the original system, where the indices i and j correspond to two surfaces with not colinear normals. This system is well conditioned if the degeneration is of first order. But this system is much more complicated and does not fit into our concept. The bisection algorithm works optimal if there are more than three cutting surfaces and the rank of the matrix consisting of the normal vectors is three. So we make the user responsible to do the geometric modeling in a stable way. If the condition of the Matrix J gets bad we expect that the root of the system has algebraic dimension one and will not calculate it.

3.1.3 Speed up by Newton's Method

If we are close to a root of the system (1) and if we have information about the derivatives of the functions f_i we can use the fast convergent Newton method. With the writing $F(x) = (f_1(x,y,z), f_2(x,y,z), f_3(x,y,z))$ and the starting point x^0 in the center of the box we can do the iteration for $k = 0, 1, 2, \ldots$:

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

The Newton method has an intuitive geometric interpretation. In the limit the components of F measure the distance to the corresponding surface and the columns of the matrix F' are colinear to the normals. The new point x^{k+1} is the projection onto the crosspoint of three plains given by the normals and the distance to x^k. The iteration is well defined if the matrices F' are regular. This is again the case if the crosspoint is regular. By continuity arguments the regularity of F' in an environment of the crosspoint follows.

If one deals with spline patches it is in general difficult to calculate the nearest point on the surface to a given point. If one is close to the surface in comparison with the curvature radius the distance function is convex over the parameter domain and can easily be minimized. With the help of the bisection algorithm we must come close enough to the surface. Then we can do Newton's method. The components of F are the distances of the point to the corresponding surface and the columns of F' are the normalized vectors from x^k to the point on the surface, which are equal to the normal vectors.

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

Theorem 1 (Kantorovich) *Let $F : D \to Y, D \subset X open$, F is differentiable on D and F' is Lipschitz continuous in D with*

$$\|F'(y) - F'(x)\| \leq \gamma \|y - x\| \qquad \forall x, y \in D$$

For a given $x^0 \in D$ let $F'(x^0)$ be regular and

$$\|F'\left(x^0\right)^{-1}\| \leq \beta$$

and

$$\|F'\left(x^0\right)^{-1} F\left(x^0\right)\| \leq \eta$$

If

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

and

$$B\left(x^0, t_*\right) \subset D$$

with

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

then Newton's method with starting point x^0 is well defined and x^k converges against a root of F in $B\left(x^0, t_*\right)$.

Remark 1 *The root is unique in*

$$B\left(x^0, t_{**}\right)$$

with

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

The proof of theorem 1 and remark 1 can be found in [9]. If F is smooth enough than the Lipschitz constant γ can be estimated by the norm of the second derivative of F, which is a measure for the curvature of the surfaces. The norm of $\left(F'\right)^{-1}$ is a measure of the well posedness of the crosspoint. For all involved constants estimations from above can be computed and therefore a test for Newton convergence can be implemented.

3.2 Extremalpoints

There are edges without crosspoint on them (see fig ???). One also needs uniquely defined points on them. A possibility is to choose the points with maximal x coordinates. Finding them is to solve a constraint optimization problem (f_1 and f_2 define the two intersecting surfaces implicitly):

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

The corresponding Lagrange functional is:

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

Necessary conditions for extremalpoints are the Kuhn Tucker conditions of first order:

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

Eliminating the Lagrange multipliers we get the system:

$$\begin{aligned} f_1 &= 0 \\ f_2 &= 0 \\ f_3 := \frac{\partial f_1}{\partial y}\frac{\partial f_2}{\partial z} - \frac{\partial f_1}{\partial z}\frac{\partial f_2}{\partial y} &= 0 \end{aligned} \tag{3}$$

When an edge is parallel to the y-z plane every point on it fulfills the necessary Kuhn - Tucker conditions of first order. So we demand the sufficient Kuhn Tucker conditions of second order:

$$\forall s \in \overline{C}, s \neq 0 : s^t \nabla^2 \mathcal{L}\left(x, \lambda\right) s > 0 \tag{4}$$

$\overline{C}$ is the tangential cone, which is in our case the one dimensional space spanned by $\nabla f_1 \times \nabla f_2$. If we demand that the left side of (4) is greater than $k\|s\|^2$ we get only extremal points with x - curvature greater than k. In practical computation one may set k to 10^{-6}.

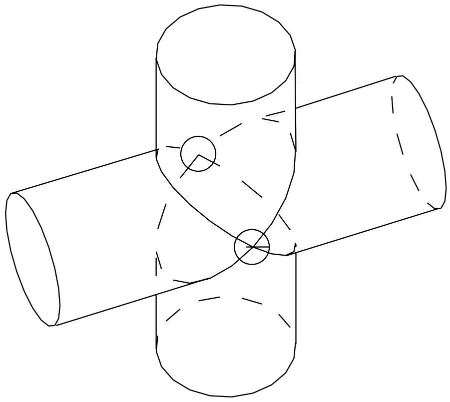

Figure 3: Two intersecting cylinders

3.2.1 Bisection Criteria

For extremalpoint calculation the first step is the bisection process again. A necessary condition to have an extremalpoint in a box is that their are at least two surfaces in the localization of the solid. This condition is fulfilled by $O\left(d^{-1}\right)$ boxes, where d is the diameter of the boxes. If one would only use this condition the number of boxes would double level by level. One has to make use of the third equation in (3). Applying the average lemma in the form

$$\|f(x) - f(x^*)\| \leq \|x - x^*\| \sup_{\xi \in [x,x^*]} \|f'(\xi)\|$$

to f_3, substitute for x the center of the box and demand $f(x^*) = 0$, write d for the diameter of the box, calculate f_3':

$$f_3' = \nabla^2 f_1 \begin{pmatrix} 0 \\ \frac{\partial f_2}{\partial z} \\ -\frac{\partial f_2}{\partial y} \end{pmatrix} + \nabla^2 f_2 \begin{pmatrix} 0 \\ -\frac{\partial f_2}{\partial z} \\ \frac{\partial f_2}{\partial y} \end{pmatrix}$$

we get a third necessary condition:

$$\|f_3(x)\| \leq \frac{d}{2}\left(\|\nabla^2 f_1\|\|\nabla f_2\| + \|\nabla^2 f_2\|\|\nabla f_1\|\right) \tag{5}$$

This condition is always fulfilled if the x coordinate is constant on an edge, but then the second Kuhn Tucker condition is violated, i. e. the x component of the curvature is 0. The curvature may be approximated by the curvature of the curve through the center of the box and specified by $f_1 = const$ and $f_2 = const$. This is computable with the help of the implicit function theorem. Condition (5) together with the check of the curvature ensures that the number of boxes per level is bounded.

3.3 Degenerated Points

A further kind of special point is a degenerated point in the following sense:

$$\begin{aligned} f_1 &= 0 \\ f_2 &= 0 \\ \nabla f_1 &= \lambda \nabla f_2 \end{aligned} \tag{6}$$

with any $\lambda \neq 0$. This means that the normals are colinear in a point of the common zeroset.

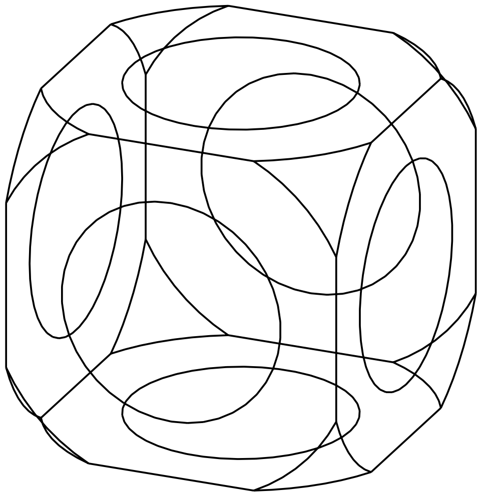

Figure 4: Edges of Example 1

3.3.1 Tangents to degenerated points

To get information about the tangents to the degenerated point one has to consider second derivatives. Therefore we define a local coordinate system (ξ, η, ζ) s. t. ζ is colinear to the common normals and ξ and η lie in the common tangential plane. In this coordinate system both surfaces can be approximated by the graphs of two pure quadratic functions:

$$\begin{aligned} \zeta_1 &= (\xi\ \eta)\, A\, (\xi\ \eta)^t \\ \zeta_2 &= (\xi\ \eta)\, B\, (\xi\ \eta)^t \end{aligned}$$

Demanding $\zeta_1 = \zeta_2$ one gets the equation

$$(\xi\ \eta)\, C\, (\xi\ \eta)^t = 0 \tag{7}$$

with $C := A - B$. This equation has no nontrivial solution if C is definite, it has one solution if C is semidefinite, namely the vector in the null space of the matrix. If C is indefinite and $c_{11} \neq 0$ one gets the quadratic equation

$$\left(\frac{\xi}{\eta}\right)^2 + 2\frac{c_{12}}{c_{11}}\frac{\xi}{\eta} + \frac{c_{22}}{c_{11}} = 0$$

with two different real solutions. If $c_{11} = 0$ and $c_{22} \neq 0$ one can exchange the role of ξ and η. If both are zero the solution is $(1\ 0)^t$ and $(0\ 1)^t$

3.4 Examples

A first example is a cube of edgelength 100, intersected with a sphere of radius 75 and the inverse of a sphere of radius 60. All of them have the same center. The calculated edges are shown in figure 4. There are 24 crosspoints (3 in each of the 8 corners) and 24 extremalpoints (4 on each of the 6 circles). The number of boxes per level for crosspoint and extremalpoint calculation are shown in table 1 and table 2 respectively.

4 Calculating Edges

The second step after special point calculation is edge finding. A special point is always a geometric point together with a tangent. There are always more special points within the

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

Table 1: Number of Boxes per Level for crosspoint bisection

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

Table 2: Number of Boxes per Level for extremalpoint bisection

same geometric point. One has to choose a start point for the edge and has to follow it until one reaches the corresponding end point. When one knows the total length of the edge it is subdivided into pieces of the length of about the demanded gridsize. Selecting start and end points is explained in section 4.1 and edge following in section 4.2.

4.1 Selecting Start Points

One can subdivide the special points into conditional and unconditional special points. In unconditional ones mesh nodes must be. These are e.g. crosspoints. Conditional ones will only be used if there are no unconditional special points on an edge. It is no need that a mesh node is exactly in an unconditional special point. Extremalpoints are examples for this kind.

So one has to choose unconditional special points as start point as long there are some available. On following the edge on has to check if there lies a conditional point on it. If there is one, it has to be deleted. The end of the edge is reached if there is a unconditional special point with a tangent in the opposite direction on it. After finishing the edge the start and end point are deleted from the special point list. If there are only conditional special points available one has to start from one of these. The other (conditional) special points with the same coordinates must be changed into unconditional such that they will be recognized as end points

4.2 Following Curves

Curve following is mainly investigated for homotopy methods used for nonlinear equations. The principle is always to move a certain length forward into the direction of the tangent and come back onto the curve. Two questions are:

- How far can one move without danger to loose the curve ?
- How precise must one come back onto the curve ?

For the first point there are step length control mechanism which compare the estimated deviation from the curve with the actual. To come back to the curve one can use a fast Newton like method. But it is not necessary to do it within machine precision. It is enough to stay within a certain environment of the curve. For these algorithms see [9].

The points along the curve are stored in a list. After reaching the end point this approximative curve is subdivided into the demanded gridsize as good as possible. The points generated now are projected to the edge with machine precision.

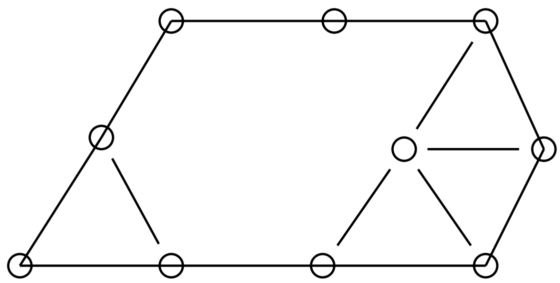

Figure 5: Example meshing problem

5 Surface and Volume Meshing

In this chapter the common strategy for plane, surface and volume meshing is described. Whenever it is enough things are formulated for the plane case, generalization to the other cases are mentioned.

Let us discuss how a thinking individual might precede in the following example. A boundary mesh is given by a set 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 with one triangle. On the right one could fill in three triangles like it is sketched in the figure. The new point could be chosen such that the shape of the triangles is optimal. 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 that image, than cut off these triangles and get that new boundary. This sequence may be described with the help of geometric rules. There are a few rules shown in figure 6. If there is a sharp corner of about 60 degree, it may be cut off by a triangle. A bay of consisting of three line segments may be filled by three triangles. A new point has to be inserted somewhere in the center. A very important rule is the third one. If three lines form a triangle, it may be filled and nothing is left.

5.1 Algorithm

Now we can discuss the whole algorithm. It is stated in figure 7. The input to the mesher are the boundary elements. These are lines for plane and surface meshing and triangles or quadrilaterals for volume meshing. They are stored in an array. For 3D meshing it is important that no linear ordering of the elements is assumed. The element data contains the indices of the accompanying points. The points are stored in the point array. The state of the preceding algorithm is always represented by the actual set of boundary elements.

Now go on as long as there are boundary elements available. Choose one of the elements. It is important to know the local environment of this selected element. Because the elements are stored in a unstructured set this is an geometric search process. Here the asymptotic time complexity of the algorithm comes in. If one uses geometric search trees the search can be done in logarithmic time, otherwise it takes time proportional to the number of boundary elements. The rest of the loop has to be done for an in average constant number of elements.

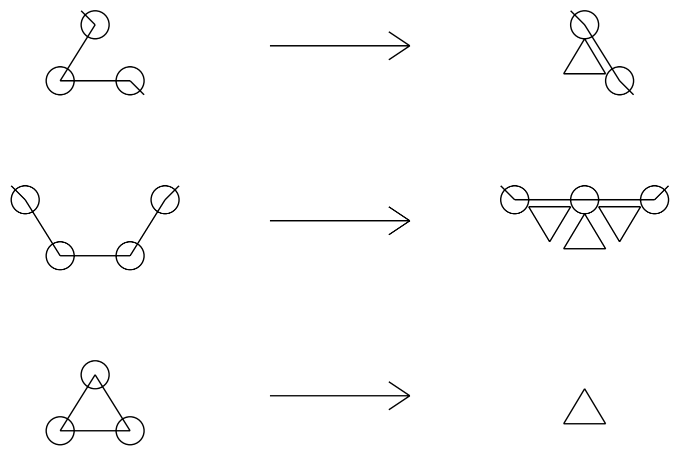

Figure 6: some rules

load boundary elements
set tolerance classes
while boundary is not empty

> choose boundary element
> get environment
> transform to reference position
> test for applicable rules
> if an rule is applicable
>
> > store new nodes and interior elements
> > get new boundary
>
> else
>
> > increase tolerance class for boundary element

Figure 7: overall algorithm

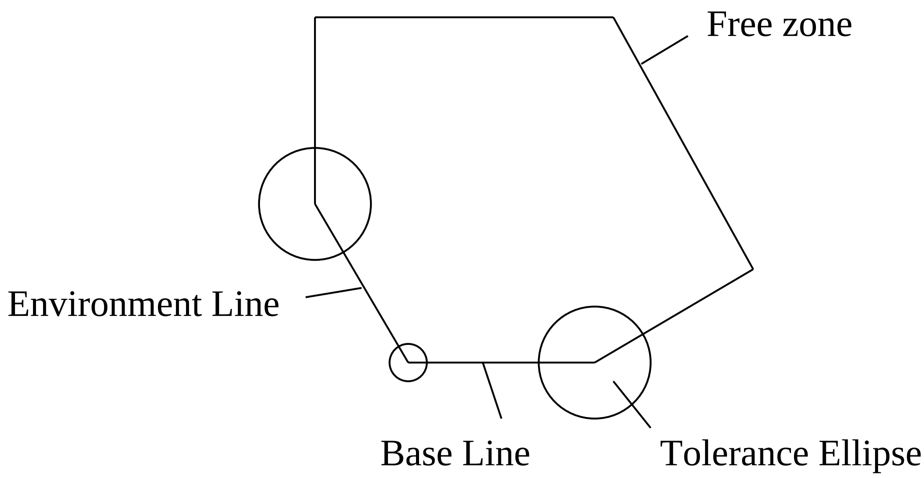

Figure 8: two triangles in 120 degree, requirements

For plane and volume meshing it is advantageous to transform this local environment to a local coordinate system, such that the base element is in reference position. This transformation is cheap in comparison to the simplification of the following steps. The generalization from plane to surface meshing is almost contained in this transformation. The local system spans the tangential plane to the surface.

In reference position every one of the rules is tested for applicability. If there are more applicable rules the one with the best produced elements will be chosen. The boundary data structure is updated according to the rule. The new generated points and interior elements are stored in the global point and element arrays.

If the test criteria for the rules are too sharp, a dead lock may appear. It may come to a situation, where no rule may be applied. If the test is too sloppy, a lot of bad shaped elements may be produced.

A possibility to make a compromise is the introduction of quality classes. Every boundary element is initialised to the highest quality class. The higher this value is the sharper the test done for a certain element will be. If there is no success, the quality class for this one element will be decreased. So it is possible to apply a worse fitting rule later on.

5.2 Rule description

The rules are stored in data structures. This makes the mesher more flexible in comparison to an algorithm where the rules are hardcoded in the program. To discuss the components of an rule we have a look at the specific rule, which inserts two triangles into an about 120 degree corner:

A rule needs one or more lines, which must be connected in a given way. One specific is the base line, which starts in the origin of the coordinate system and is parallel to the x axis. The length is scaled by the wanted edgelength and will be about unit length, if the boundary mesh is of the wanted edgelength. The second segment in our example must have the same second node as the base line as first node. In addition to the topological connection metric requirements have to been met. Every node has to be in a tolerance area, which can be an arbitrary orientated ellipse. The ellipse is small for high quality classes and increases for less quality.

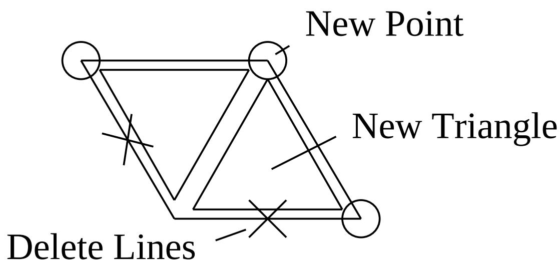

Figure 9: two triangles in 120 degree, actions

An other requirement is that no additional point or element covers a region, called free zone. This avoids overlapping when two fronts meet. The free zone is a polygon. It has to be modified if the actual points differ from the exact points. This can be described by means of a linear mapping, which assigns to a point deviation a correction to the free zone corners. In three dimensions the polygon is replaced by the convex hull of an point set, or, more general, the union of more convex hulls.

Figure 9 shows the action the rule above.

For some rules new points have to be inserted. To the point coordinates of the reference position a displacement is added. This displacement is, like for the free zone points above, a linear function of the point deviations of the existing points. To get the new front some boundary elements are inserted. Some of the old ones are deleted, some other having just the function of a catalyst. Front points are not deleted explicitly, they are removed when there is no adjacent boundary element. The interior elements are appended to the element list.

5.3 Examples

Our first example for three dimensional meshing is the union of a sphere and a cylinder. The input to the mesher is the following geometry description file:

```
solid cyl = cylinder ( 100, 0, 0; 0, 0, 0; 50 );
solid ball = sphere (0, 0, 0; 70.71);
solid all = (cyl or ball)
            and plain ( 100, 0, 0; 1, 0, 0)
            and plain ( -100, 0, 0; -1, 0, 0);
```

The result for $h = 25$ is given in figure 10. The first picture shows the calculated edges with the special points and tangents, the second the surface mesh.

The next example is also a union of a cylinder and a sphere, but with a 90° edge cut off and a drilled hole. The input is the following:

```
solid cyl = cylinder ( -100, 0, 0; 0, 0, 0; 50 ) and
            plain ( -100, 0, 0; -1, 0, 0 ) and
            plain ( 40, 0, 0; 1, 0, 0 ) ;
solid s1 = ( cyl or sphere (50, 0, 0; 70.71067812) );
solid s2 = plain (50, 0, 0; 1, 0, 0) or
```

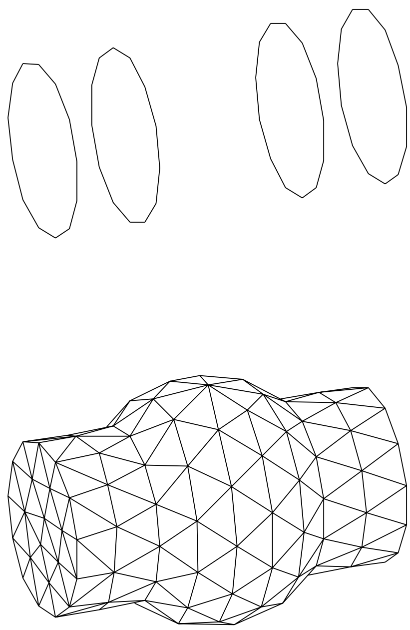

Figure 10: Union of cylinder and sphere

```
           plain (50, 0, 0; 0, 1, 0);
solid s3 = not cylinder ( 100, -50, 0; 50, 0, 100; 30);
solid all = s1 and s2 and s3;
```

The result for $h = 20$ is given in figure 11.

The third examples is the intersection of a cube with a sphere and the complement of a sphere. The result with $h = 8$ is given in figure 12.

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

6 Graded Meshes

To achieve a more precise result in FE computation with the same effort it is often useful to vary the mesh parameter h over the domain. In this chapter a method to determine the local parameter for two dimensional mesh generation is described.

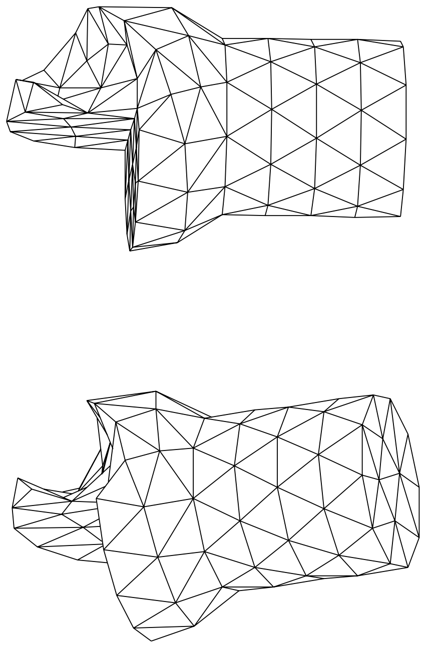

Figure 11: Abstract Tool

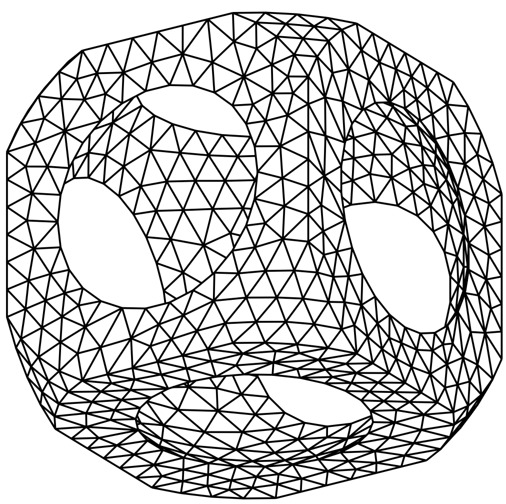

Figure 12: Cube and Spheres

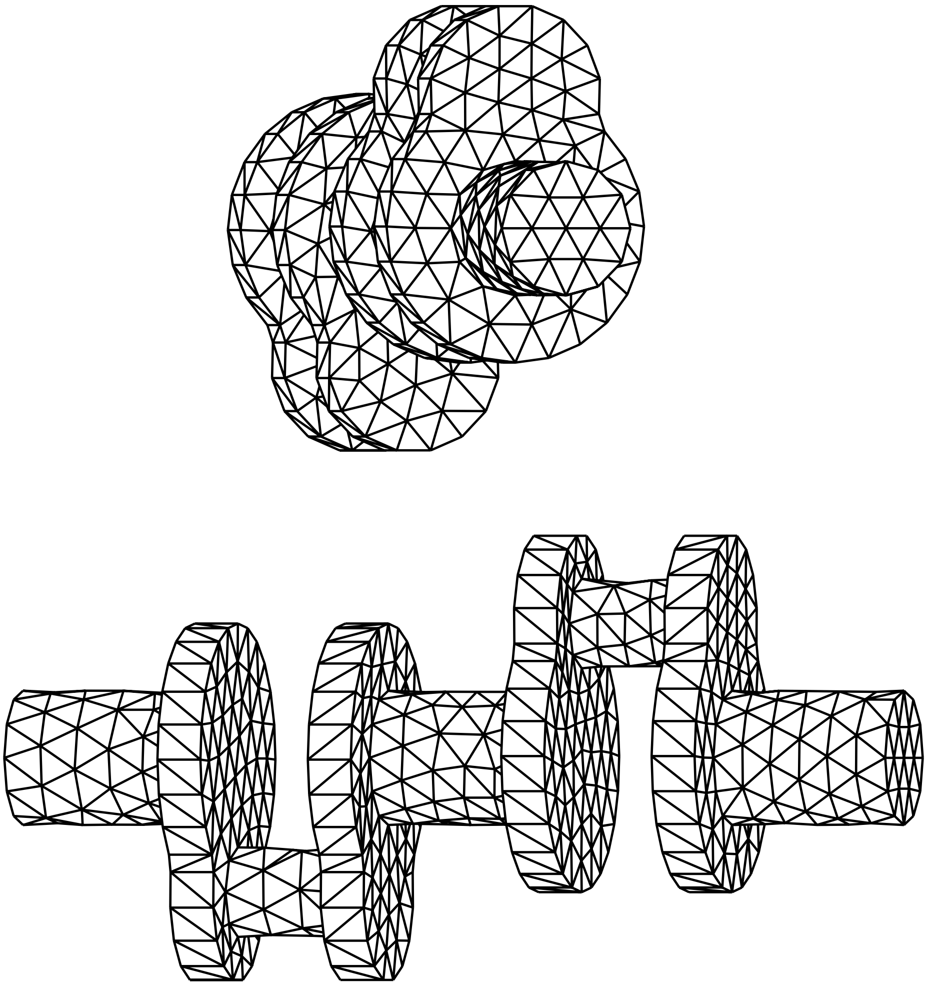

Figure 13: Crank Shaft

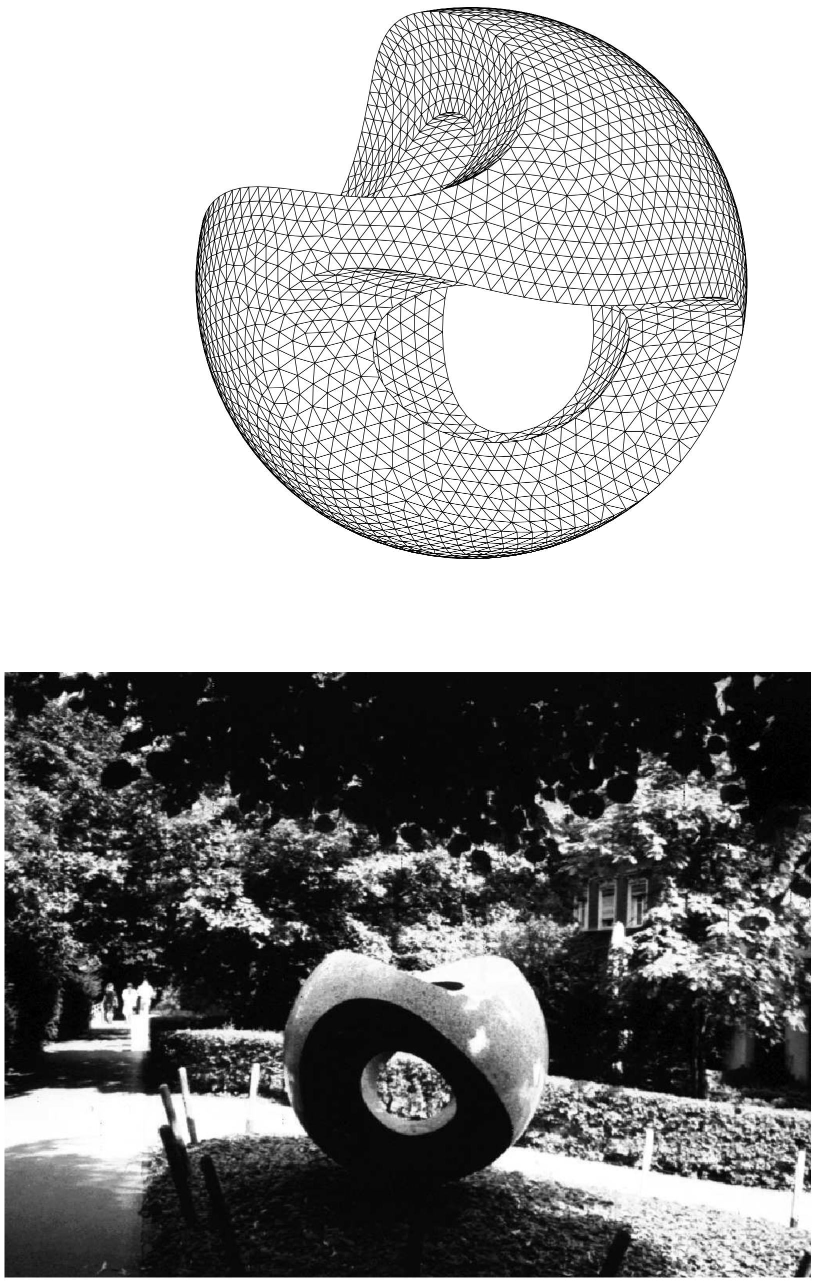

Figure 14: Sculpture in St. Gallen

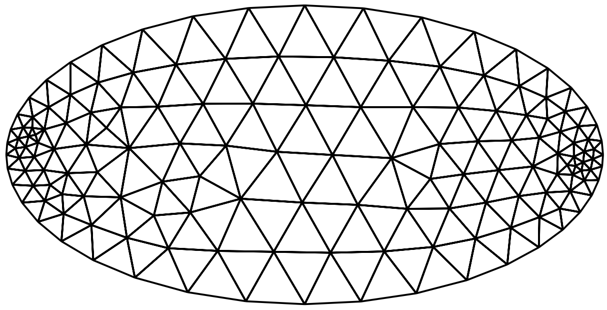

Figure 15: Graded Mesh, example 1

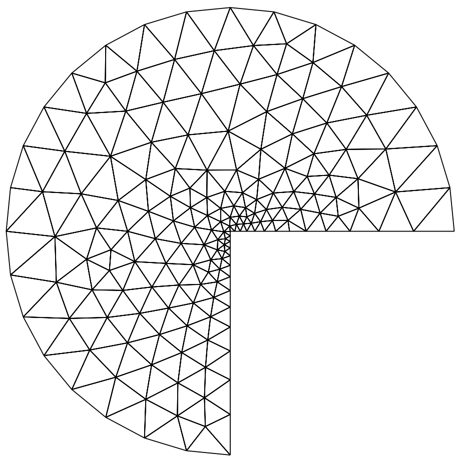

Figure 16: Graded Mesh, example 2

The required input to plane meshing is the set of boundary segments. We assume we have a boundary mesh with line segments of different length at hand. So we are given a function $h(x)$ on the boundary, namely the length of the according boundary element. The job is to extend this function on the boundary to the interior. In [3] a suggestion is made: One could calculate the Delaunay triangulation according to the boundary points and use linear interpolation of the boundary function. Our approach is computational more costly but delivers a smoother function: One could solve the Dirichlet problem for the Laplace equation using boundary element method. BEM is explained in a lot of text books, e. g. [7]. It is required to generate and solve a linear system with a dimension equal to the number of boundary segments. To evaluate the function in one interior point one has to form a sum over the calculated boundary values.

If the mesh parameter h varies slow across the domain it is enough to use a locally constant h to apply one rule. If it changes fast ($|\mathrm{grad} h| \geq 0.3$) one should do something better: One could use a local orthogonal curvilinear coordinate system such that the unit length in this system corresponds to h. Before the application of a rule the boundary image is transformed to cartesian coordinates and afterwards back to the curvilinear system. This gives a smooth change of the gridsize. Examples for graded meshes are given in figure 15 and figure 16.

7 Mesh Improvement

The quality of the generated mesh can be improved dramatically by two kind of mesh improvers. The standard smoothing procedure is Laplacian smoothing. This method moves every mesh node to the center of its neighbors in a cyclic way. This method is very similar to the Gauss-Seidel iteration in equation solving. The counterpart to the GS iteration is the Jacobi iteration. This method calculates the defect and after this it does a simultaneous correction step. The Jacobi iteration can also be applied for mesh smoothing. The advantage of this variant is, that one can work with the standard FE data structures and does not need the neighbors to each node. An other approach to smoothing is to minimize an error functional. For the graded examples the functional

$$\text{err} = \sum_{T_i} \frac{\text{circumference}(T_i)}{\text{area}(T_i)}$$

is used. For the three dimensional examples with constant h an additional term is added:

$$\text{err} = \sum_{T_i} \frac{\text{circumference}(T_i)}{\text{area}(T_i)} + \sum_{e_i} \frac{h}{l_i} + \frac{l_i}{h}$$

This function and also its gradient with respect to point coordinates can be calculated element by element. The advantage of the first term is, that the value goes to infinity as one triangle degenerates to a line.

An other method to mesh improvement is edge swapping. The optimal number of triangles according to one node is six. If there is a surplus or a shortage on elements the remaining elements cannot be well formed. Edge swapping detects such shortcomings and tries to fix them by topological changes to the mesh. Criterion when to swap edges are discussed in [2].

8 Further Goals

Now a basic meshing software is at hand, which can be extended into different directions:

- On the front size different geometric models will be implemented. This is important to couple the mesh generator to existing CAD software.
- People from fluid mechanics prefer quadrilaterals and hexahedrons instead of triangles and tetrahedrons. The rule application code will be extended to handle these elements, and rules for these elements will be written. If it is not possible to produce only hexahedrons, a mesh consisting of about 90 percent hexahedrons and the rest pentahedrons and tetrahedrons can be created. By subdividing every hexahedron into eight smaller ones, every pentahedron into six smaller hexahedrons and every tetrahedron into four smaller hexahedrons a pure hexahedron mesh can be generated.
- Edge length control methods will be applied to three dimensional solids. Also optimization algorithms for volume elements will be used.
- In structural mechanics it is important to couple one and two dimensional manifolds to volume domains. The mesh generator will be extended to manage these requirements.

References

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

[2] W. H. Frey and D. A. Field. Mesh relaxation: A new technique for improving triangulations. *Int. j. numer. methods eng.*, 31:1121–1133, 1991.

[3] P. L. George and E. Seveno. The advancing-front mesh generation method revisited. *Int. j. numer. methods eng.*, 37:3605–3619, 1994.

[4] Hoschek and Lasser. Grundlagen der geometrischen Datenverarbeitung.

[5] B. P. Johnston and J. M. Sullivan. A normal offsetting technique for automatic mesh generation in three dimensions. *Int. j. numer. methods eng.*, 36:1717–1734, 1993.

[6] D. Manocha. Solving systems of polynomial equations. *IEEE Comp. Graph. and Appl.*, March 1994.

[7] S. Rjasanov. Vorkonditionierte iterative Auflösung von Randelementgleichungen für die Dirichlet-Aufgabe. Technical report, TU - Chemnitz, 1990.

[8] W. Zulehner. A simple homotopy method for determining all isolated solutions to polynomial systems. *Math. Comp.*, 50(181):167–177, January 1988.

[9] W. Zulehner. Skriptum zur Vorlesung Numerik I (Gleichungen). Institut für Mathematik, JKU Linz, 1991.