# Automatic Mesh Generation for 3D Objects

Ferdinand Kickinger
Institut für Mathematik
Johannes Kepler Universität Linz

**Abstract**

In this paper, an automatic mesh generator for three dimensional objects is presented. This mesh generator produces uniform meshes locally fitted to the boudary and to interfaces. Sometimes, such meshes are also called locally irregular. The computational complexity of the mesh generator is of the order $O(n)$, where $n$ denotes the number of mesh points. Therefore, the algorithm is suited for generating rather fine grids, quite efficiently. Aspects of a-priori and a-posteriori adaptiv mesh refinements are also discussed.

**Keywords:** Automatic 3D mesh generator, locally irregular meshes, a-priori and a-posteriori adaptive mesh refinement.

**AMS(MOS) subject classification:** 65N50, 65N55.

## 1 Introduction

This paper is devoted to the description of an *automatic mesh generator* in 2D and, above all, 3D. If we look arround for mesh-generators, we find a lot of them for two dimensions, which are running very satisfyingly. There are algorithms for parallel generations of meshes and we can produce graded grids. In three dimensions the situation in totaly different. Out of the recent hardware development, the engeneering of three dimensional mesh generators has started in the last few years. In three dimensions, we subdivide into **semi-automatic** mesh stategies, and **(full-) automatic** strategies. The semi-automatic mesh generators typically require some topological information about the geometry in the background, to produce a mesh correctly and sucessfully. By using a (isoparametric, transfinite) mapping from a simple geometry with a known mesh, to the origin geometry, part by part, the wanted mesh appears on the geometry. The disadvantage of these methods is obviously, that we have to do a lot of handwork, making these mappings, and putting the parts together. In (full-)automatic stategies, the situation is more convienient. We put in the geometry, given in different data models and get, after quite a while the more or less convienient mesh out. A problem appearing at the point of putting a geometry into a mesh generator, is the coupling with CAD- CAM-systems, which we need for doing calculations on complicated real world problems. One strategy for doing such an automatical mesh generation in three dimensions is the moving front technique. Here we start at a given surface mesh of the boundary and the interfaces of the geometry. Cut layer by layer tetrahedrons (hexaedrons) off, untill there is nothing left of the geometry. The disadvantage of this algorithms is, that we have to search in the actual front for neighbourships, and collabing of the front. This search in the front raises the complexity to the theoretical complexity of $O(n \log n)$, where $n$ denotes the number of meshpoints. This result depends on the used search algorithm. In practical problems, we get a real complexity of $O(n^p \log n)$ with $p$ is a real number shortly bigger than one ($p = 1.15$ see [22]). Our approach is based on producing meshes, which are locally fitted to the boundary. This allows the optimal complexity of the algorithm. Part by part we build up meshes for objects which become more and more complicated. Complicated in this context means the stucture of the boundary. First we give an algorithm, that creates meshes for objects with sufficient smooth boundary ($\in \mathbf{C}^1$). We then generalize the algorithm for acting on boundarys with, so called, **critical points**. This are points, where the surface is continious, but not differentiable. Next, we catch **critical lines**, which are one dimesional subsets of the boundary, where the surface is continious, but not differentiable. We end up with a method, which is useable for objects, coming from real world problems.

We now start with the construction of an optimal mesh generator, for the generation of tetrahedron / oktahedron meshes in three dimensions. As special case we obtain a generator for two dimensions, producing meshes using triangles.

## 2 Definitions and Preliminary Results

The main idea for constructing the generator is cutting the $\mathbf{R}^3$ into tetrahedrons and oktahedrons of edge length $h$. This results in a node structure, that can be written as follows:

**Definition 1**

$$\textit{Let } \mathbf{D}(h) \textit{ be } \mathbf{Z}\times\mathbf{Z}\times\mathbf{Z}. \tag{1}$$

$$\textit{For an arbitary point } p=(i,j,k)\in D$$
$$\textit{there is a corresponding point } q\in\mathbf{R}^3 \textit{ as following :}$$
$$(i,j,k)\hat{=}h\cdot(i+\tfrac{1}{2}j+\tfrac{1}{2}k,\tfrac{\sqrt{3}}{2}j+\tfrac{1}{2\cdot\sqrt{3}}k,\tfrac{\sqrt{2}}{\sqrt{3}}k). \tag{2}$$

It is obvious, that the vertices above build up some tetrahedron/oktahedron structure.

**Definition 2**

$$\textit{The four Points } p_1,p_2,p_3,p_4\in\mathbf{D}(h) \textit{ are forming a } \textbf{Tetrahedron}$$
$$\Updownarrow$$
$$\textit{the corresponding } q_1,q_2,q_3,q_4\in\mathbf{R}^3 \textit{ fulfil}$$
$$\|q_i-q_j\|=h,\quad i\neq j,\quad i,j=1,2,3,4. \tag{3}$$

$$\textit{The six Points } p_1,p_2,p_3,p_4,p_5,p_6\in\mathbf{D}(h) \textit{ are forming an } \textbf{Oktahedron}$$
$$\Updownarrow$$
$$\textit{the corresponding } q_1,q_2,q_3,q_4,q_5,q_6\in\mathbf{R}^3 \textit{ fulfil}$$
$$h\leq\|q_i-q_j\|\leq h\cdot\sqrt{2},\quad i\neq j,\quad i,j=1,2,3,4,5,6. \tag{4}$$

$$\textit{The five Points } p_1,p_2,p_3,p_4,p_5\in\mathbf{D}(h) \textit{ are forming a } \textbf{Pyramid}$$
$$\Updownarrow$$
$$\textit{the corresponding } q_1,q_2,q_3,q_4,q_5\in\mathbf{R}^3 \textit{ fulfil}$$
$$h\leq\|q_i-q_j\|\leq h\cdot\sqrt{2},\quad i\neq j,\quad i,j=1,2,3,4,5. \tag{5}$$

Let in the following **3D-elements** be Tetrahedrons, Oktahedrons or Pyramids.

We now intruduce a metric on D:

**Definition 3**

$$\textit{The distance } d_D(p_1,p_2)=0\Leftrightarrow p_1=(i_1,j_1,k_1)=(i_2,j_2,k_2)=p_2$$
$$\textit{The distance } d_D(p_1,p_2)=1 \textit{ (near neigbours)}\Leftrightarrow$$
$$1\leq\mid i_1-i_2\mid+\mid j_1-j_2\mid+\mid k_1-k_2\mid\leq 2\quad\wedge$$
$$0\leq\mid(i_1-i_2)+(j_1-j_2)+(k_1-k_2)\mid\leq 1.$$
$$\textit{The distance } d_D(p_1,p_2)=2 \textit{ (far neigbours)}\Leftrightarrow$$
$$\mid i_1-i_2\mid+\mid j_1-j_2\mid+\mid k_1-k_2\mid=3\quad\wedge$$
$$0\leq\mid(i_1-i_2)+(j_1-j_2)+(k_1-k_2)\mid\leq 1.$$
$$\textit{the distance is } 3 \textit{ else.} \tag{6}$$

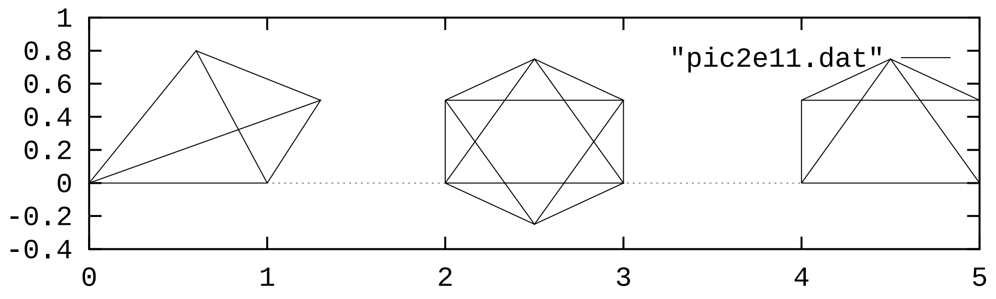

Figure 1: Tetrahedron, Oktahedron, Pyramid

It is easy to show that this is a metric. With this definition the following theorem gives us an abstract condition if tetrahedron or not.

**Theorem 1**

1) *Two points* $p_1, p_2 \in \mathbf{D}(h)$ *have distnace* $d_D(p_1, p_2) = 0 \Leftrightarrow$
*the corresponding points* $q_1, q_2 \in \mathbf{R}^3$ *have distance* $d_{R^3}(q_1, q_2) = 0.$

2) *Two points* $p_1, p_2 \in \mathbf{D}(h)$ *have distnace* $d_D(p_1, p_2) = 1 \Leftrightarrow$
*the corresponding points* $q_1, q_2 \in \mathbf{R}^3$ *have distance* $d_{R^3}(q_1, q_2) = h.$

3) *Two points* $p_1, p_2 \in \mathbf{D}(h)$ *have distnace* $d_D(p_1, p_2) = 2 \Leftrightarrow$
*the corresponding points* $q_1, q_2 \in \mathbf{R}^3$ *have distance* $d_{R^3}(q_1, q_2) = h\sqrt{2}.$

4) *Two points* $p_1, p_2 \in \mathbf{D}(h)$ *have distnace* $d_D(p_1, p_2) = 3 \Leftrightarrow$
*the corresponding points* $q_1, q_2 \in \mathbf{R}^3$ *have distance* $d_{R^3}(q_1, q_2) > h\sqrt{2}.$

5) *The four points* $p_1, p_2, p_3, p_4 \in \mathbf{D}(h)$ *are forming a* **Tetrahedron**

$$\Updownarrow$$

$$d_D(p_i, p_j) = 1, \quad i \neq j, \quad i, j = 1, 2, 3, 4. \tag{7}$$

6) *The six points* $p_1, p_2, p_3, p_4, p_5, p_6 \in \mathbf{D}(h)$ *are forming a* **Oktahedron**

$$\Updownarrow$$

$$1 \leq d_D(p_i, p_j) \leq 2, \quad i \neq j, \quad i, j = 1, 2, 3, 4, 5, 6. \tag{8}$$

7) *The five points* $p_1, p_2, p_3, p_4, p_5 \in \mathbf{D}(h)$ *are forming a* **Pyramid**

$$\Updownarrow$$

$$1 \leq d_D(p_i, p_j) \leq 2, \quad i \neq j, \quad i, j = 1, 2, 3, 4, 5. \tag{9}$$

Proof: trivial.

Now we are able to start with generating surface meshes of sufficient smooth ($\mathbf{C}^1$) surfaces.

## 3 Surface Mesh Generation for Sufficiently Smooth Surfaces

Let in the following $\Phi$ be a surface in space, given by explicit or implicit representation. We do not care about details, we just request some smoothness ($\mathbf{C}^1$), and a projector $p$, that puts a point $x$ "close" to $\Phi$ onto $\Phi$ such that this point $\bar{x}$ has minimal distance to $\Phi$.

We now define a surface in the discret case (with elements in $\mathbf{D}(h)$). The aim is, to find such a surface in $\mathbf{D}(h)$ "close" to $\Phi$.

Similar to tetrahedron and oktahedrons, we can define 2D-elements as following:

**Definition 4**

$$\begin{gathered} \textit{The three points } p_1, p_2, p_3 \in \mathbf{D}(h) \textit{ are forming a } \textbf{Triangle} \\ \Updownarrow \\ \textit{the corresponding } q_1, q_2, q_3 \in \mathbf{R}^3 \textit{ fulfil} \\ \|q_i - q_j\| = h, \quad i \neq j, \quad i,j = 1,2,3. \end{gathered} \tag{10}$$

**Definition 5**

$$\begin{gathered} \textit{The four points } p_1, p_2, p_3, p_4 \in \mathbf{D}(h) \textit{ are forming a } \textbf{Quadrilateral} \\ \Updownarrow \\ \textit{the corresponding } q_1, q_2, q_3, q_4 \in \mathbf{R}^3 \textit{ fulfil} \\ h \leq \|q_i - q_j\| \leq h\sqrt(2), \quad i \neq j, \quad i,j = 1,2,3,4. \quad \wedge \\ \textit{each corresponding point } q_i \textit{ has distance } h \textit{ to } q_j, q_k, \wedge \\ q_i \textit{ has distance } h\sqrt(2) \textit{ to } q_l, \quad i,j,k,l = 1,2,3,4, \quad i \neq j \neq k \neq l \end{gathered} \tag{11}$$

**Definition 6**

$$\begin{gathered} \textit{The neigbourhood of order 1 of a point} \quad (N_1(p)) \quad p \in \mathbf{D}(h) \\ \textit{are all elements } q \in \mathbf{D}(h) \textit{ with distance } d_D(p,q) = 1. \end{gathered} \tag{12}$$

**Definition 7**

$$\begin{gathered} \textit{The neigbourhood of order 2 of a point} \quad (N_2(p)) \quad p \in \mathbf{D}(h) \\ \textit{are all elements } q \in \mathbf{D}(h) \textit{ with distance } 1 \leq d_D(p,q) \leq 2. \end{gathered} \tag{13}$$

**Definition 8**

$$\begin{gathered} \textit{A point } p \in \mathbf{D}(h) \textit{ has closed neigbourhood in} \\ S = (q_1, q_2, \ldots, q_n), \quad q_i \in \mathbf{D}(h) \Leftrightarrow \\ \textit{all points } \bar{q}_i \in S \wedge \in N_2(p) \textit{ have two points} \\ \dot{q} \neq \ddot{q} \neq \bar{q}_i \in S \wedge \in N_2(p) \wedge \in N_1(\bar{q}_i) \end{gathered} \tag{14}$$

**Definition 9**

$$\begin{gathered} \textit{A list of points } S = (p_1, p_2, \ldots, p_n), \quad p_i \in \mathbf{D}(h) \\ \textit{is called surface in } \mathbf{D}(h) \Leftrightarrow \\ 1) \qquad \textit{there are no 3D-elements in } S \\ 2) \qquad \textit{each point } p_i \in S \textit{ has closed neigbourhood.} \end{gathered} \tag{15}$$

**Theorem 2**

$$\begin{gathered} \textit{The following algorithm calculates out the surface in } \mathbf{D}(h) \\ \textit{beeing closest to } \Phi \end{gathered} \tag{16}$$

With this definitions, we can compute and implement the following algorithm for calculating surface meshes of closed $\mathbf{C}^2$ surfaces.

"Closest" in this context means a distance between two surfaces in space, where the first is $\Phi$ and the second the induced surface by $S$, with linear boundary at triangles, and bilinear boundary at quadrilaterals.

**Algorithm 1** *Surface Meshing*

```
1. START
    (a) Take an arbitary point $p$ on the surface $\Phi$.
    (b) Search for a point $\bar{p} \in \mathbf{D}$ with corresponding point $\bar{q}$ beeing closest to $p$.
    (c) Search in the $N_2(\bar{p})$ for a point $\dot{p} \in \mathbf{D}$ beeing closest to $\Phi$.
    (d) Take $\dot{p}$ as starting value for the recursiv part.
2. RECURSIVE PART
    (a) The starting value we call $p$
    (b) We take the corresponding $q$ to $p$ and project it on $\Phi$, and add this new point
        in the meshpointlist.
    (c) While (p has no closed neigbourhood)
        • Search in the $N_2(p)$ for the point $\hat{p}$, which has minimal distance to $\Phi$,
          and no 3D-elements are build with prior points.
        • Call recursive part with starting value $\hat{p}$.
```

With this algorithm, we can calculate, for example, the surface-mesh of a sphere and a torus, shown in Figure 6 and 7.

The next point is, that most of the problems which have to be meshed, have not a surface $\Phi \in \mathbf{C}^1$. Real world objects are mostly piecewise $\mathbf{C}^1$. The lines and points of being $\neg\mathbf{C}^1$ are called **Critical Lines** and **Critical Points**.

## 4 Surface Meshes in Critical Points

The typical problem in this case is building up a surface mesh of a cone, where the problems occurs in its peak. To give a solution for this problems, we have to explain, what **collabing of points** mean.

**Definition 10**

$$\begin{aligned} &\textit{Two Points } p_1, p_2 \in \mathbf{D}(h), \quad d_D(p_1,p_2) = 1 \textit{ are } \textbf{collabed} \\ &\textit{if, per definition, the corresponding points } q_1, q_2 \in \mathbf{R}^3 \textit{are equeal} \end{aligned} \tag{17}$$

This means for example, that from now on, the point $p_1 \in \mathbf{D}(h)$ has the corresponding point as $p_1$. What we get out, is a change in the tetrahedon/oktahedron structure. Picture 2 shows what happens in the 2D case.

The Neigbourhoods of the two points $p_1, p_2$ must be changed too. The new $N_1(p_1) = N_2(p_2)$ is the union of neigbourhoods of $p_1, p_2$, the same for $N_2$. With this modification, all the other definitions hold, especialy the definition of 2D- and 3D-elements and the closed neigbourhood. Now its clear what we do in a critical points $p$:

**Algorithm 2** *Critical Point Meshing*

```
• Search for a point $\bar{p} \in \mathbf{D}$ with corresponding point $\bar{q}$ beeing closest to $p$.
• Project this point onto $p$.
• Collab enough points arround $\bar{p}$.
```

Enough points mean, as much as we need to catch the geometry.

Now we are able to build up a mesh of an infinite cone shown in Figure 8.

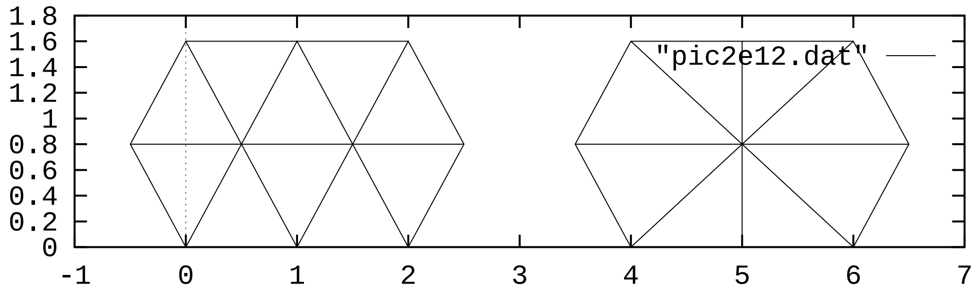

Figure 2: Collabing of points

## 5 Surface Mesh Along a Critical Line

This part of generating a mesh for three dimensional objects is the most difficult. In principal the approach is similar to critical points. We go along the critical line, and collab points $\in \mathbf{D}(h)$ "near" to the line into groups.

In the following, let the critical line be given in implizit or explizit representation for a curve in space ($\phi \in \mathbf{C}^1$). The following algorithm will explain roughly how we deal with this problem.

**Algorithm 3** *Critical Line Meshing*

1. **START**
   - Search for a point $\bar{p} \in \mathbf{D}$ with corresponding point $\bar{q}$ beeing closest to $\phi$.
   - Search in the $N_1(\bar{p})$ for a point $\dot{p} \in \mathbf{D}$ beeing closest to $\Phi$.
   - Take $\dot{p}$ as starting value for the recursiv part.
2. **RECURSIVE PART**
   (a) The starting value we call $p$
   (b) if $(p - p_\phi \leq h \cdot param)$ and we have not visited this point yet
      - if ($p$ is near a group of collabed points collab it too)
      - else (build up a new group)
      - Call recursive part with starting value all $\hat{p} \in N_1(p)$.

$p_\phi$ means the projection of $p$ onto $\phi$ and $param$ stands for a parameter of about one, that depends on the angle between to (ore more) surfaces ($\in \mathbf{C}^1$) ending up in the critical line. (If the angel is small the parameter is small, if the angel is big, the parameter is big (max.: 2).)

Combining all of the above algorithms, we can generate surface meshes of the following objects (Figure 9,10,11).

## 6 Building up the Interiour Mesh

Now we can create a surface mesh for an given object consisting of triangles and quadrilaterals. This mesh can be used as a border for an also recursiv algorithm, that puts all interiour nodes into the mesh vertex list. If we have more than one material, we apply this algorithm for each material.

**Algorithm 4** *Interiour Mesh*

```
1. START
   Search for a point $\bar{p} \in \mathbf{D}$ with corresponding point $\bar{q}$ in the interiour of the material.
2. RECURSIVE PART
   (a) The starting value we call $p$
   (b) if (we have not visited this point yet) (also in the surface mesh generation)
       • Call recursive part with starting value: all $\hat{p} \in N_2(p)$.
```

## 7 Computational Time

We can write down the following steps for generating a mesh of a 3D-object.

- Put all critical points into the mesh
- Put all critical lines into the mesh
- Generate the surface mesh of the $\mathbf{C}^1$ pieces
- Building up the 2D-elements
- Smoothing the 2D-elements
- Building up the interior mesh
- Building up the 3D-elements
- Smoothing the 3D-elements

The following formular gives approximately the complexity of the whole algorithm.

$$T = c_1 \cdot L \cdot h^{-1} + c_2 \cdot S \cdot h^{-2} + c_3 \cdot V \cdot h^{-3}. \tag{18}$$

| | |
|---|---|
| $c_1, c_2, c_3$ | are positiv constants, bounded and independent of $h$, |
| $L$ | means the length of the critical lines, |
| $S$ | is the quantitativ surface of the object, |
| $V$ | is the volume of the object. |

Note, that we take a three dimensional array of integers as analogon for $\mathbf{D}(h)$. In this array we put in, if visited, the number of the gridnode in mesh-point list. This results in the optimal complexity of order $O(n)$. We do not need to check any surrounding like in moving-front algorithms.

One of the disadvantages is, that we have to produce also fine grids at areas where we do not need them. The solution therefore is a-priori and a-posteriori mesh adaption.

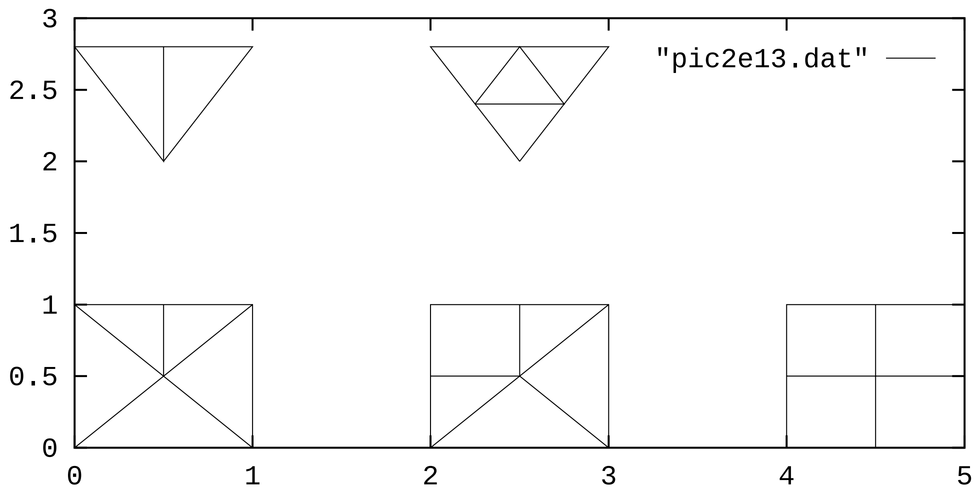

Figure 3: Refinemet of 2D elements

## 8 Mesh Adaption

First we have to look on our elements (2D or 3D) if we refine them. Suppose we have a discrete function given in each meshpoint, that gives us the order of refining (order of refining $n$ means, that the discretisation parameter must be $h \cdot 2^{1-n}$). What we do with triangels and quadrilaterals we see in Figure 3.

For the case of maximal refinement parameter 2, Figure 3 shows us the different cases in the 2D-case. If we concentrate on the rule, that an edge is only refine if both ends have refinement parameter 2, nothing can go wrong (dealing with quadrilaterals we note, that if at least one edge is refined, we add the centerpoint). In the 3D-case it is similar (see Figure 4).

For the case of maximal refinement parameter 2, Figure 4 shows us the different cases in 3 dimensions. The same rule as above holds. (dealing with oktahedrons and pyramids we note, that if at least one edge is refined, we add the centerpoint). In the 3D-case it is similar (see Figure 4). Let us call the case where every edge is halved **total refinement**.

Now we are able to increase the maximum discretisation parameter. By a recursive use of the above cases we can rewrite this in form of an algorithm.

Note, that in the case of maximum discretisation parameter $> 2$, all values $> 2$ have to be surrounded by a value $\geq 2$. If a given refinement function does not fulfil this condition, we have to do one step of "smoothing" by replacing all 1s in the surrounding of values $> 2$ by 2s.

**Algorithm 5** *Adaptiv Refinement*

- Parameters: refinement parameter for each node
- Whatever for an element we have to refine
- if (the refinement information is included in one of the above cases) apply the case
- else make a total refinement of the element, and make a recursive call of this procedure with every new element resulting of the refining. As refinement parameter for the nodes take

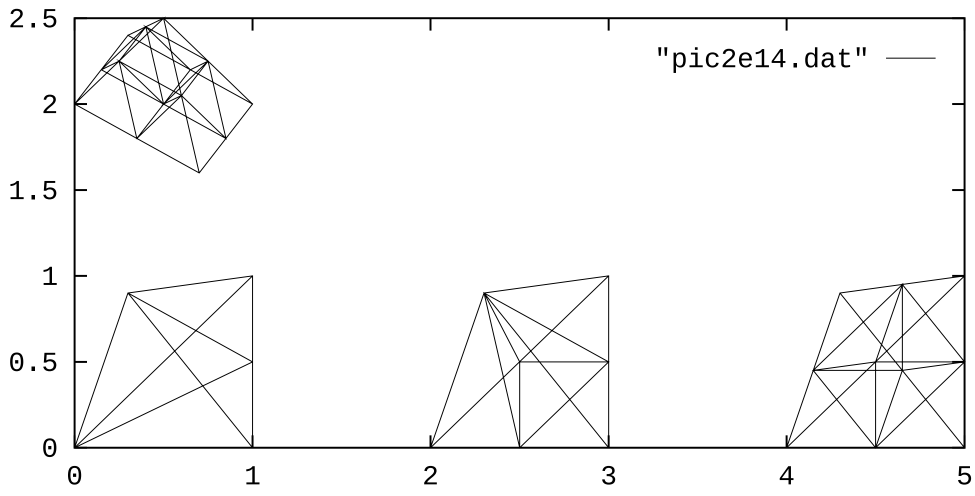

Figure 4: Refinemet of tetrahedrons

- the refinement parameter -1 if it is also a node of the non refined element
- else if the refinement parameters of the nodes in the non refined grid, out of this new node was created by halvening, are equeal, it is refinement parameter -1 too.
- else (the refinementparameters are different) take the minimum of the "old" parameters.
- when refining quadrilaterals, oktahedrons and pyramids, the center point gets as refinement parameter the minimum of the "old" parameters.

In Figure 12 we see two examples of adaptiv refinements in the 2D-case.

Now we are able to discuss a-prori and a-posteriori refinement strategies for our mesh generator. In the a-priori case, we add to each node in our three dimensional array, that is modelling $\mathbf{D}(\mathbf{h})$ some refinement information. That means, we substitute this array of Integers, by an array of arrays of Integers, which are dynamicly allocated. This means, where the geometry of our object requires a smaller discretisation parameter than $h$, we refine as much as we need. Note that this nefining does not change anything in using the above algorithms. The above definitions hold also for this adapted mesh structure. A-posteriori refining is much easier if we do not care about how we get the refining information. Guess we have this knowledge, then we have "only" to go through the element list, and refine element by element.

## 9 The Code NAOMI, FEM3DSYM and Numerical Tests

The mesh generating algorithms described in the proceding sections were implemented in *C++*. Further we have developed a finite element code together with an algebraic multigrid solver (see [27] and [28] for deatails). The resulting code is called FEM3DSYM, standing for Finite Element Package for Symmetric Problems, and NAOMI, standing for Numerical Automatical Optimal Meshing Instrument. This Package is able to generate meshes for a combination of shperes, zylinders, cones, toris and planes. After meshing, we build up the finite element equations using tetrahedrons, pyramids, oktaherons, triangles and quadrilaterals (see [28]). The algebraic multigrid solver used in the package is presented in [28]. A detailed description of the programs used can be found in [27].

In the following, we generate meshes for different 3D-objects with increasing complexity in the boundary structure. Using these meshes, we then solve the following Dirichlet problem for the Poisson equation:

$$- \triangle u = 1 \text{ in } \Omega \text{ and } u = 273.15 \text{ on } \Gamma = \partial\Omega. \tag{19}$$

All numerical experiments were tested out on a Pentium with 100 Mhz and 32 MByte RAM. In the following tables, the time values are given in seconds.

**Example 1** *Sphere*

The first test is a sphere with radius $r = 0.2$. The objekt and the mesh ($h = 0.04$) is given in Figure 6.

| Sphere | h=.04 | h=.02 | h=.01 |
|---|---|---|---|
| Meshpoints | 948 | 6748 | 50683 |
| 3D elements | 2225 | 17911 | 142999 |
| 2D elements | 693 | 2772 | 11086 |
| Time mesh | 1 | 4 | 33 |
| Time matrix | 1 | 6 | 60 |
| Time solver | 1 | 6 | 120 |

**Example 2** *Torus*

The next test example is a torus with outer radius $r_1 = 0.175$ and inner radius $r_2 = 0.75$. The torus and the mesh (h=0.02) are presented in 7.

| Torus | h=.04 | h=.02 | h=.01 |
|---|---|---|---|
| Meshpoints | 637 | 4301 | 30771 |
| 3D elements | 1266 | 10495 | 82842 |
| 2D elements | 700 | 2831 | 11604 |
| Time mesh | 1 | 4 | 28 |
| Time matrix | 1 | 3 | 30 |
| Time solver | 1 | 4 | 26 |

**Example 3** *Cone*

The third test example is a cone with radius $r = 0.15$ and height $h = 0.3$. The cone and the mesh (h=0.02) are given in Figure 8.

| Cone | h=.04 | h=.02 | h=.01 |
|---|---|---|---|
| Meshpoints | 228 | 1615 | 11079 |
| 3D elements | 495 | 4002 | 29736 |
| 2D elements | 303 | 1193 | 4672 |
| Time mesh | 1 | 3 | 19 |
| Time matrix | 1 | 2 | 10 |
| Time solver | 1 | 1 | 10 |

**Example 4** *Hexaedron with Torus*

The next test example has a surface with edges. The geometry and the mesh (h=0.014) is drawn in Figure 9.

| Testobj. 1 | h=.02 | h=.014 | h=.012 |
|---|---|---|---|
| Meshpoints | 8368 | 23853 | 39551 |
| 3D elements | 22426 | 65277 | 109604 |
| 2D elements | 4514 | 9663 | 39551 |
| Time mesh | 7 | 18 | 29 |
| Time matrix | 7 | 23 | 39 |
| Time solver | 8 | 24 | 38 |

**Example 5** *Zylinder with Torus*

The last test example consists of a cylinder intersected by a torus. The geometry an the mesh (h=0.01) is presented in Figure 10.

| Testobj. 2 | h=.02 | h=.01 |
|---|---|---|
| Meshpoints | 3586 | 28360 |
| 3D elements | 9488 | 78758 |
| 2D elements | 2192 | 9309 |
| Time mesh | 4 | 27 |
| Time matrix | 3 | 28 |
| Time solver | 3 | 28 |

Finally we applied our mesh generator to a more sophisticated object called *Motor Block*. The geometry and the mesh are given in Figure 11. At some places, the mesh seems to be inconsistent (see also Figure 10). However, this comes out of projecting tree nodes of a quadrilateral onto an edge.

The complexity estimate given in the previous section for the mesh generator is confirmed by numerical results presented in the tables above. At least for fine grids, we observe that

$$\frac{N_h}{N_H} \approx \frac{tmesh_h}{tmesh_H}, \quad h, H \text{ are two different mesh parameter.} \tag{20}$$

The same is true for the matrix generation and the algebraic multigrid solver. Terefore, all components of the package show an optimal behaviour with respect to the CPU-time in practice too.

## 10 Further Goals in Automatic Mesh Generation

Usage of Splines in Geometrical Modelling

The automatic mesh generator for tree dimensions, implemented in the C++ program **NAOMI** produces quite satiesfying meshes. The restriction on the possible geometries like sphere, cone, cylinder, plane, ... is not very convienient. This leads to a coupling with CAD, CAM systems. As we see, we require only a projector, that puts a point near to a given surface or curve onto this surface or curve. Also the use of splines in the geometrical modelling is a thing of interest.

Implementation of a-priori and a-posteriori Mesh Adaption

Further, if we want to calculate electrical machines, we have to implement a-priori and a-posteriori mesh adaption. As a special case of the 3D mesh generator, a strategie for two dimensions appear. In this case, we have already implemented the two strategies. For results see [28] or the appendix.

Parallelization of the Mesh Generator

Doing serious calculations in three dimensions, we have to think about parallelization, to get satisfying results and to deal with computer ressources, without them, we can not calculate any complex three dimensional geometry.

## References

[1] R. Dautray J.-L. Lions: **Mathematical Analysis and numerical Methods for Science and Technology** Vol 1. Springer-Verlag Berlin, 1990.

[2] V. Girault, P.-A. Raviart: **Finite Element Approximation of the Navier-Stokes Equations** Springer-Verlag Berlin, 1979.

[3] T. Rossi: **Fictious Domain Methos**. University of Jyväskylä, 1995.

[4] A. A. Reusken: **A Multigrid Method Based on Incomplete Gaussian ELimination**. Eindhoven University of Technology, Department for Mathematics and Computer Science, RANA 95-13, 1995.

[5] J. H. Bramble, J. E. Pasciak, J. Xu: **Parallel Multilevel Preconditioners**. Mathematic of Computation, 55(191):1-22, 1990.

[6] F. Brezzi, M. Fortin: **Mixed and Hybrid Finite Elements**. Springer Verlag, 1991.

[7] J. W. Ruge, K.Stüben **Algebraic Multigrid (AMG)**. Multigrid Methods (St. Mc Cormick, ed.), Frontiers in Applied Mathematics, Vol 5, SIAM, Philadelphia 1986.

[8] P.M. de Zeeuw: **Matrix Dependent Prolongations and Restrictions in a Black Box Multigrid**. J. Comp. and Appl. Mathematics 33, 1-27 1990.

[9] F. Chatelin and W.L. Miranker: **Acceleration by Aggregation of Successive Approximation Methods**. LAA 43, 17-47 1982.

[10] W. Hackbusch **Iterative Löser großer schwachbesetzter Gleichungssysteme** Teubner Studienbücher Mathematik, 1993.

[11] Ch. Großmann H.-G. Roos: **Numerik partieller Differentialgleichungen** Teubner Studienbücher Mathematik, 1994.

[12] S. Margenov, J. Maubach: **Optimal Algebraic Multilevel Preconditioning for Local Refinement along a Line**. Numerical Linear Algebra with Application 2 (4), 347-361, 1995.

[13] B. Heise: **Parallel solvers for linear and nonlinear exterior magnetic field problems based upon FE/BE formulations**. Institutsbericht Nr. 486, Universität Linz, Institut für Mathematik, 1995.

[14] B.Heise: **Comparison of Parallel Solvers for Nonlinear Ellipic Problems Based on Domain Decomposition Ideas**. Institutsbericht Nr. 494, Universität Linz, Institut für Mathematik, 1995.

[15] B. Heise: *A Mixed Variational Formulation for 3D Magnetostatics and Its Finite Element Discretisation*. Tecnical Report 96-3, Universität Linz, Institut für Mathematik, Arbeitsgruppe Numerische Mathematik und Optimierung, 1996.

[16] J. Xu **The Auxiliary Space Method and optimal Multigrid Preconditioning Techniques for Unstructured Grids**. Computing, 1996 (to appear).

[17] P. Vanek, J. Krizkova: **Two-Level Method on Unstructured Meshes With Convergence Rate Independent of the Coarse-Space Size**. Report No. 35 University of Coorado at Denver, Center for Computational Mathematics, 1995.

[18] P. Vanek, J. Mandel , M. Brezina: **Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems** . Computing 56, 179-196, 1996.

[19] P. Vanek, J. Krizkova: **Algebraic Multigrid on Unstructured Meshes**. Report No. 34 University of Coorado at Denver, Center for Computational Mathematics, 1994.

[20] T. Greßner, A. Schneider **A 2-D Grid Editing Package**. Report no. 15, Universität Bonn, Institut für Mathematik, 1995.

[21] F. Kickinger: **Algebraic Multigrid for Elliptic Problems of Second Order**. Tecnical Report 96-2, Universität Linz, Institut für Mathematik, Arbeitsgruppe Numerischa Mathematik und Optimierung, 1996.

[22] H. Jin and R. I. Tanner: **Generation of unstructured tetrahedral meheshes by advancing front tequnique**. International Journal of Numerical Methods in Engeneering , Vol 38 , 1995.

[23] C. Yerker, I. Zeid: **Automatic Three-Dimensional Finite Element Mesh Generation via Modified Ray Casting**. International Journal of Numerical Methods in Engeneering , Vol 31 , 1991.

[24] H. Jin and R. I. Tanner: **Unstructured Tetrahedral Mesh Generation for Three-Dimensional Viscous Flow**. International Journal of Numerical Methods in Engeneering , Vol 39 , 1996.

[25] W. Hackbusch, S. A. Sauter: **Adaptive Composite Finite Elements for the Solution of PDEs containing non-uniformly distributed Micro-Scales**. Bericht 95-2, Berichtsreihe des Mathematischen Seminar Kiel, Universität Kiel.

[26] A. Ecker and W. Zulehner:**On the Smoothing Property for the Non-Symmetric Case**. Institutsbericht Nr. 489 ,Universität Linz, Institut für Mathematik, 1995.

[27] F. Kickinger: **Automatic Mesh Generation for 3D Objects**. Tecnical Report 96-1, Universität Linz, Institut für Mathematik, Arbeitsgruppe Numerische Mathematik und Optimierung, 1996.

[28] F. Kickinger: **Algebraic Multigrid for Discrete Elliptic Second Order Problems** *a program description*. Tecnical Report 96-5, Universität Linz, Institut für Mathematik, Arbeitsgruppe Numerische Mathematik und Optimierung, 1996.

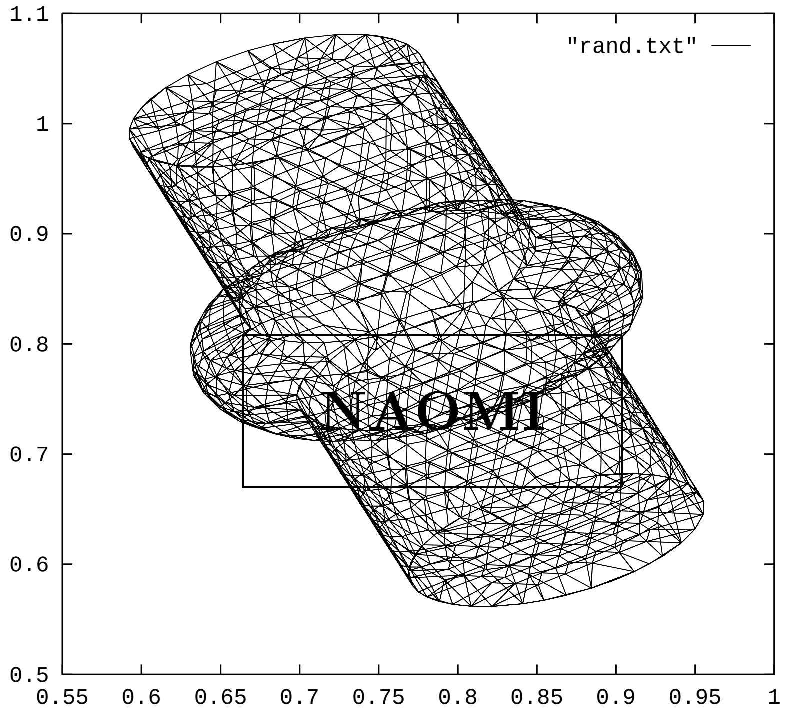

Figure 5: NAOMI - numerical automatical optimal meshing instrument

## A Meshes

- Figure 6 ··· Sphere
- Figure 7 ··· Torus
- Figure 8 ··· Cone
- Figure 9 ··· Hexaedron with Torus
- Figure 10 ··· Cylinder with Torus
- Figure 11 ··· Hexaedron with Zulinders and Toris
- Figure 12 ··· Local refinement in the 2D-case

Meshes generated by **NAOMI**

Figure 6: Sphere

Figure 7: Torus

Figure 8: Cone

Figure 9: Testobject1

Figure 10: Testobject2

Figure 11: Testobject3

Figure 12: Local refinement in the 2D case