Technische Universität Chemnitz-Zwickau
Sonderforschungsbereich 393
Numerische Simulation auf massiv parallelen Rechnern

Matthias Pester

Treating Curved Surfaces in a 3D Finite Element Program for Parallel Computers

Preprint SFB393/97-10

Preprint-Reihe des Chemnitzer SFB 393

SFB393/97-10 **April 1997**

Contents

Matthias Pester
TU Chemnitz-Zwickau
Fakulty of Mathematics
D-09107 Chemnitz

Germany

`pester@mathematik.tu-chemnitz.de`
`http://www.tu-chemnitz.de/~pester/`

1 Introduction

This paper provides both a simple and for many purposes sufficient method of defining and dealing with curved surfaces in a 3D Finite Element program ([1], [10], [12], [13]).

Automatic mesh generation for three-dimensional problems, especially with respect to the parallelization of computational algorithms, has become a research field of current interest where different approaches are applicable ([6], [7], [9]).

In this paper we use a method of describing the geometry of surfaces in a compact way and using this description in the mesh refinement algorithms to obtain the required non-linear intersection of edges and faces.

Although this method is not restricted to parallel computation, it is well-suited for this case. An important object in developing parallel algorithms is a well balanced distribution of computational effort as well as memory resources. Communications between processors are to be reduced to the inevitable minimum. Therfore, the *user mesh* should be as coarse as possible to avoid large sets of input data. In our present realisation this *coarse grid* determines the initial amount of communication by distributing only such coarse elements. The computational mesh is obtained locally on each processor for its subdomain consisting of one or more of the coarse grid elements. Moreover, this initial distribution determines the amount of communication via coupling faces and edges when solving the equation system ([2], [10]). Since the mesh refinement algorithm may consider the geometry of faces while generating new grid points, in many cases the starting grid may be very coarse (some examples are provided here).

Thus, we get any better approximation of curved surfaces only by increasing the level of (completely parallel) mesh refinement.

There were some contributions to the examples mentioned in section 4 by: Arnd Meyer, Uwe Reichel, Thomas Grund and Kornelia Pietsch. The problems involved by transmitting geometric data were first implemented in the mesh refinement algorithm by Frank Milde. A tool for geometrical transformations and merging of mesh data files is available by Dag Lohse. The mesh representations used in the figures were mostly produced with a built-in visualization tool of the FEM program mentioned above ([12]), and partially with an external program based on the Graphical Programming Environment (GRAPE, [14]).

2 Geometrical Definition in the Coarse Grid

The Finite Element Program *SPC-PMPo 3D* uses a standard input file in a generally free and human readable format which is described in [1]. This file, among other things, describes the topological properties of a three-dimensional domain using the leading keywords `#VERTEX, #EDGE, #FACE, #SOLID` for the hierarchy of k-dimensional manifolds (k-Faces, $k = 0, 1, 2, 3$). Each k-Face ($k > 0$) is defined by a line containing its *name* followed by a *type* specification and a certain number of $(k-1)$-faces given by their names. For an example see Figure 1.

Thus, we are able to assign special geometrical properties to a face (2-*face*) using the type specification in the definition line. This geometrical type specification can be understood as a pointer to a more detailed characteristic of the current face geometry which follows the keyword `#FACE_GEO`. First we restrict our considarations to simple

```
   ...
#VERTEX: 6
   1     0.0  -1.0   0.0
   2     1.0   0.0   0.0
   3     0.0   1.0   0.0
   4    -1.0   0.0   0.0
   5     0.0   0.0   1.0
   6     0.0   0.0   0.0
#EDGE: 13
   12     0     1  2
   23     0     2  3
   34     0     3  4
   41     0     4  1
   15     0     1  5
   25     0     2  5
   35     0     3  5
   45     0     4  5
   16     0     1  6
   26     0     2  6
   36     0     3  6
   46     0     4  6
   56     0     5  6
#FACE: 12
    1     0     3     12  25  15
    2     0     3     23  35  25
    3     0     3     35  45  34
    4     0     3     41  15  45
    5     0     3     12  26  16
    6     0     3     23  36  26
    7     0     3     34  46  36
    8     0     3     14  16  46
    9     0     3     16  15  56
   10     0     3     26  25  56
   11     0     3     36  35  56
   12     0     3     46  45  56
#SOLID: 4
      1   1     4     1   5   9  10
      2   1     4     2   6  10  11
      3   1     4     3   7  11  12
      4   1     4     4   8  12   9
   ...
```

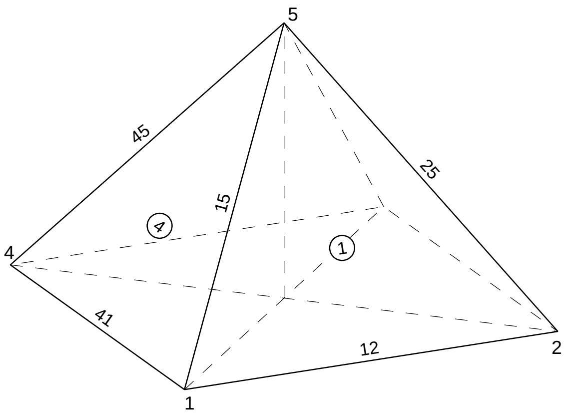


Figure 1: Example for the hierarchical description of a domain's topology: a square pyramid consisting of four tetrahedra

```
  ...
#FACE: 12
    1    2    3    12  25  15
    2    2    3    23  35  25
    3    2    3    35  45  34
    4    2    3    41  15  45
    5    0    3    12  26  16
    6    0    3    23  36  26
    7    0    3    34  46  36
    8    0    3    14  16  46
    9    0    3    16  15  56
   10    0    3    26  25  56
   11    0    3    36  35  56
   12    0    3    46  45  56
  ...
#FACE_GEO: 1
    2   31    7    0. 0. 1.   0. 0. 0.   1.
  ...
```

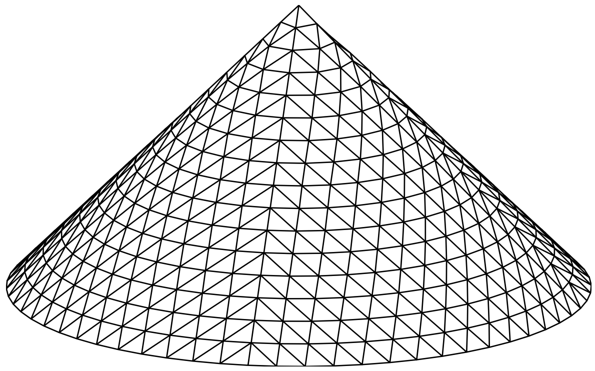


Figure 2: Modifications for a geometrical description of surfaces, the mesh from Fig. 1 after four steps of refinement

forms that can be defined by only a few parameters (cylinder, sphere, cone, rotational solid, torus)

Each different geometrical surface requires an entry in the `FACE_GEO` section of the standard file containing those few parameters to declare this surface. In the example from Fig. 1) only one (common) geometry for all four sides of the pyramid is necessary to define a cone. The changes in the input file are shown in Figure 2. The four faces are defined with a `<type>` entry "2", which refers to a line of the `FACE_GEO` section that begins with the `<name>` "2" .

The structure of an entry for a geometrical description is similar to the In the example the data line

```
2   31   7    0. 0. 1.   0. 0. 0.   1.
```

means in the given sequence:

2	name of the geometry (line),
31	code for the kind of surface, here a cone
7	number of additional parameters to define the geometry
0. 0. 1.	coordinates of the vertex of the cone
0. 0. 0.	coordinates of another point on the rotational axis
1.	radius of the cone at the distance of the second point

Thus, the general structure of a geometry definition line is

```
<name>  <type>  <n>  <parameter_1> ... <parameter_n>
```

where `<name>` is referred to by the `<type>` entry of a face. For the beginning, some geometries were attached to `<type>` classes with a stride of 10

`1 ... 9`	for a plane
`11 ... 19`	for a cylinder
`21 ... 29`	for a sphere
`31 ... 39`	for a cone
`41 ... 49`	for a rotational solid (ellipsoid, hyperboloid)
`51 ... 59`	for a torus

The single steps within a type class (form 1 to 9) may be used for varying the kind of parameters which are prefered by different users to define the same geometry, e. g. define a sphere by center and radius or by a second point instead of the radius, or define an axis either by two points or by a point and a vector.

The current implementation includes specifications for each of the just noted types of surfaces. They are listed in Table 1.

`<type>`	`<n>`	meaning of the parameters
1	6	$(n_x, n_y, n_z)\,,(x_0, y_0, z_0)$ normal vector and one point of the plane
2	6	$(x_0, y_0, z_0)\,,(n_x, n_y, n_z)$ point of the plane and normal vector
11	7	$(x_1, y_1, z_1)\,,(x_2, y_2, z_2)\,,r$ two points on the axis of the cylinder, and radius
21	4	$(x_m, y_m, z_m)\,,r$ center and radius of the sphere
31	7	$(x_1, y_1, z_1)\,,(x_2, y_2, z_2)\,,r_2$ vertex of the cone, 2nd point on the axis, and radius at this distance
41	8	$(x_1, y_1, z_1)\,,(x_2, y_2, z_2)\,,A, B$ two points defining a rotational axis, where the first point is considered to be the origin of a local (ξ, η)-coordinate system, and the two parameters for the equation of the conic section (the rotating curve): $\pm\frac{\xi^2}{a^2} \pm \frac{\eta^2}{b^2} = 1, \quad \text{where } A = \pm a^2, B = \pm b^2$ Note: $B < 0 \rightarrow$ one-sided, $A < 0 \rightarrow$ two-sided hyperboloid
51	8	$(x_1, y_1, z_1)\,,(n_x, n_y, n_z)\,,r_1, r_2$ center of the torus, normal vector of the base plane, radius of rotation (distance of the center of the rotating circle) and radius of the rotating circle

Table 1: Overview on surface types currently implemented
(see `http://www.tu-chemnitz.de/~pester/facegeo.html`)

By default, there is no need to declare planes as special surfaces, since mesh refinement by linear intersection automatically generates new points in the same plane. An exception, however, is the case of edges at the intersection of a curved and a plane surface. Depending on the algorithm used for generating new mesh points on the curved surface, the plane must be declared explicitly. Figure 3 illustrates the possible effect.

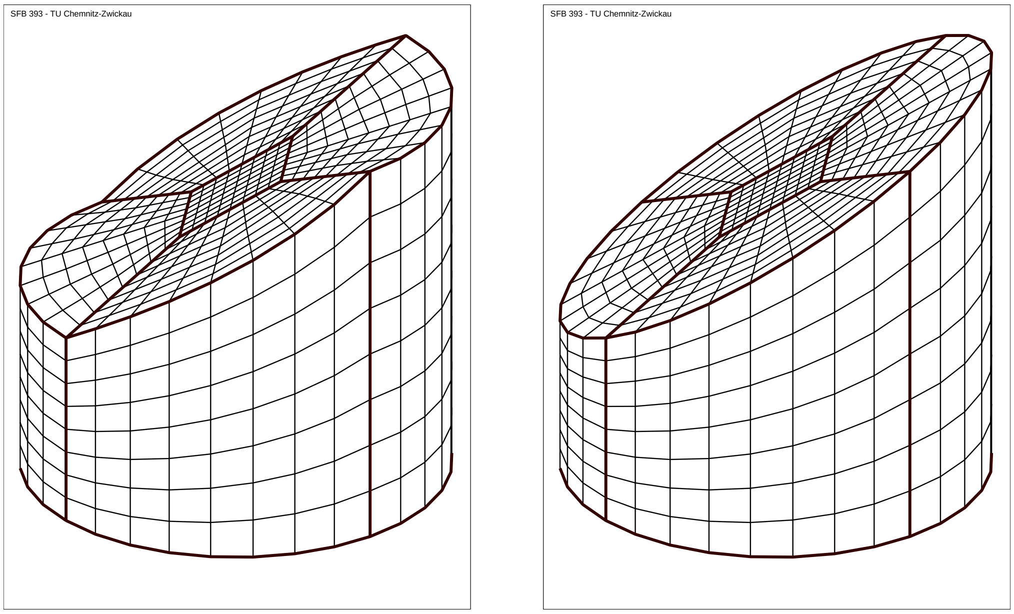


Figure 3: Comparison of two meshes where only in the right picture the cutting plane is declared explicitly; the coarse grid is marked by stronger lines

3 Geometry Related Mesh Refinement

3.1 General Aspects

The consideration of faces with special geometrical properties is independent of the kind of mesh refinement (uniform or adaptive). At the initial stage we have a coarse mesh

$$\Omega_0 = \{V_0, E_0, F_0, S_0\}\,,$$

consisting of

$$\begin{array}{lll} \text{a set} & V_0 = \left\{v_k^{(0)}\right\} & \text{of nodes,} \\ \text{a set} & E_0 = \left\{e_k^{(0)}\right\} & \text{of edges, defined by 2 nodes,} \\ \text{a set} & F_0 = \left\{f_k^{(0)}\right\} & \text{of faces, defined by 3 or 4 edges, and} \\ \text{a set} & S_0 = \left\{s_k^{(0)}\right\} & \text{of volume elements, defined by 4 or 6 faces.} \end{array}$$

During mesh refinement there is generated a sequence of meshes $\Omega_i = \{V_i, E_i, F_i, S_i\}$ for $i = 1, 2, 3, \ldots$, satisfying the common hierachical relations between levels i and $i+1$:

$$V_i \subset V_{i+1}, \quad e_k^{(i)} = \bigcup_{j=1}^{n_1} e_{k_j}^{(i+1)}, \quad f_k^{(i)} = \bigcup_{j=1}^{n_2} f_{k_j}^{(i+1)}, \quad s_k^{(i)} = \bigcup_{j=1}^{n_3} s_{k_j}^{(i+1)},$$

where n_q denotes for a q-dimensional object (q-face) the number of its q-dimensional sub-faces of the next level (for uniform refinement of tetrahedra and hexahedra $n_1 = 2, n_2 = 4, n_3 = 8$).

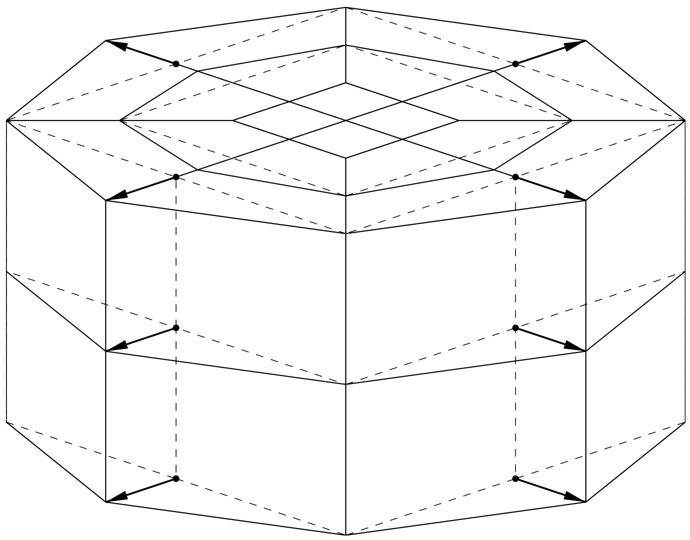


Figure 4: Projection of new generated nodes to the surface of the cylinder

Furthermore, there exists a set $G = \{g_j\}$ of all geometry definitions needed to describe the surface of the solid. This set does not change while the mesh is refined. Some of the coarse grid faces $f_k^{(0)}$ are marked by a *face attribute* g_j which is transmitted to the child faces $f_{k_j}^{(i+1)}$ of the refined mesh. Obviously, the face attribute g_j has to be considered for all nodes arising from the intersection of the face $f_k^{(i)}$ and its bounding edges. Such new nodes $v_{k_j}^{(i+1)}$ are first approximated by a linear refinement (bisection of an edge or, in the cas of hexahedra, center points of faces or volumes) and thereafter they are projected to the surfaces defined by g_j (Fig. 4). The next section will give several examples for the way to perform this projection.

Edges ever belong to two or more faces and so they can belong to more than one curved surface as their intersection curve. For practical relevant examples it is sufficient to restrict to the case of intersecting two different face geometries in one edge. Thus, up to two face attributes can be attached to an edge. The case of two different geometries requires a special consideration. Instead of the projection to a surface we have to perform a projection to the intersection curve of two faces.

Once the initial coarse grid and geometry specifications are distributed among the processors of a parallel computer, the whole process of mesh refinement can be executed locally without any communication.

3.2 Tetrahedral Meshes

With respect to their topology tetrahedral meshes can be treated in a very simple way. For the transmission of geometrical properties the following cases have to be considered (cf. Fig. 5):

- New nodes may occur only by intersection of edges. If the edge was marked by at least one geometrical attribute g_j (caused by a curved surface at one of the

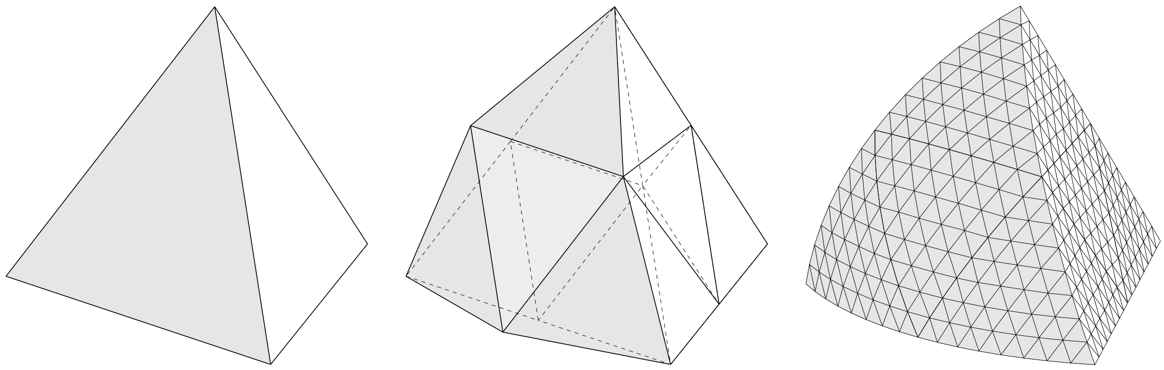


Figure 5: Passing a face geometry to the child faces of a tetrahedron

triangles it belongs to) the linear bisection of the edge has to be *improved* by shifting the node to the surface defined by g_j.

- The child faces obtain the geometry information of their original face, and child edges get those one or two geometry attributes being attached to the original edge.
- New edges inside a curved face receive the attribute g_j of that face.
- New edges and faces inside the tetrahedron are considered to be free of any geometrical information. Otherwise we would have to find a transformation of a boundary geometry to an inner geometry respecting a certain set of neighbored tetrahedra.

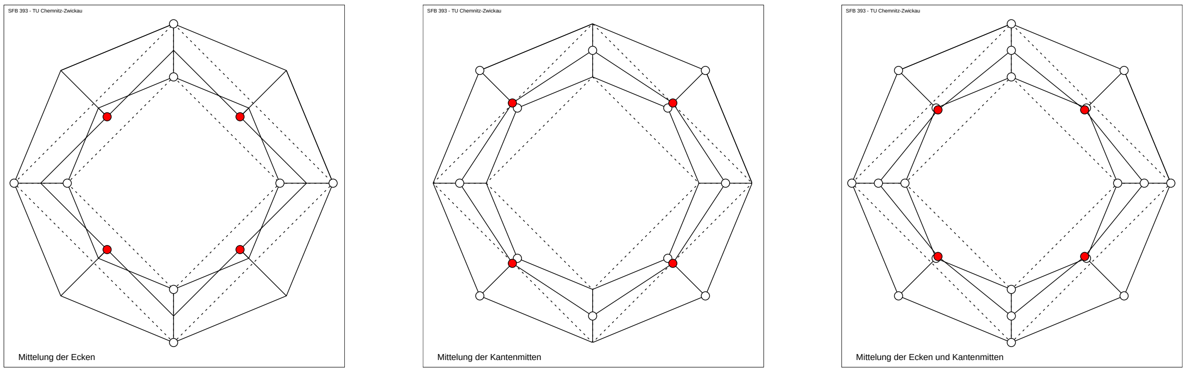
Pentahedra can be treated in the same way as tetrahedra with respect to the transmission of face and edge geometries.

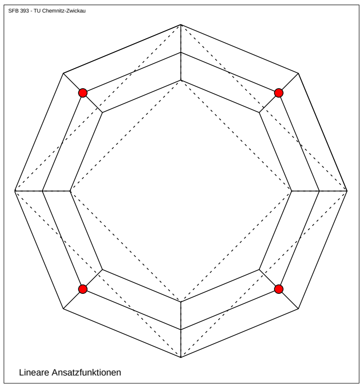
3.3 Hexahedral Meshes

A hexahedral mesh requires some more attention than a tetrahedral mesh. In addition to the bisection of edges there have to be determined the center points of faces or volumes (hexahedra) to become new vertices in the next level.

Having only straight edges, the linear refinement of a hexaeder would be sufficient, i. e. compute

- the center point of edges from their 2 vertices;
- the face center points from their 4 vertices or the centers of its four edges; the result is the same for plane faces
- the centerpoint of the hexahedron from its 8 vertices or from the 6 face center points.





above: arithmetical mean of the coordinates
of vertices,
of edge centers,
of vertices and edge centers.

left: using form functions over vertices and edge centers

Dotted lines mark the four hexahedral elements to be refined.

Figure 6: Different results by different computation of the face center points of hexahedral elements in a piece of a flue

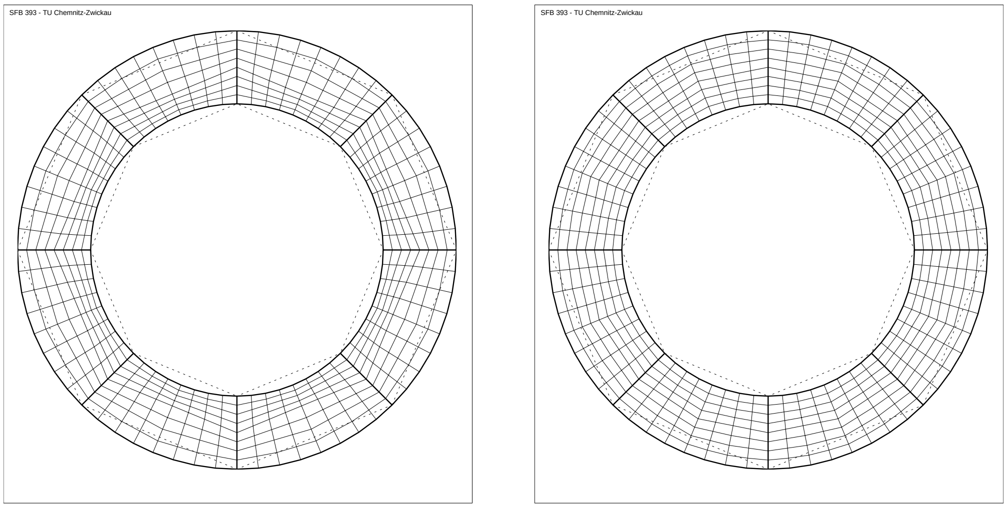


Figure 7: The effect of different computations of the face center points after a three-level refinement of the 8-element coarse grid; (left: arithmetic mean values)

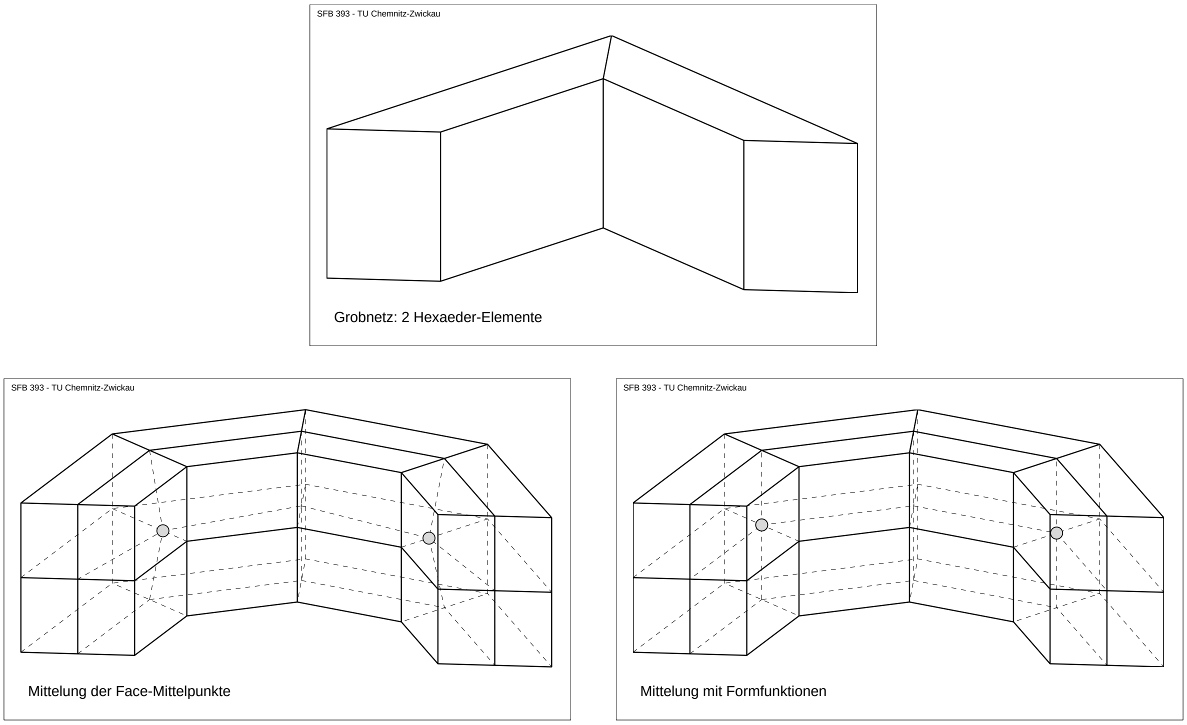


Figure 8: Center of a hexhedron with curved faces;
above: two hexahedral elements to be refined
left: arithmetical mean is not in the center
right: center determined by form functions

If at least one face of a hexahedron is part of a curved surface the computation of mean values may lead to inacceptable results. First we consider the face center points of a hexahedron which has to be divided in to 8 sub-hexahedra.

If such a point belongs to a curved face it can be treated in a way similar to the points on edges (i. e. projection to the surface defined by a g_j). The more critical case occurs if one or more of the bounding edges of the face are curved. As Figure 6 illustrates the arithmetical mean may give bad results, also by refering to the centers of edges which are already shifted corresponding to the real geometry. Therefore, those inner points should be determined by a linear combination

$$x^{(m)} = \sum_{i=1}^{8} x_i \varphi_i^{(8)} \left(\xi^{(m)} \right),$$

where the $\varphi_i^{(8)}$ denote the quadratic form functions of an 8-node SERENDIPITY element on the reference element $[0,1] \times [0,1]$ (cf. [15], [5]) and $\xi^{(m)} = \left(\frac{1}{2}, \frac{1}{2} \right)$ is the center of this reference element. This approach leads to the computation of a face center as a linear function of the 4 vertices $x_k^{(v)}$ and the 4 edge centers $x_k^{(e)}$

$$x^{(m)} = \frac{1}{2} \sum_{k=1}^{4} x_k^{(e)} - \frac{1}{4} \sum_{k=1}^{4} x_k^{(v)}$$

A similar behavior is found for the center of a hexahedral element bounded by curved surfaces (cf. Fig. 8). A center point corresponding to the geometry of the element can be obtained as a linear combination again

$$x^{(m)} = \sum_{i=1}^{20} x_i \varphi_i^{(20)} \left(\xi^{(m)}\right)$$

of the appropriate form functions $\varphi_i^{(20)}$ of the 20-node SERENDIPITY element on the reference element $[0,1] \times [0,1] \times [0,1]$.

The evaluation for the center of the reference element $\xi^{(m)} = \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$ leads to the following linear combination of the 8 vertices $x_k^{(v)}$ and the 12 center points $x_k^{(e)}$ of the edges

$$x^{(m)} = \frac{1}{4} \sum_{k=1}^{12} x_k^{(e)} - \frac{1}{4} \sum_{k=1}^{8} x_k^{(v)} .$$

This calculation gurantees the central position of the new inner vertex with respect to all directions of the hexahedral element. Otherwise, it could be moved to one side as shown in the left part of Fig. 8 where it is closer to the common center of the curvature of inner and outer surfaces.

4 Projections to Curved Faces

By means of some examples we will now consider a few algorithms for the projection of an approximated new vertex $\tilde{x}$ to the surface given by a geometry descriptor g, or to a curved edge at an intersection of two curved faces given by their two descriptors g_1, g_2.

An implementation of such a projection is simply a function

$$x^* = F(\tilde{x}, g)$$

of the coordinates $\tilde{x}$ obtained by linear subdivision, and the geometry descriptor g given with the coarse mesh. For edges at an intersection of two curved faces we have to take in acount two geometries:

$$x^* = F(\tilde{x}, g_1, g_2) .$$

4.1 Projection to One Face

4.1.1 Cylinder

According to Table 1 (p. 4), the cylinder is defined by two points p_1, p_2 of the axis and its radius r, i. e. $g_{11} = \{p_1, p_2, r\}$. In this case the transformation F is realized by radial shifting the point $\tilde{x}$ into the point x^* (Fig. 9).

The distance of the point $\tilde{x}$ from the axis has to be increased to r into the direction of the vector $\vec{v} = \tilde{x} - p_0$. We denote

$$\begin{array}{rcll} \vec{v}_a & = & p_2 - p_1 & \text{direction of the axis} \\ \vec{v}_0 & = & \tilde{x} - p_0 & \text{vector from the axis to } \tilde{x}, \quad \vec{v}_0 \perp \vec{v}_a \\ \vec{v}_1 & = & \tilde{x} - p_1 & \text{vector from an axis point to } \tilde{x} \end{array}$$

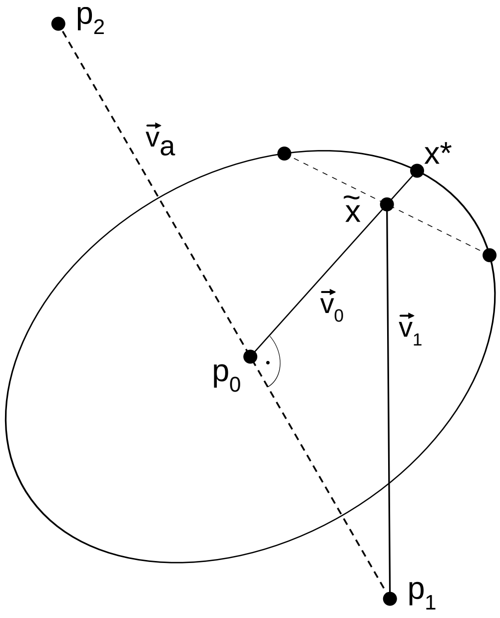


Figure 9: Projection of the point $\tilde{x}$ to the cylinder surface

The point p_0 (closest to $\tilde{x}$ point of the axis) is not needed explicitly. The shift is realized by simple vector operations. Given the values $p_1, p_2, r, \tilde{x}$ we determine x^* by the following algorithm:

$$
\begin{aligned}
\vec{v}_a &:= p_2 - p_1 \\
\vec{v}_1 &:= \tilde{x} - p_1 \\
\alpha &:= (\vec{v}_a \cdot \vec{v}_a) \\
\beta &:= (\vec{v}_a \cdot \vec{v}_1) \\
\vec{v}_0 &:= \vec{v}_1 - \frac{|\vec{v}_1| \cos \sphericalangle(\vec{v}_a, \vec{v}_1)}{|\vec{v}_a|} \vec{v}_a \;=\; \vec{v}_1 - \frac{\beta}{\alpha} \vec{v}_a \\
x^* &:= \tilde{x} + (r - |\vec{v}_0|) \frac{\vec{v}_0}{|\vec{v}_0|} \;=\; \tilde{x} + \left(\frac{r}{|\vec{v}_0|} - 1 \right) \vec{v}_0
\end{aligned}
$$

with an inessential computational effort of three dot products $(\alpha, \beta, |\vec{v}_0|)$ and four `saxpy` operations for short coordinate vectors.
Figure 10 shows a coarse initial grid changing to a cylinder by mesh refinement.

4.1.2 Sphere

A spherical surface is defined by $g_{21} = \{p_m, r\}$. The transformation from $\tilde{x}$ to x^* is as simple as above: a radial shift in the direction of $\vec{v}_0 = \tilde{x} - p_m$ up to a distance of r from the center p_m. The short algorithm:

$$
\begin{aligned}
\vec{v}_0 &:= \tilde{x} - p_m \\
x^* &:= p_m + r \frac{\vec{v}_0}{|\vec{v}_0|}
\end{aligned}
$$

Examples are shown in the figures 11 and 12. There is no need in the example of Fig. 12 to declare the three intersecting planes as special geometry because each of them

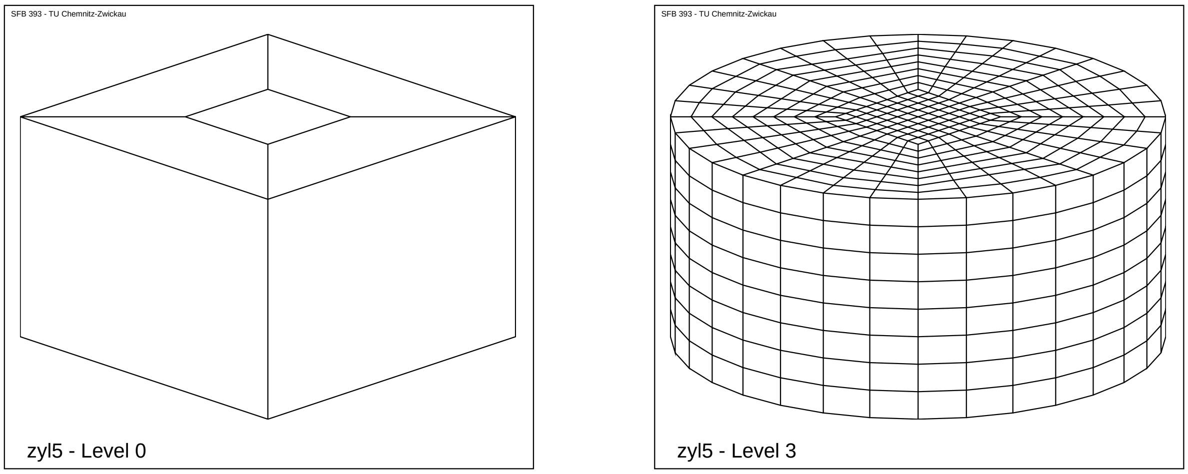


Figure 10: `mesh4/zyl5.std`: 5 Hexahedral elements, 4 faces belonging to one cylindric surface

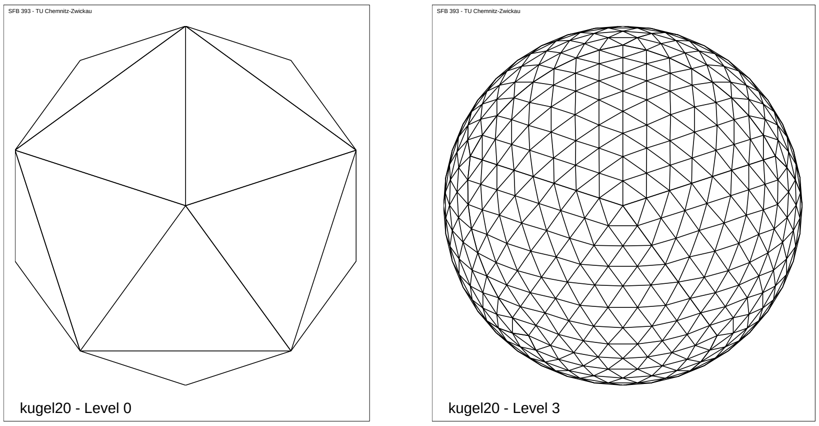


Figure 11: `mesh3/kugel20.std`: 20 tetrahedral elements (icosahedron), spherical surface

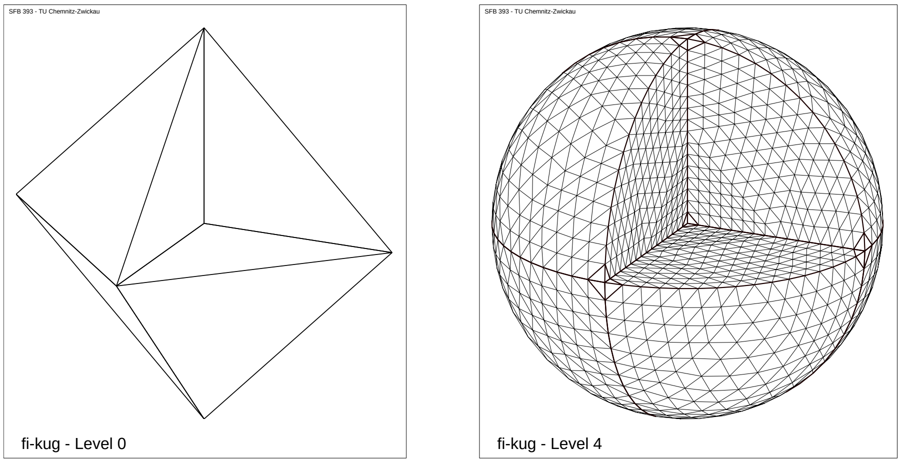


Figure 12: `mesh3/fi-kug.std`: 7 tetrahedral elements, sphere with one sector extracted

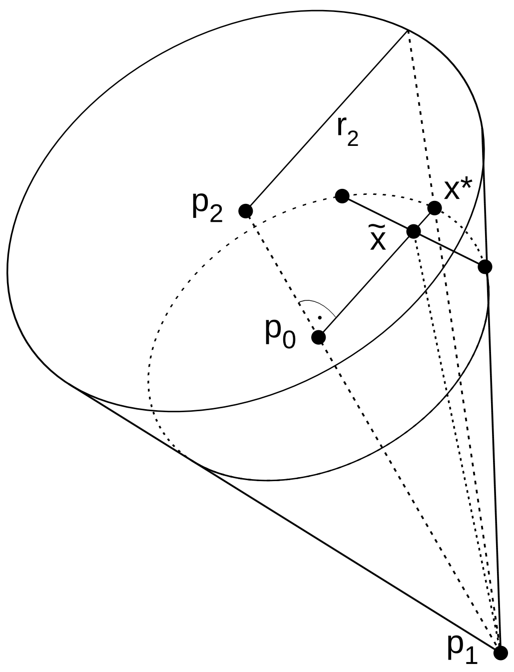


Figure 13: Projection of the point $\tilde{x}$ to the cone

contains the center of the sphere, so the radial shift of any point on the outer edges of intersection does not leave the plane.

4.1.3 Cone

A cone is defined by $g_{31} = \{p_1, p_2, r_2\}$. To shift the approximating point $\tilde{x}$ orthogonally to the cone's axis is the same task as for a cylinder, except that the distance from the axis is variable - a linear function of the height over the base of the cone which is given by p_1 (Fig. 13). The explicit coordinates of p_0 are not needed again.
The marginally modified algorithm follows:

$$\begin{aligned}
\vec{v}_a &:= p_2 - p_1 \\
\vec{v}_1 &:= \tilde{x} - p_1 \\
\alpha &:= (\vec{v}_a \cdot \vec{v}_a) \\
\beta &:= (\vec{v}_a \cdot \vec{v}_1) \\
\vec{v}_0 &:= \vec{v}_1 - \frac{\beta}{\alpha}\vec{v}_a \\
r_0 &:= \frac{|p_0 - p_1|}{|\vec{v}_a|} r_2 = \frac{\beta}{\alpha} r_2 \\
x^* &:= \tilde{x} + (r_0 - |\vec{v}_0|)\,\frac{\vec{v}_0}{|\vec{v}_0|} \;=\; \tilde{x} + \left(\frac{r_0}{|\vec{v}_0|} - 1\right)\vec{v}_0
\end{aligned}$$

4.1.4 Hyperboloid, Ellipsoid

Here we consider the surfaces of rotational hyperboloids or ellipsoids with their geometry defined by the parameters $g_{41} = \{p_1, p_2, A, B\}$. Assume p_1 to be the origin of a local

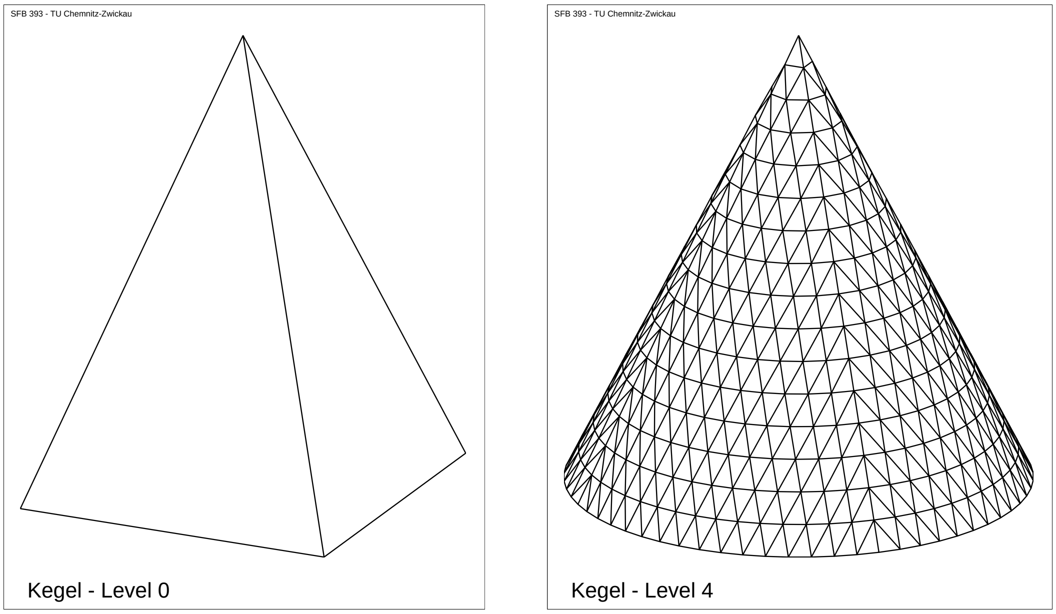


Figure 14: `mesh3/kegel4.std`: 4 tetrahedral elements, four faces defined as a cone

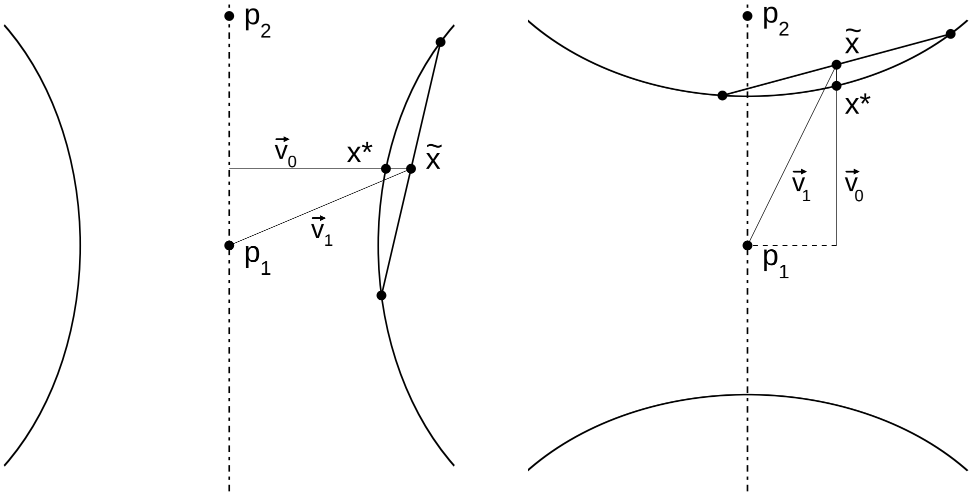


Figure 15: Projection of $\tilde{x}$ to a one- or two-sided rotational hyperboloid

(ξ, η)-coordinate system, where the η-axis defined by $\overrightarrow{p_1 p_2}$ is the rotational axis. The ξ-axis the *rotating* axis. The current (ξ, η)-plane is determined by the (fixed) η-axis and the approximating point $\tilde{x}$. The rotating curve is defined by

$$\pm\frac{\xi^2}{a^2} \pm \frac{\eta^2}{b^2} = 1, \qquad \text{mit } A = \pm a^2, B = \pm b^2.$$

For $A > 0, B > 0$ a rotational ellipsoid is defined, $A > 0, B < 0$ defines a one-sided hyperboloid (Fig. 15, left) and $A < 0, B > 0$ a two-sided hyperboloid (Fig. 15, right). We will realize the transformation of $\tilde{x}$ to x^* by a shift either orthogonal or parallel to the rotational axis (parallel for the latter of the cases mentioned above). Thus, we use

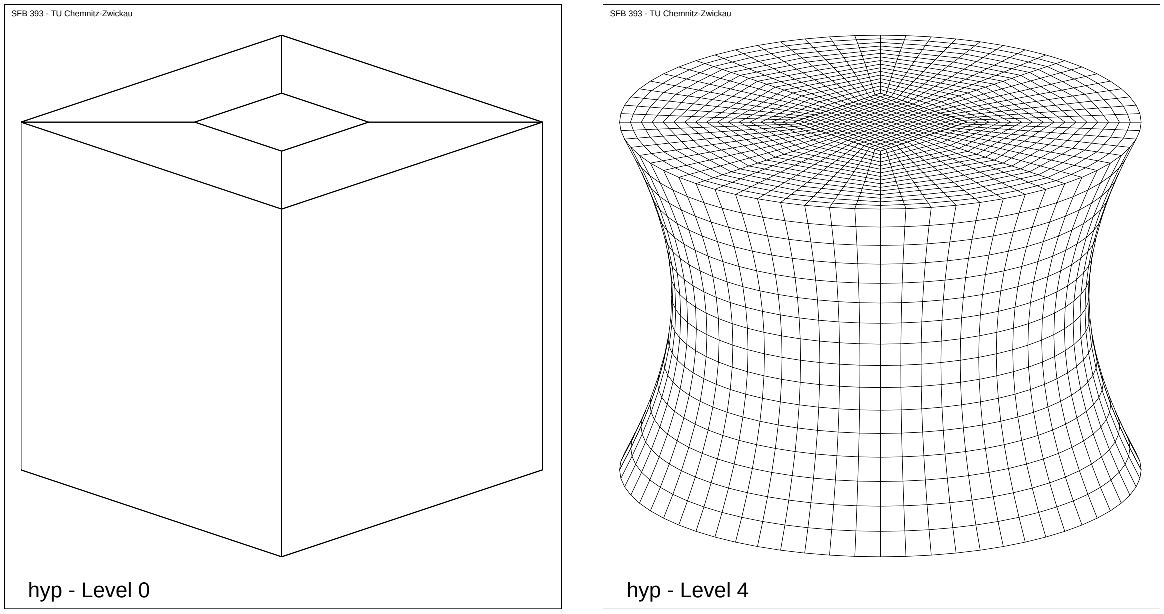


Figure 16: `mesh4/hyp.std`: 5 hexahedral elements resulting in a hyperboloid

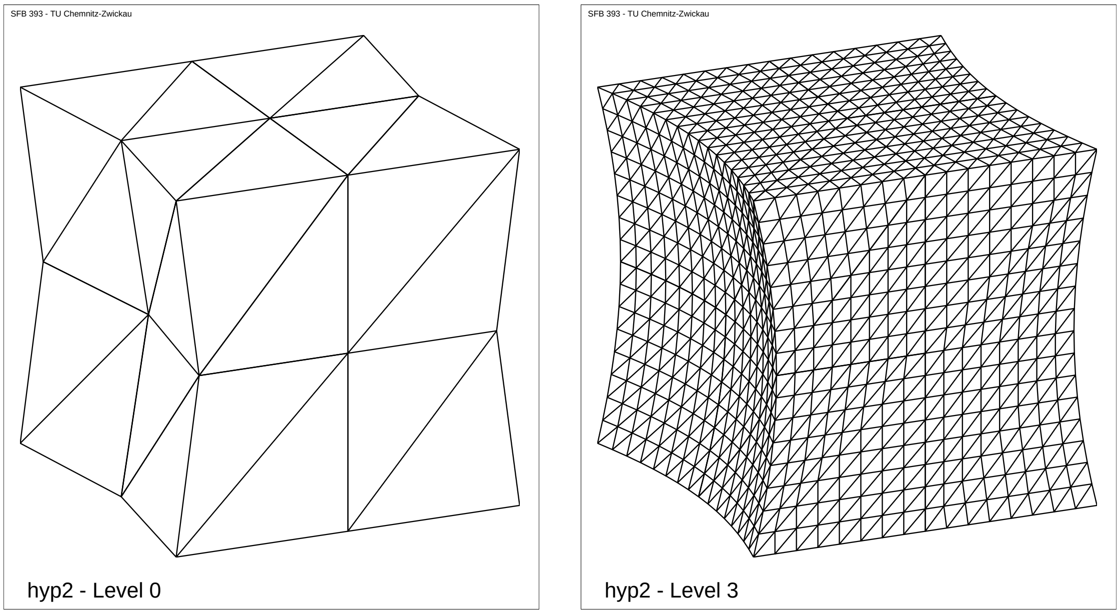


Figure 17: `mesh3/hyp2.std`: Two opposite faces belonging to a two-sided hyperboloid

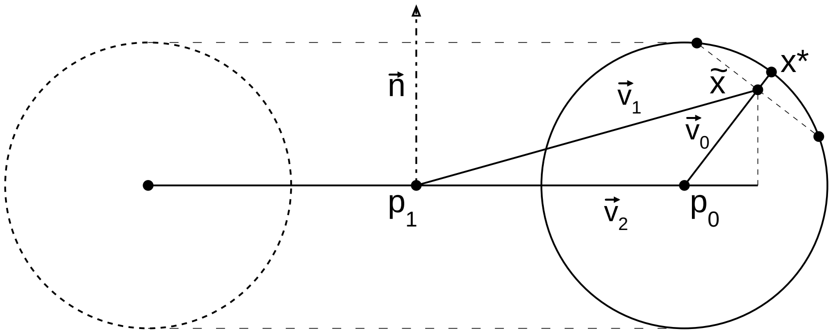


Figure 18: Projection of the point $\tilde{x}$ to a torus

the following algorithm:

$$\begin{aligned}\vec{v}_a &:= p_2 - p_1\\ \vec{v}_1 &:= \tilde{x} - p_1\\ \alpha &:= (\vec{v}_a \cdot \vec{v}_a)\\ \beta &:= (\vec{v}_a \cdot \vec{v}_1)\end{aligned}$$

case $A > 0$: $(\vec{v}_0 \perp \vec{v}_a)$

$$\begin{aligned}\vec{v}_0 &:= \vec{v}_1 - \frac{\beta}{\alpha}\vec{v}_a\\ \eta^2 &:= \frac{\beta^2}{\alpha}\\ r &:= \sqrt{A\left(1 - \frac{\eta^2}{B}\right)}\end{aligned}$$

case $A < 0$: $(\vec{v}_0 \parallel \vec{v}_a)$

$$\begin{aligned}\vec{v}_0 &:= \frac{\beta}{\alpha}\vec{v}_a\\ \xi^2 &:= (\vec{v}_1 \cdot \vec{v}_1) - \frac{\beta^2}{\alpha}\\ r &:= \sqrt{B\left(1 - \frac{\xi^2}{A}\right)}\end{aligned}$$

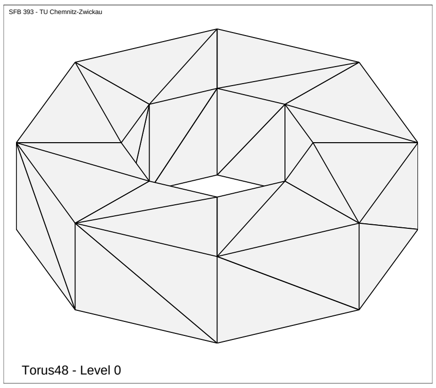
$$x^* := \tilde{x} + \left(\frac{r}{|\vec{v}_0|} - 1\right)\vec{v}_0$$

Examples are shown in Figures 16 and 17.

Remark. The criteria used to select a direction for the projection orthogonal or parallel to the axis, in general yields good results. This is obviously clear for points near the vertex of the hyperbola. Otherwise, far points are very close to the asymptote, so the distance between $\tilde{x}$ and x^* is very small.

4.1.5 Torus

The geometry of the surface of a torus is defined by $g_{51} = (p_1, \vec{n}, r_1, r_2)$. The torus is origined by rotating circle with the radius r_2 around an axis at a distance of r_1 between the axis and the circle's center. The direction of the axis is given by $\vec{n}$ and p_1 is the intersection point of the axis with the rotational plane. The transformation of $\tilde{x}$ to x^* is done by a radial projection within the rotating circle. The two points $p_1, \tilde{x}$ and the vector $\vec{n}$ define a plane, also containig the center p_0 of that circle. Then x^* is found at



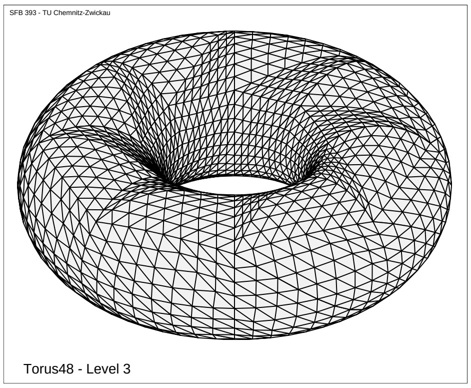
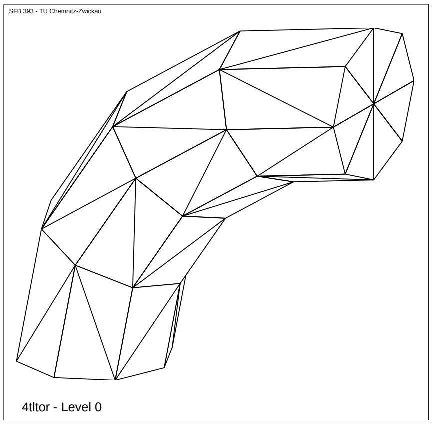


Figure 19: `mesh3/torus48.std`: 48 tetrahedral elements forming a torus



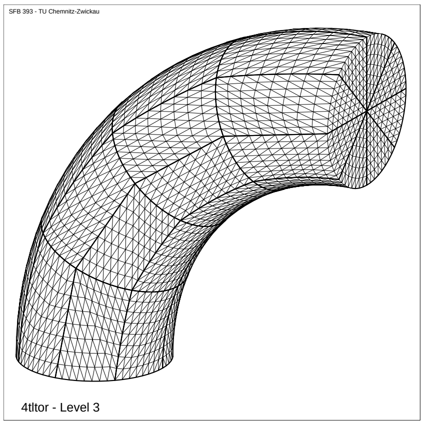


Figure 20: `mesh3/4tltor8.std`: 96 tetrahedral elements, quarter of a torus,

a distance of r_2 from p_0 in the direction $\overrightarrow{p_0\tilde{x}}$ (Abb. 18) by the following algorithm:

$$
\begin{aligned}
\vec{v}_1 &:= \tilde{x} - p_1 \\
\alpha &:= (\vec{n} \cdot \vec{n}) \\
\beta &:= (\vec{n} \cdot \vec{v}_1) \\
\vec{v}_2 &:= \vec{v}_1 - \frac{\beta}{\alpha}\vec{n} \\
\vec{v}_0 &:= \vec{v}_1 - \frac{r_1}{|\vec{v}_2|}\vec{v}_2 \qquad (=\ \tilde{x} - p_0) \\
x^* &:= \tilde{x} + (r_2 - |\vec{v}_0|)\frac{\vec{v}_0}{|\vec{v}_0|} \ = \ \tilde{x} + \left(\frac{r_2}{|\vec{v}_0|} - 1\right)\vec{v}_0
\end{aligned}
$$

The torus in Figure 21 has some *irregular* displacements of nodes on the surface caused by the radial shift of $\tilde{x}$ to x^* for such points $\tilde{x}$ that arise from a subdivision of *horizontal* edges (parallel to the base plane).

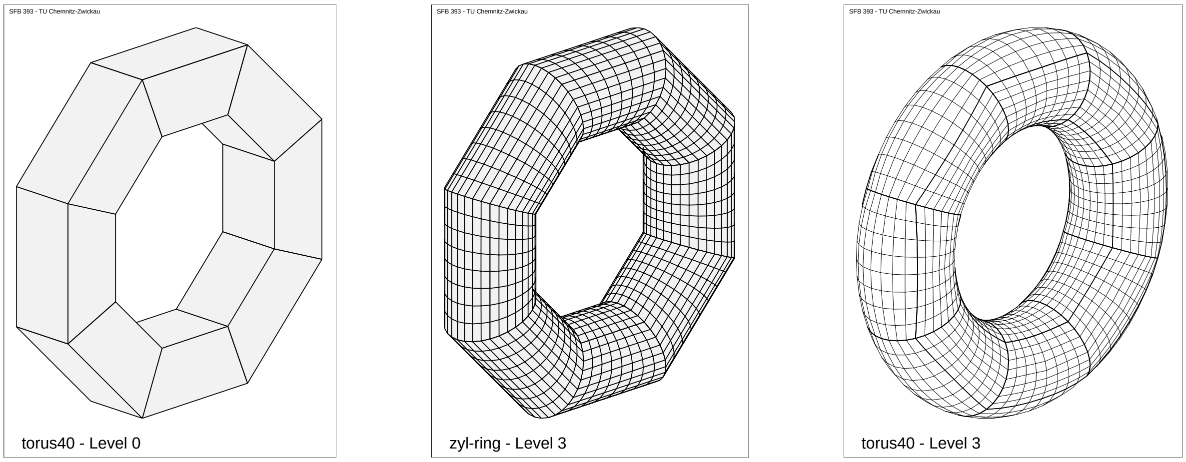


Figure 21: `mesh4/torus40.std, mesh4/ring40.std`:
40 hexahedral elements; using different face geometries to obtain a ring from 8 cylindric pieces or a torus.

4.2 Projection to Intersection Curves

4.2.1 Cylinder and Plane

If a cylinder is cut by a plane non-orthogonally[1] to the axis, the shift of $\tilde{x}$ to the destination point x^* on the cylinder has to follow the plane (Fig. 22), i. e. the shift direction can be obtained as a projection of the radial direction into the cutting plane parallel to the axis of the cylinder.

There are given the geometrical parameters $g_{11} = \{p_1, p_2, r\}$, $g_{01} = \{\vec{n}, p_0\}$ where the point p_0 is unnecessary for the calculations. We may assume the complete edge being refined to belong to the given plane and so the point $\tilde{x}$ found by linear subdivision. Independent of the definition in g_{01} we denote by p_0 the intersection point of the plane and the axis. Now, different to the simple case of a cylinder (Section 4.1.1), we have **not** $\vec{v}_0 \perp \vec{v}_a$, but $\vec{v}_0 \perp \vec{n}$. The new point $x^* = F(\tilde{x}, g_{01}, g_{21})$ is determined as follows:

$$\begin{aligned}
\vec{v}_a &:= p_2 - p_1 \\
\text{If } (\vec{v}_a \cdot \vec{n}) &= 0, \text{ the plane is parallel to the cylinder's axis} \\
& \text{hence } x^* = \tilde{x},
\end{aligned}$$

otherwise:

$$\begin{aligned}
\vec{v}_1 &:= \tilde{x} - p_1 \\
\alpha_a &:= (\vec{v}_a \cdot \vec{v}_a) \\
\alpha_n &:= (\vec{n} \cdot \vec{n}) \\
\beta_a &:= (\vec{v}_1 \cdot \vec{v}_a) \\
\beta_n &:= (\vec{v}_1 \cdot \vec{n}) \\
\vec{v}_2 &:= \vec{v}_1 - \frac{\beta_a}{\alpha_a} \vec{v}_a
\end{aligned}$$

[1] An orthogonal cutting plane may be explicitly defined ($\vec{n} \| \vec{v}_a$) giving, however, the same results as the consideration of only the cylinder geometry with the projection from Section 4.1.1.

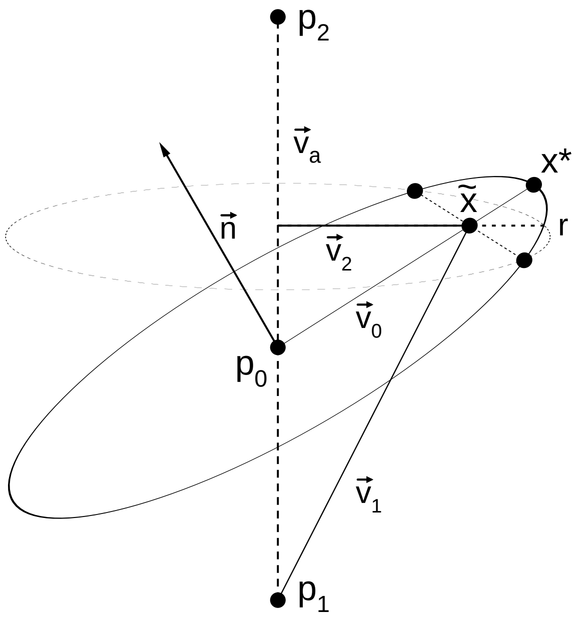


Figure 22: Projection of the point $\tilde{x}$ to the intersection of a plane and a cylinder

$$\begin{aligned}
\vec{v}_0 &:= \vec{v}_1 - \frac{\beta_n}{\alpha_n}\vec{n} \\
x^* &:= \tilde{x} + \frac{(r - |\vec{v}_2|)}{|\vec{v}_2|}\frac{\vec{v}_0}{|\vec{v}_0|} = \tilde{x} + \left(\frac{r}{|\vec{v}_2|} - 1\right)\frac{\vec{v}_0}{|\vec{v}_0|}
\end{aligned}$$

An alternate method could try to fix the point x^* on the intersection curve in such a way that the distance $|x^* - \tilde{x}|$ becomes a minimum. This requires some more complicated calculations without any remarkable improvement of the approximation of the curve ([8]).

4.2.2 Sphere and Plane

A planar cut in a sphere requires that any edge point x^* is obtained from its linear approximation $\tilde{x}$ by a shift in a radial direction within the cutting plane, i. e. within a circle with the center in M_0 (the projection of the sphere's center to the plane). Given the geometrical parameters $g_{01} = \{\vec{n}, p_0\}$ and $g_{21} = \{M, r\}$ the transformation $x^* = F(\tilde{x}, g_{01}, g_{21})$ can be realized in the following way where M_0 and r_0 denote the center and the radius of the circle at the intersection of the sphere and the plane:

$$\begin{aligned}
\vec{v}_1 &:= \tilde{x} - M \\
\alpha &:= (\vec{n} \cdot \vec{n}) \\
\beta &:= (\vec{v}_1 \cdot \vec{n}) \\
r_0 &:= \sqrt{r^2 - \left|\overline{MM}_0\right|^2} = \sqrt{r^2 - \frac{\beta^2}{\alpha}} \\
\vec{v}_0 &:= \vec{v}_1 - \left|\overline{MM}_0\right|\frac{\vec{n}}{|\vec{n}|} = \vec{v}_1 - \frac{\beta}{\alpha}\vec{n} \\
x^* &:= \tilde{x} + (r_0 - |\vec{v}_0|)\frac{\vec{v}_0}{|\vec{v}_0|} = \tilde{x} + \left(\frac{r_0}{|\vec{v}_0|} - 1\right)\vec{v}_0
\end{aligned}$$

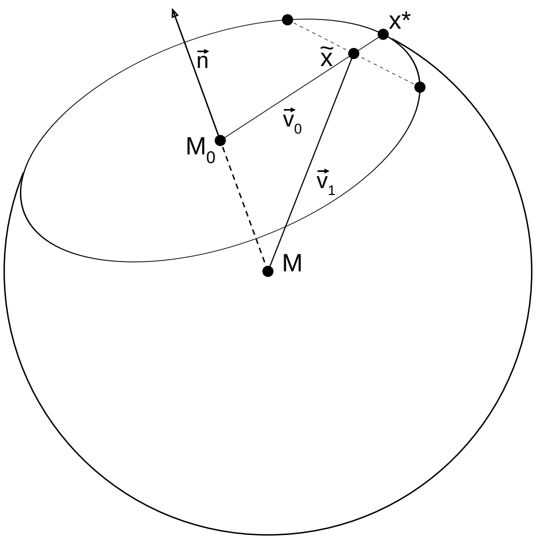


Figure 23: Projection pf the point $\tilde{x}$ to the intersection of a plane and a sphere

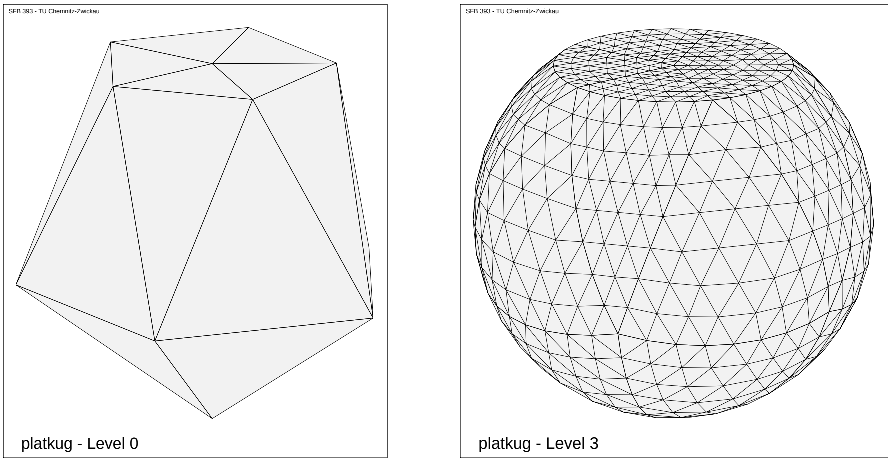


Figure 24: `mesh3/platkug.std`: 20 tetrahedral elements, sphere with cutting plane.

The example in Figure 24 has the same toplogical structure of the coarse grid as the sphere in Figure 11, with the distinction that the upper faces are defined to belong to a plane.

4.2.3 Cone and Plane

The planar cut through a cone is only marginally more complicated to realize than the cut of a cylinder (Fig. 25). The most simlple way is to use the projection from 4.1.3 to shift $\tilde{x}$ into an auxiliary point x_h on the cone, orthogonally to the axis of the cone. Thereafter, we determine x^* as the intersection of the line from the vertex p_1 through x_h with the given plane. With the given values $\tilde{x}$, $g_{31} = \{p_1, p_2, r_2\}$, $g_{01} = \{\vec{n}, p_0\}$ we use the following algorithm:

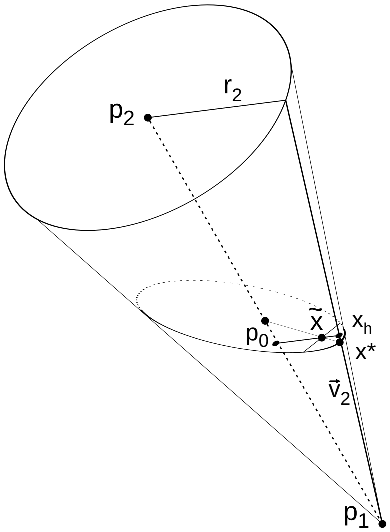


Figure 25: Projection of the point $\tilde{x}$ to the intersection of a plane and a cone

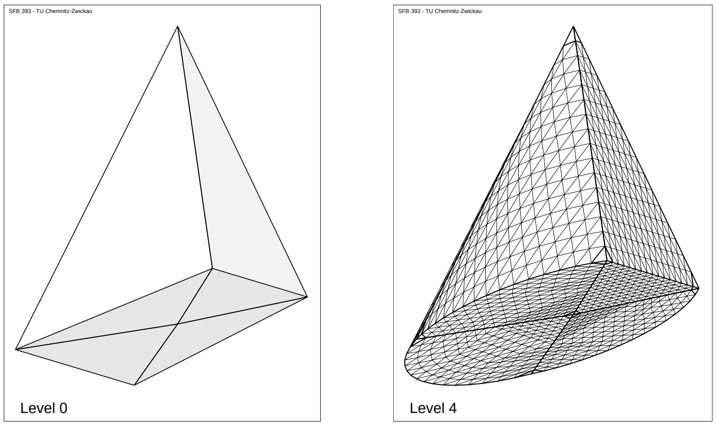


Figure 26: `mesh3/kegel4e2.std`: 4 tetrahedral elements, a cone with two cutting planes.

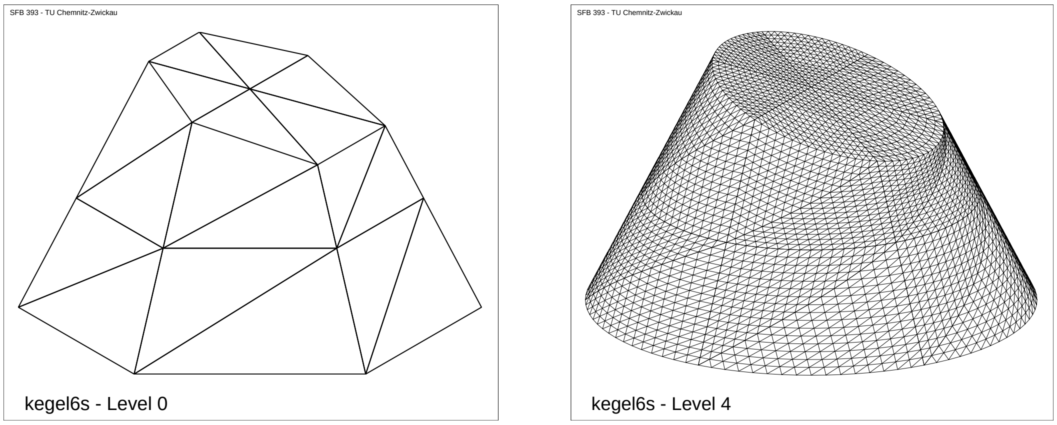


Figure 27: `mesh3/kegel6s.std`: Truncated cone, limited by a sloping plane – **the** conic section.

$$\begin{aligned}
\vec{v}_a &:= p_2 - p_1 \\
\vec{v}_1 &:= \tilde{x} - p_1 \\
\alpha &:= (\vec{v}_a \cdot \vec{v}_a) \\
\beta &:= (\vec{v}_a \cdot \vec{v}_1) \\
\vec{v}_0 &:= \vec{v}_1 - \frac{\beta}{\alpha}\vec{v}_a \\
r_0 &:= \frac{\beta}{\alpha} r_2 \\
x_h &:= \tilde{x} + \left(\frac{r_0}{|\vec{v}_0|} - 1\right)\vec{v}_0 \\
\vec{v}_2 &:= x_h - p_1 \;=\; \vec{v}_1 + \left(\frac{r_0}{|\vec{v}_0|} - 1\right)\vec{v}_0
\end{aligned}$$

if $(\vec{v}_0 \cdot \vec{n}) = 0$, then $x^* = x_h$, else:

$$x^* \;:=\; p_1 + \frac{(\vec{v}_1 \cdot \vec{n})}{(\vec{v}_2 \cdot \vec{n})}\vec{v}_2$$

Obviously the auxiliary point x_h is not needed explicitly because the vector $\vec{v}_2$ can be determined by other values.

4.2.4 Two Spheres

The intersection of two spheres $g_{21}^{(1)} = \{M_1, r_1\}, g_{21}^{(2)} = \{M_2, r_2\}$ is a circle in a plane which is orthogonal to the connection line of the two centers of the spheres. Therefore, this case leads back to the just considered case of a sphere and a plane by considering one of the spheres and the plane defined by the normal vector $\overrightarrow{M_1 M_2}$. The plane becomes unique by the valid assumption that $\tilde{x}$ belongs to it.

Figure 29 shows a construction from an intersection of four spheres with its four centers being the vertices of a regular tetrahedron.

4.2.5 Two Cylinders and Other Intersections

The point x^* at an edge of the intersection curve of two cylinders $g_{11}^{(1)} = \{p_{11}, p_{12}, r_1\}$, $g_{11}^{(2)} = \{p_{21}, p_{22}, r_2\}$ is determined by a short iteration starting with the linear approximation $\tilde{x}$. The iteration consists of an alternating projection (using 4.1.1) to both cylinders until the point belongs to each of them (i. e. no more improvement). If the two axes are orthogonal this method needs just two steps of iteration to find x^*. For any arbitrary other situation a few more steps are necessary to obtain a certain accuracy (cf. Fig. 30).

This method is generally extendible to edges at the intersection of any two faces of different geometry, if there is no special and more efficient solution available (or not yet implemented).

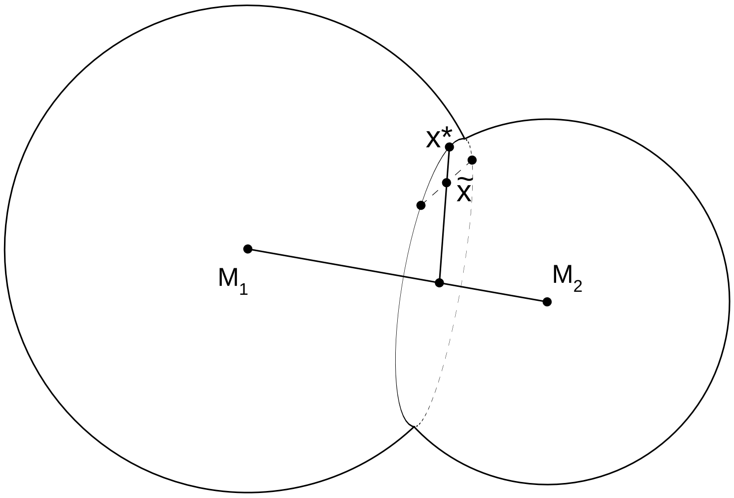


Figure 28: Intersection of two spheres

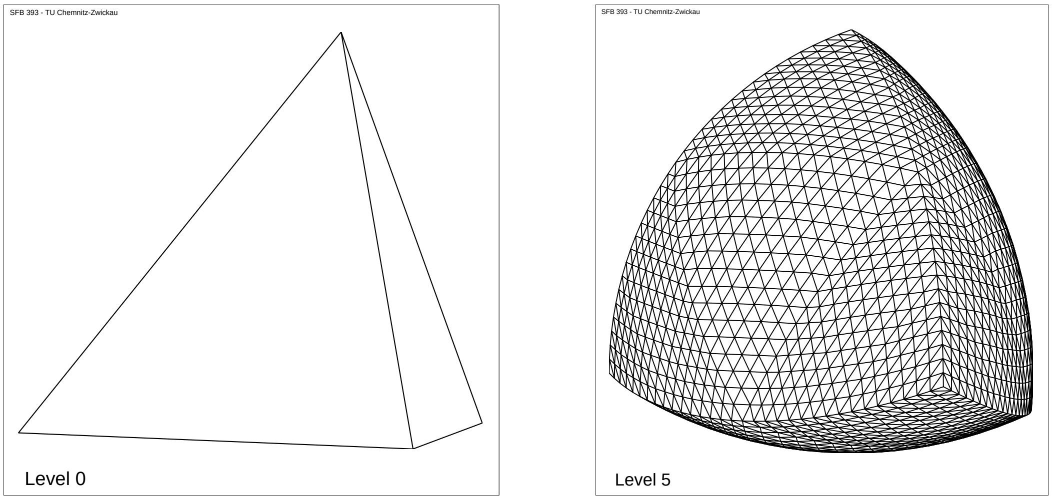


Figure 29: The vertices of the tetrahedron are center points of four spheres intersecting each other; the radius equals the tetrahedral edge length

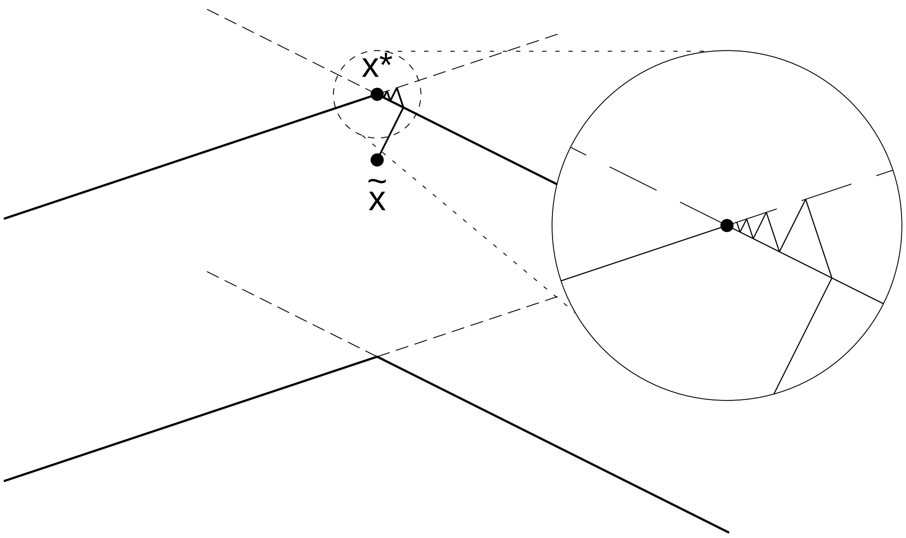
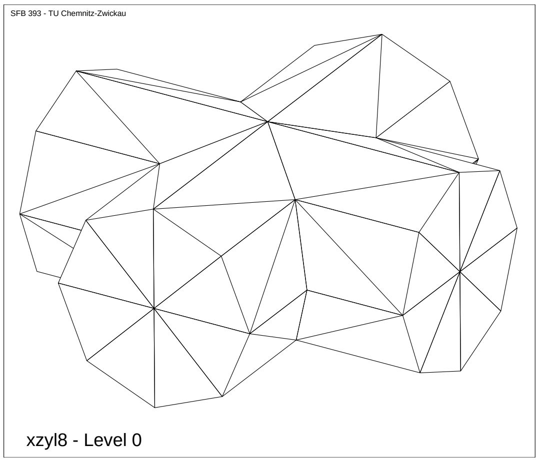


Figure 30: Iterative calculation of x^* at an intersection of two cylinders



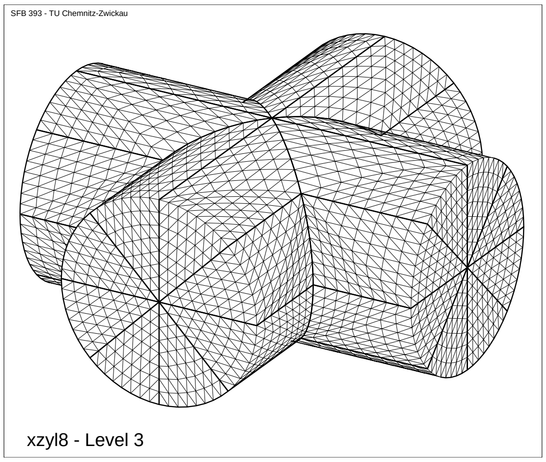


Figure 31: Durchdringung zweier Zylinder, Startnetz mit 96 Tetraedern

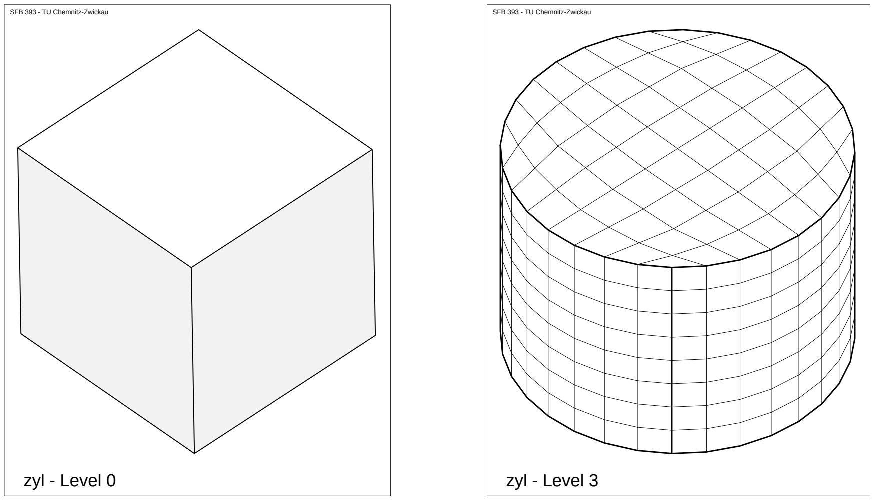


Figure 32: Unsuitable definition of a cylinder, since refinement degenerates some of the hexahedral elements (cf. Figure 10, p. 12)

4.3 Problems Defining the Coarse Grid Geometry

The examples introduced are to demonstrate how simple the algorithms may work in order to shift points to a surface given with a certain standard geometry. Since the meshes shall be used in numerical computations using the method of finite elements, they have to satisfy some additional conditions, especially those affecting the inner angles of the elements to avoid degenerated elements. This requires much care for defining an initial grid and providing face geometries. This is quite a non-trivial problem (cf. Fig. 32 and 33). For example, in Figure 32 four sides of a cuboid were declared to belong to a cylinder. In the process of mesh refinement and shifting new points to the cylinder the hexahedral elements near the original four *corners* are degenerating with angles of nearly 180 degrees. This can be avoided if the original cuboid is divided into 5 hexahedral elements as shown in Figure 10 on page 12. By inserting a face in radial direction (orthogonal to the cylindric surface) those angles cannot exceed 90 degrees in the refined mesh.

The same problem may also occur for tetrahedral meshes if a coarse grid is constructed with an angle of 90 degrees with both sides belonging to the same surface defined as any "round" geometry (as in Fig. 33 the base of a cone). Although the subdivision of tetrahedra is realized using the *stable refinement* as described by J. Bey [4] which provides only six classes of similar tetrahedra for non-curved faces, some elements near curved faces may degenerate. Analogously, it is sufficient to insert additional edges and faces in such critical regions, orthogonal to the curved face in order to limit the angles inside.

However, a cylindric bore may be defined as a cuboid in the coarse grid without having such a degeneration (Fig. 35).

We further presume that the coordinates of all nodes in the coarse grid satisfy the corresponding equations of face geometries which they are defined to belong to. The effect of non-matching points is shown in Figure 36 where the radius of a cylinder does not match the distance of the initially defined coarse grid points from the axis. Only the new points are shifted to the cylinder. A similar (but not equivalent in each case)

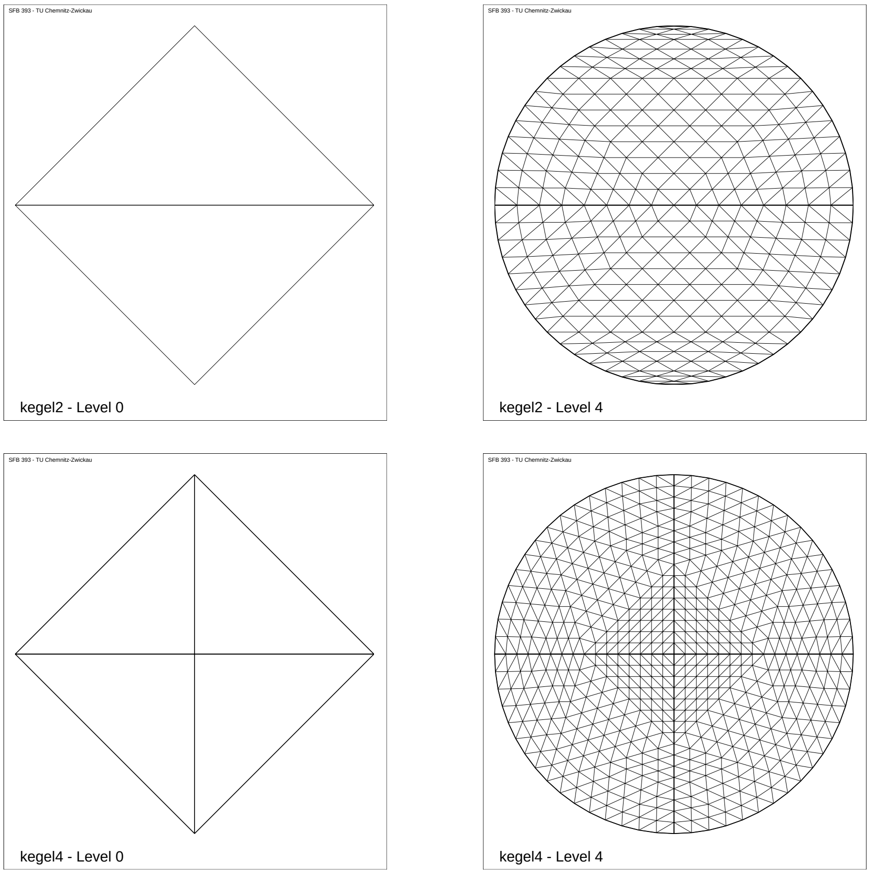


Figure 33: Degenerating tetrahedral elements may occur, if the coarse grid does not sufficiently take into account the geometry of faces; here illustrated viewing at the base of a cone; above: cone with two original tetrahedra, below: cone with four original tetrahedra

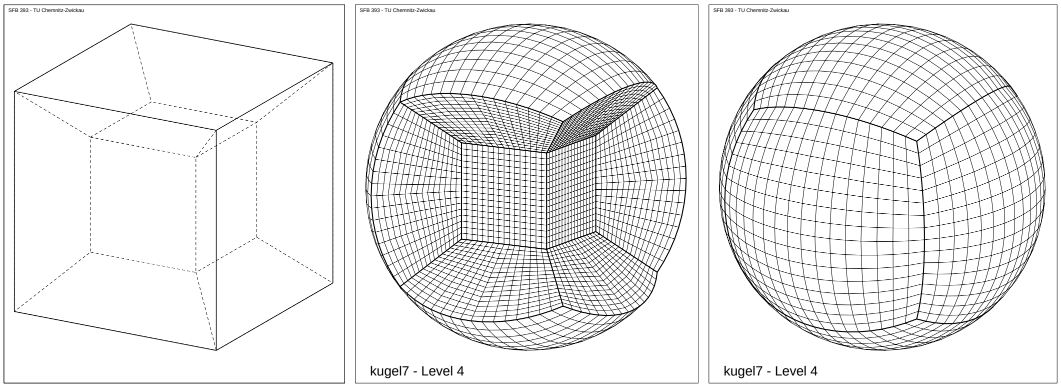


Figure 34: A cube subdivided into 7 hexahedral elements may also be refined to an acceptable sphere. The picture in the middle shows the structure of the refined mesh inside.

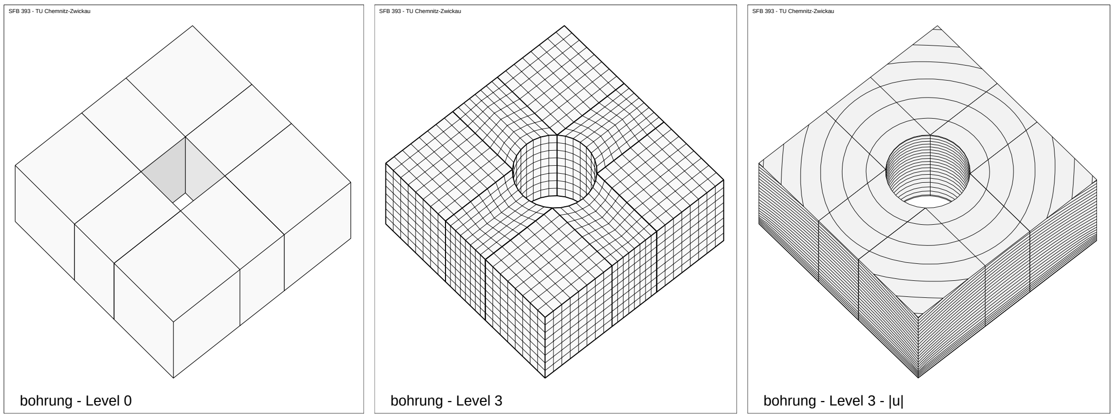


Figure 35: Cylindric bore defined by a cuboid in the coarse grid, right: an isoline representation for an elastic deformation

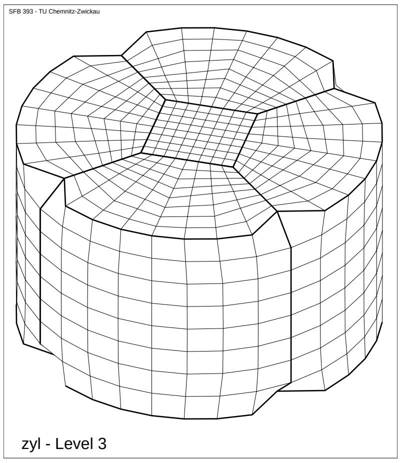


Figure 36: The effect of an error in the geometry definition (the radius was specified too large)

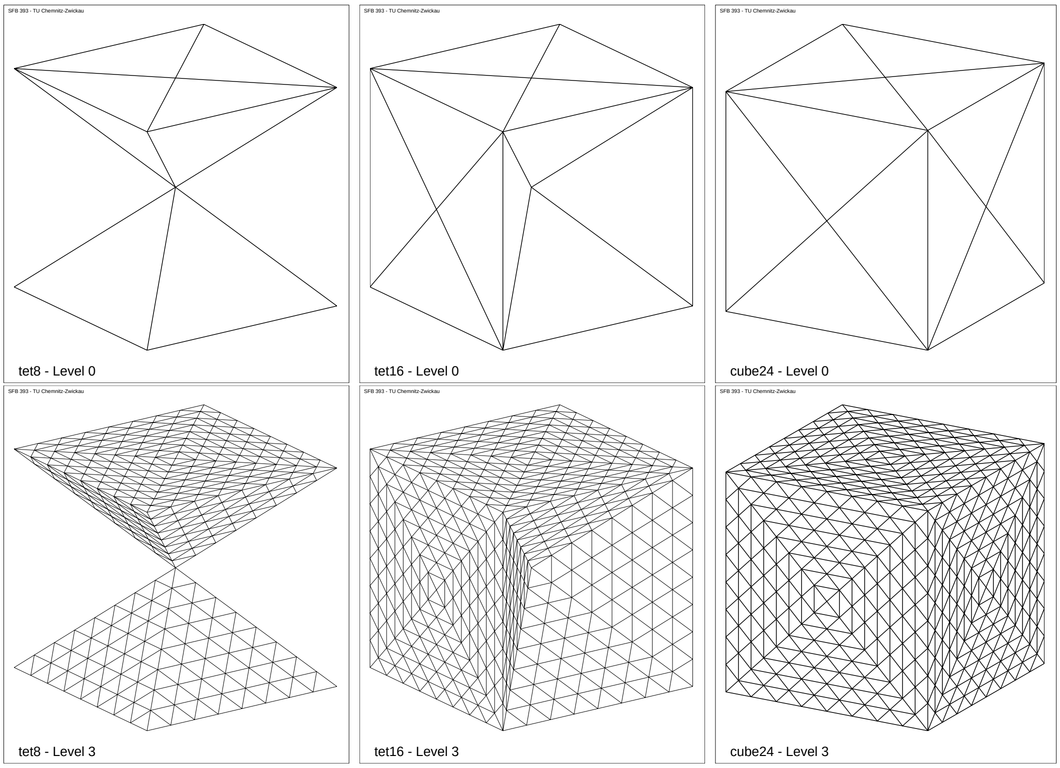


Figure 37: Three intermediate stages generating a mesh that may be defined as a cylinder or a sphere läßt

correction of the coordinates could be applied to coarse grid points as for the points at the mesh refining procedure. The difference is that we could not assume the point to be on a line between two points of the given curved face which could be assumed for the points $\tilde{x}$ on subdivided edges or faces.

Just the construction of more complex domains with different surface geometries intersecting each other mostly requires some effort. The figures demonstrate that some helpful tools should be used to generate such domains from basic components. So for example a program `geo_conv` by D. Lohse for simple geometrical transformations including a merge option for multiple input data files to obtain such complex domains in a constructive way.

Fig. 37 illustrates such a process of constructing a cube that consists of 24 congruent tetrahedra (the mesh is also shown with a 3-level refinement). The first picture is the result of the reflection of a 4-tetrahedra pyramid (cf. Fig. 14) at the x-y-plane. The same double pyramid is twice merged together with the original one, once after a rotation by 90 degrees round the x-axis and once more after a rotation round the y-axis. Now, the surface of this cube can be defined to become a sphere or a cylinder if the center points of all 6 or of 4 sides are shifted to $\sqrt{2}$- or $\sqrt{3}$-times the original distance, respectively, from the center of the cube (Figs. 38, 39).

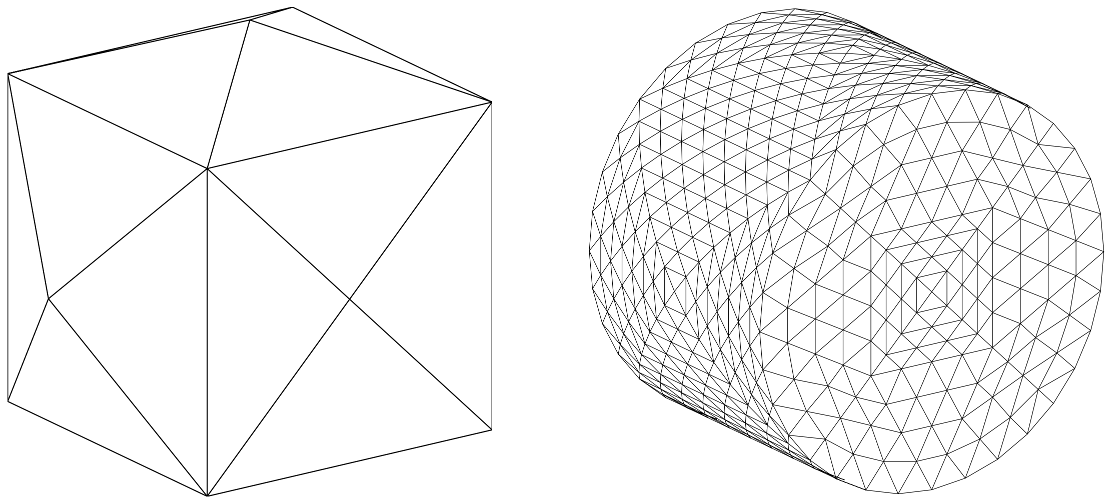


Figure 38: The cube of Fig. 37 becomes a cylinder ...

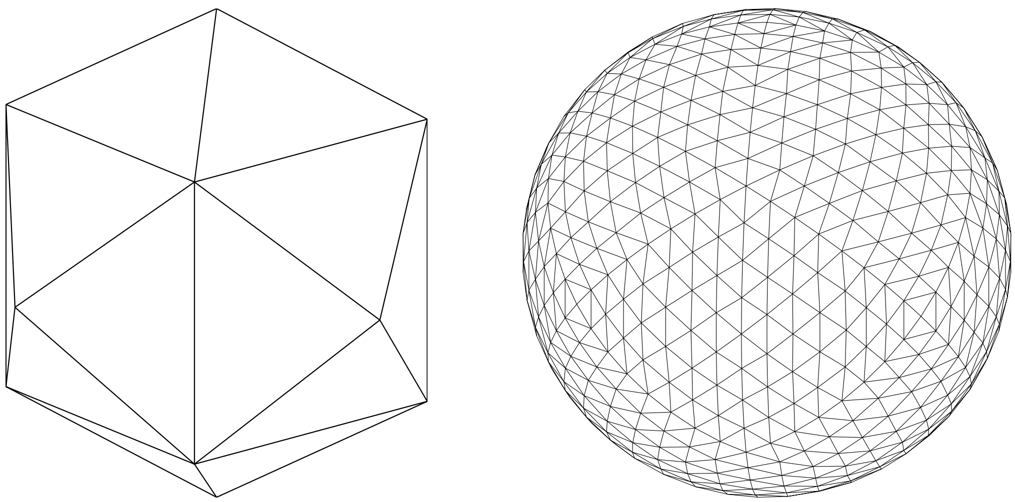


Figure 39: ... or a sphere

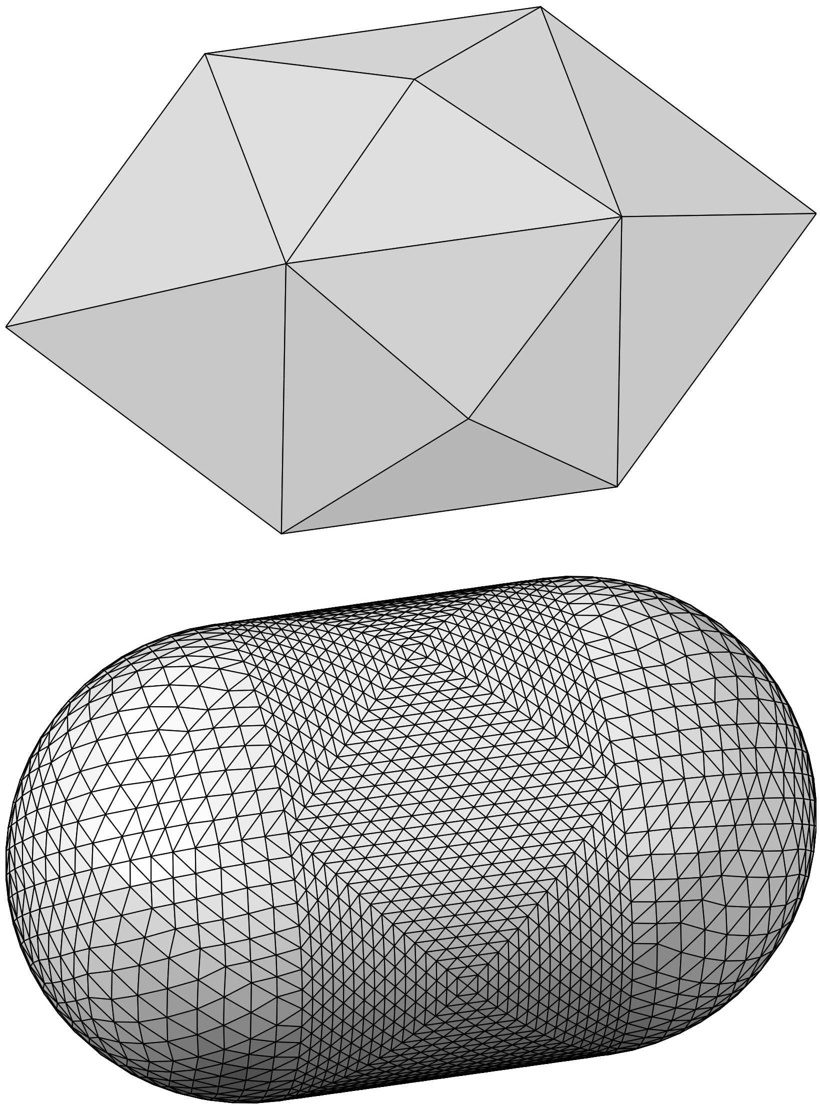


Figure 40: Adding (2×4) tetrahedral elements to the cylinder it may become such a *“capsule”*.

4.4 More Examples With Different Geometries

The following pictures give some more examples of curved surfaces that can be defined by a relatively coarse initial grid. After a few levels of mesh refinement the curved surfaces are well approximated.

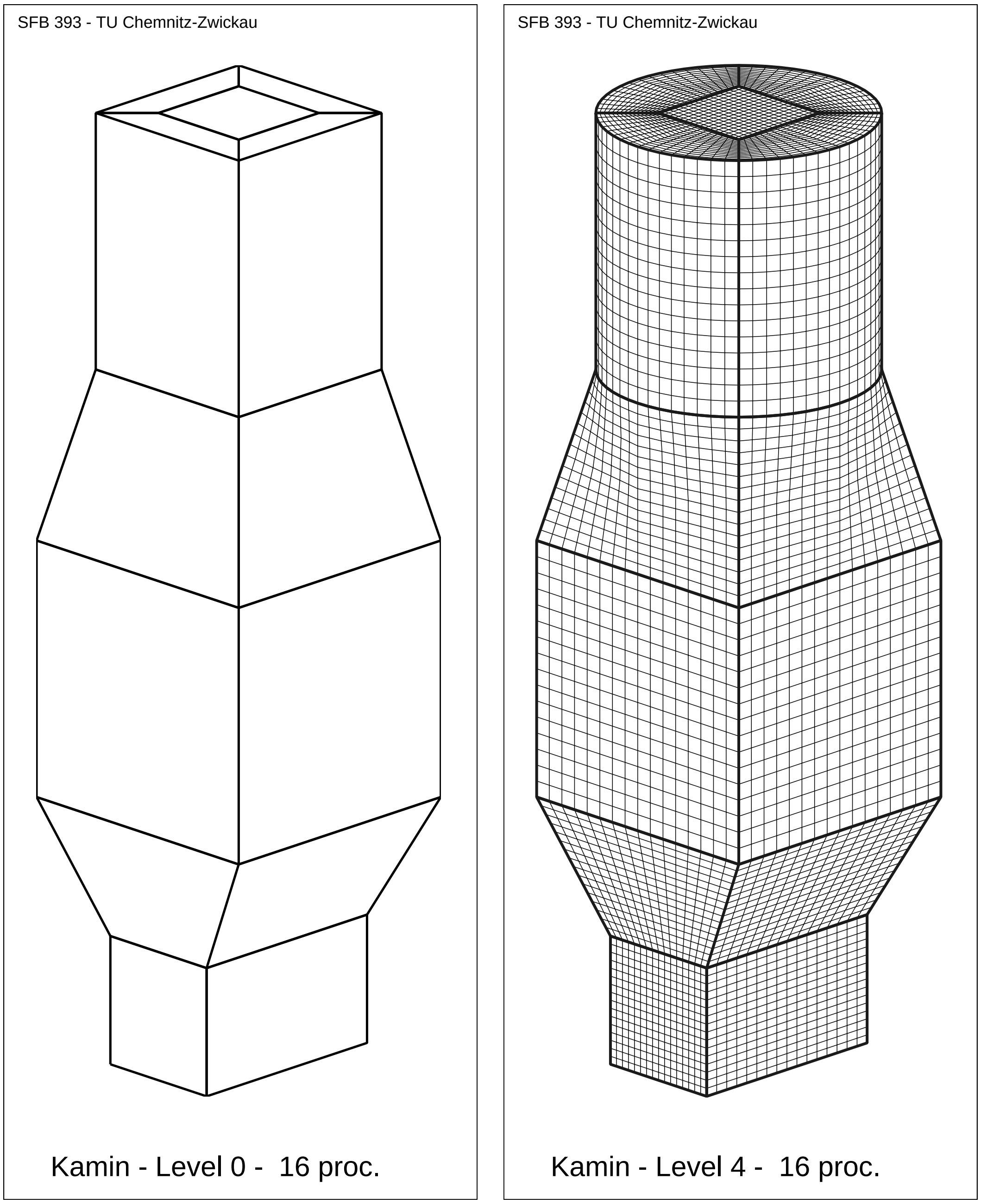


Figure 41: A chimney with changing forms at the cross-section.
Initiasl mesh: 25 hexahedral elements

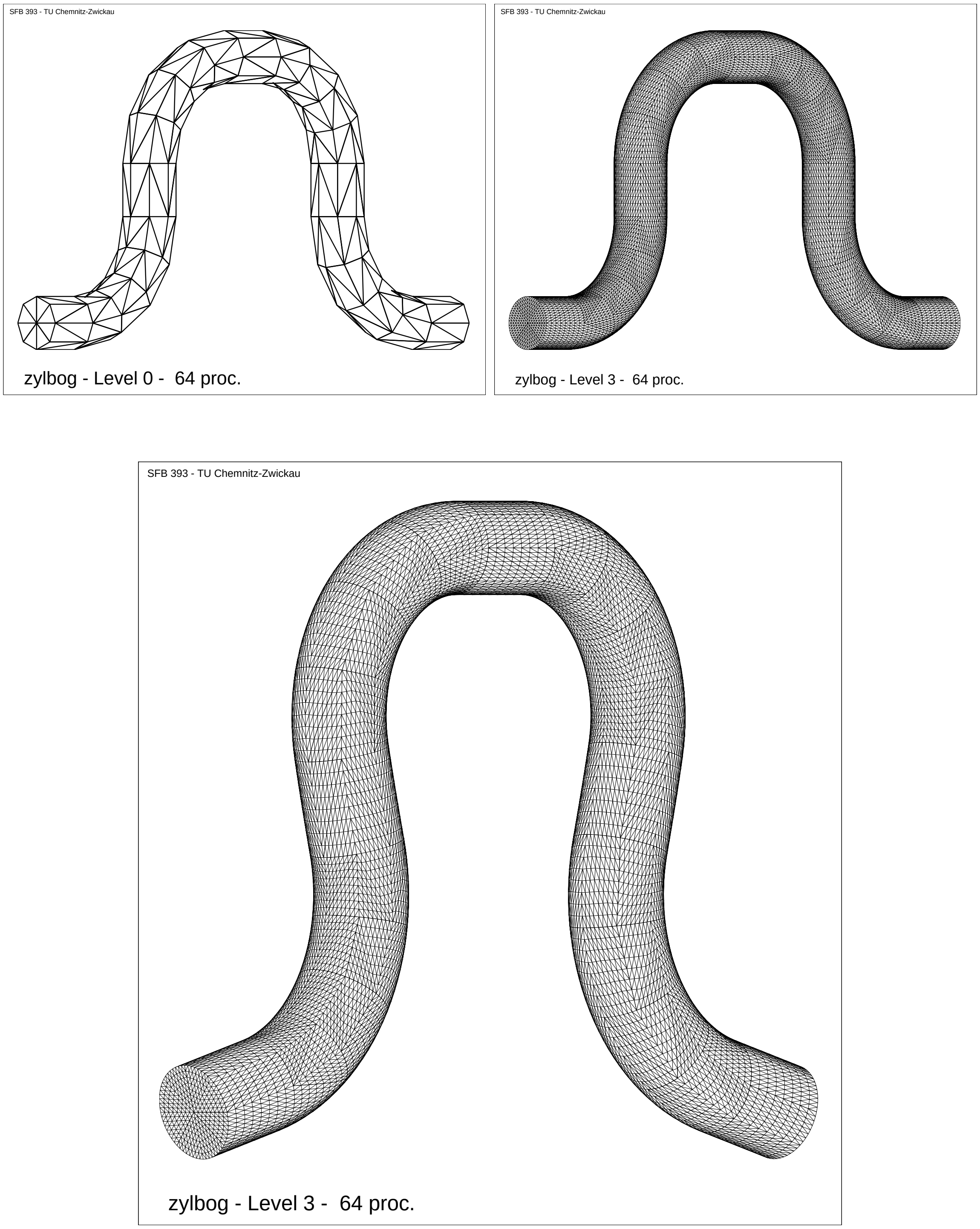


Figure 42: A "pipeline" composed of cylinder and torus elements. Initial mesh: 504 Tetrahedral elements,
below: solution of an elastic deformation problem

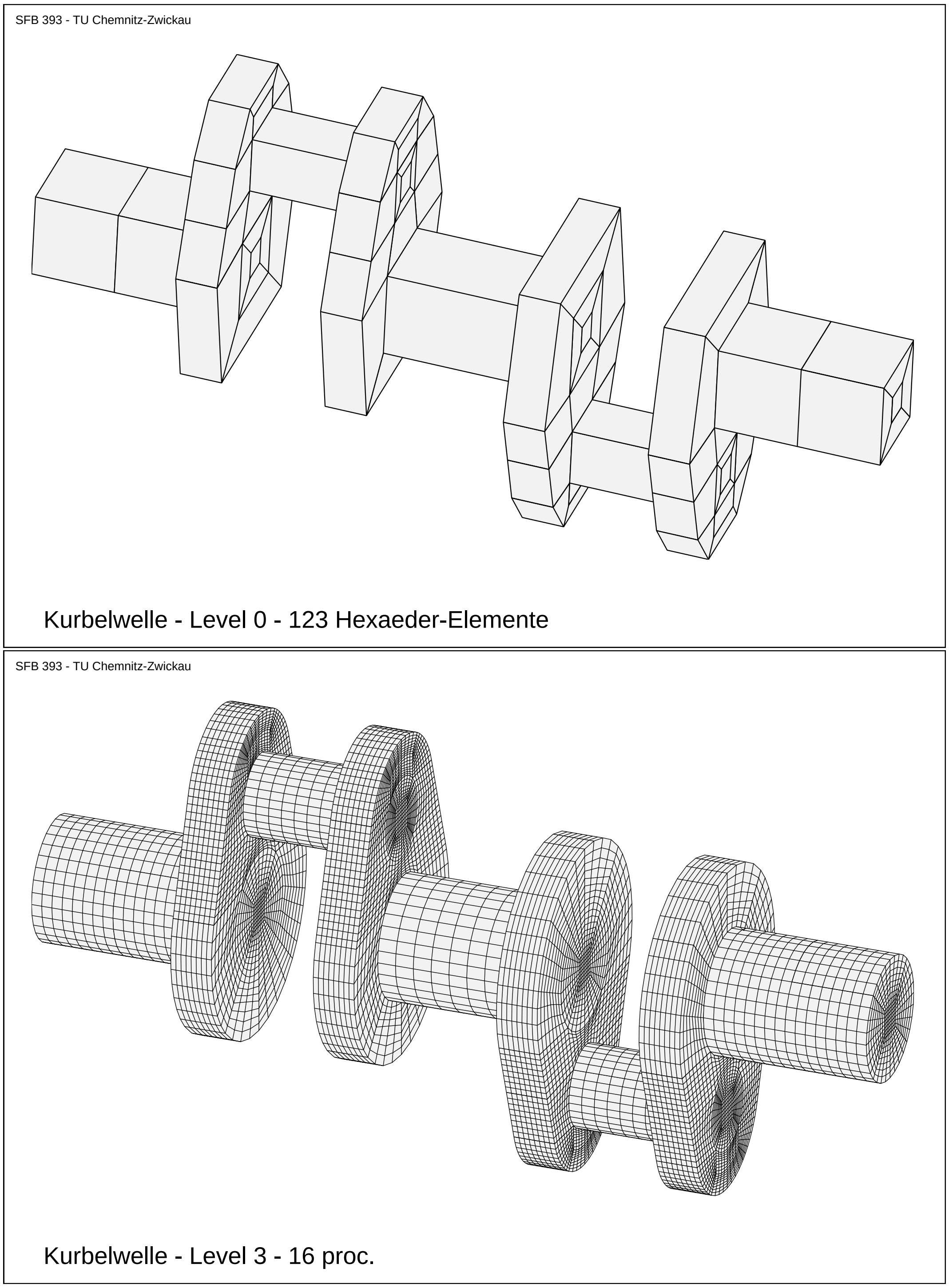


Figure 43: Crankshaft, bounded by plane and cylindric faces,
Initial mesh: 123 hexahedral elements

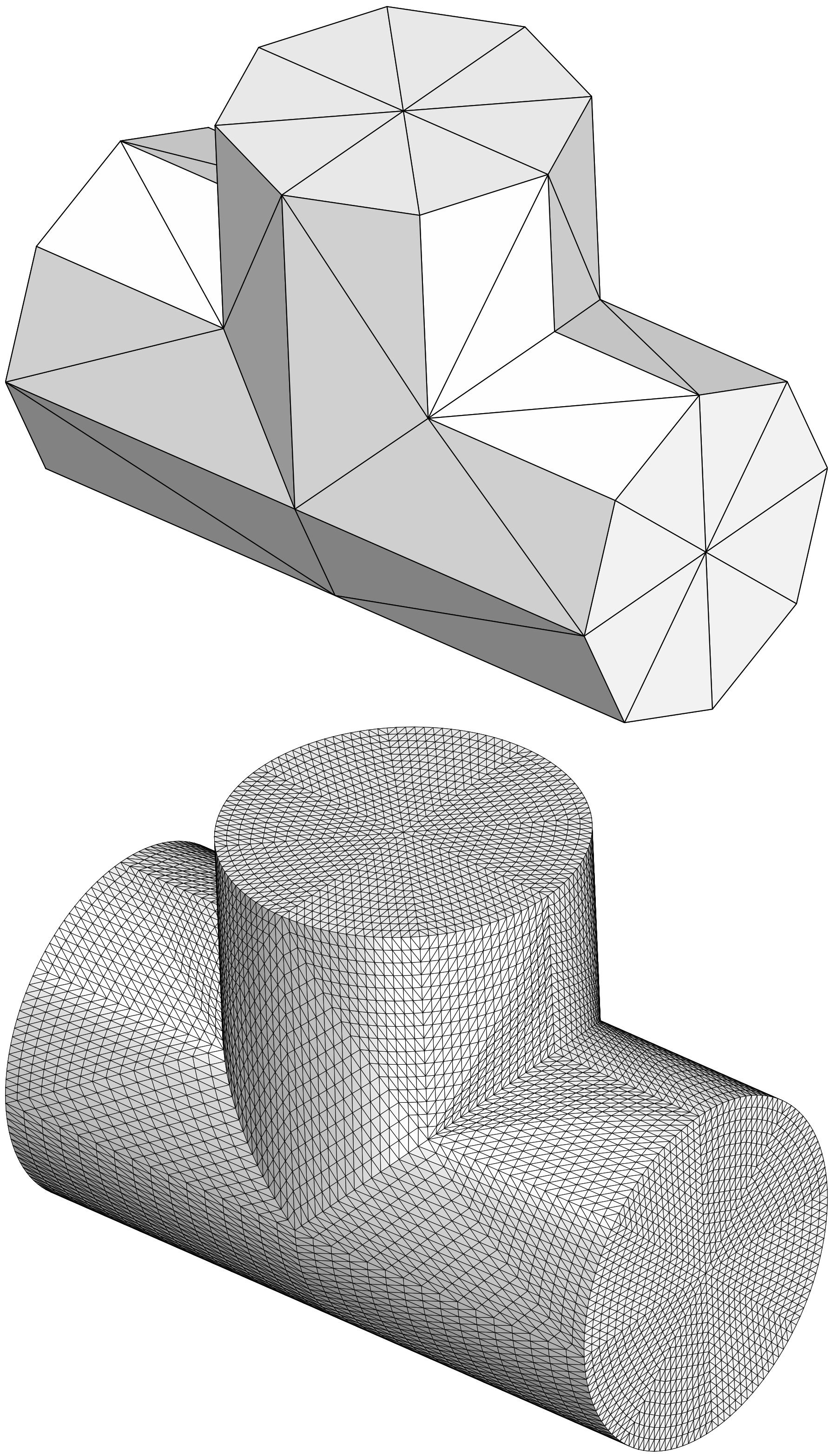


Figure 44: T-shape of two cylinders, initial mesh: 72 tetrahedral elements

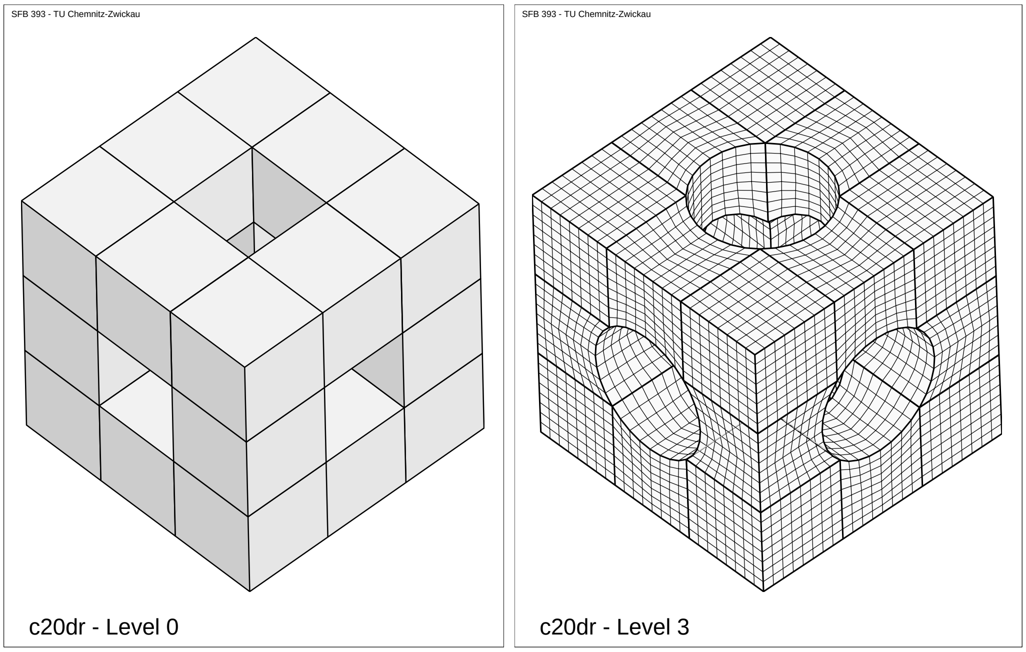


Figure 45: Three cylindric bores intersecting each other

5 Conclusion

The object of this paper was to introduce an easy-to-use method for describing and handling geometrical properties of surfaces in three-dimensional Finite Element computations. The geometry should be defined with only a few parameters, and the initial grid should be as coarse as possible. The real mesh matching the curved surfaces is generated afterwards by mesh refining which can be executed locally (for subdomains) on the processors of the parallel computer where the further numerical computation takes place using the refined mesh. This reduces the size of input data files and the effort for distributing this initial data to the processors. Nevertheless, such a coarse grid includes an exact representation of any geometry which can be approximated with any precision by an Finite Element mesh, depending on the total memory capacity of the parallel computer.

The method described is very suitable for a constructive definition of domains. It is open for extensions. For each new face geometry that can be declared by a certain set of parameters the user has to supply an additional implementation of the function $x^* := F(\tilde{x}, g)$ to shift the approximation $\tilde{x}$ into the correct position x^* in the curved face. For special applications it is quite possible to use more complicated surfaces which cannot be described by a few fixed parameters but can be “realized” by special software.

Edges at the intersection of curved faces can be considered as a special case or be treated in a general way using the iterative method as described in 4.2.5.

The examples considered in this paper are generally such domains that were refined homogenously. However, the treatment of curved faces is completely independet of the refinement strategy, because only the current approximating point and the given geometry are needed for that process.

References

[1] T. Apel. SPC-PM Po 3D — User's manual. Preprint SPC 95_33, TU Chemnitz-Zwickau, December 1995.

[2] T. Apel, G. Haase, A. Meyer, and M. Pester. Parallel solution of finite element equation systems: efficient inter-processor communication. Preprint SPC 95_5, TU Chemnitz-Zwickau, Februar 1995.

[3] T. Apel, F. Milde, and M. Theß. SPC-PM Po 3D — Programmer's manual. Preprint SPC 95_34, TU Chemnitz-Zwickau, December 1995.

[4] J. Bey. Der BPX-Vorkonditionierer in 3 Dimensionen: Gitter-Verfeinerung, Parallelisierung und Simulation. Preprint 92 - 03, IWR Universität Heidelberg, 1992.

[5] P. Ciarlet. *The finite element method for elliptic problems.* North–Holland, Amsterdam, 1978.

[6] G. Globisch. PARMESH - a parallel mesh generator. Preprint SPC 93_3, TU Chemnitz-Zwickau, June 1993. Parallel Computing 21, No. 3, March 1995, pp. 509-524.

[7] G. Globisch. On an automatically parallel generation technique for tetrahedral meshes. Preprint SPC 94_6, TU Chemnitz-Zwickau, April 1994.

[8] T. Grund and K. Pietsch. Gitterverfeinerung krummlinig begrenzter Körper. Programmierpraktikum, TU Chemnitz-Zwickau, 1997.

[9] F. Kickinger. Automatic mesh generation for 3D objects. Technical Report No 96-1, Johannes Kepler Universität Linz, Institut f r Mathematik, February 1996.

[10] A. Meyer. A parallel preconditioned conjugate gradient method using domain decomposition and inexact solvers on each subdomain. *Computing*, 45 : 217–234, 1990.

[11] M. Pester. Grafik-Ausgabe vom Parallelrechner für 2D-Gebiete. Preprint SPC 94_24, TU Chemnitz-Zwickau, November 1994.

[12] M. Pester. On-line visualization in parallel computations. In W. Borchers, G. Domick, D. Kröner, R. Rautmann, and D. Saupe, editors, *Visualization Methods in High Performance Computing and Flow Simulation*, pages 91–98, Utrecht, The Netherlands, 1996. VSP. Preprint: SPC 94_23 (Nov. 1994).

[13] U. Reichel. Partitionierung von Finite-Elemente-Netzen. Preprint SFB393/96-18, TU Chemnitz-Zwickau, November 1996.

[14] A. Wierse and M. Rumpf. GRAPE Eine objektorientierte Visualisierungs- und Numerikplattform. *Informatik Forsch. Entw.*, 7:145–151, 1992.

[15] O. C. Zienkiewicz. *Die Methode der finiten Elemente.* Fachbuchverlag, Leipzig, 1975.

Other titles in the SFB393 series:

96-01 V. Mehrmann, H. Xu. Chosing poles so that the single-input pole placement problem is well-conditioned. Januar 1996.

96-02 T. Penzl. Numerical solution of generalized Lyapunov equations. January 1996.

96-03 M. Scherzer, A. Meyer. Zur Berechnung von Spannungs- und Deformationsfeldern an Interface-Ecken im nichtlinearen Deformationsbereich auf Parallelrechnern. March 1996.

96-04 Th. Frank, E. Wassen. Parallel solution algorithms for Lagrangian simulation of disperse multiphase flows. Proc. of 2nd Int. Symposium on Numerical Methods for Multiphase Flows, ASME Fluids Engineering Division Summer Meeting, July 7-11, 1996, San Diego, CA, USA. June 1996.

96-05 P. Benner, V. Mehrmann, H. Xu. A numerically stable, structure preserving method for computing the eigenvalues of real Hamiltonian or symplectic pencils. April 1996.

96-06 P. Benner, R. Byers, E. Barth. HAMEV and SQRED: Fortran 77 Subroutines for Computing the Eigenvalues of Hamiltonian Matrices Using Van Loans's Square Reduced Method. May 1996.

96-07 W. Rehm (Ed.). Portierbare numerische Simulation auf parallelen Architekturen. April 1996.

96-08 J. Weickert. Navier-Stokes equations as a differential-algebraic system. August 1996.

96-09 R. Byers, C. He, V. Mehrmann. Where is the nearest non-regular pencil? August 1996.

96-10 Th. Apel. A note on anisotropic interpolation error estimates for isoparametric quadrilateral finite elements. November 1996.

96-11 Th. Apel, G. Lube. Anisotropic mesh refinement for singularly perturbed reaction diffusion problems. November 1996.

96-12 B. Heise, M. Jung. Scalability, efficiency, and robustness of parallel multilevel solvers for nonlinear equations. September 1996.

96-13 F. Milde, R. A. Römer, M. Schreiber. Multifractal analysis of the metal-insulator transition in anisotropic systems. October 1996.

96-14 R. Schneider, P. L. Levin, M. Spasojević. Multiscale compression of BEM equations for electrostatic systems. October 1996.

96-15 M. Spasojević, R. Schneider, P. L. Levin. On the creation of sparse Boundary Element matrices for two dimensional electrostatics problems using the orthogonal Haar wavelet. October 1996.

96-16 S. Dahlke, W. Dahmen, R. Hochmuth, R. Schneider. Stable multiscale bases and local error estimation for elliptic problems. October 1996.

96-17 B. H. Kleemann, A. Rathsfeld, R. Schneider. Multiscale methods for Boundary Integral Equations and their application to boundary value problems in scattering theory and geodesy. October 1996.

96-18 U. Reichel. Partitionierung von Finite-Elemente-Netzen. November 1996.

96-19 W. Dahmen, R. Schneider. Composite wavelet bases for operator equations. November 1996.

96-20 R. A. Römer, M. Schreiber. No enhancement of the localization length for two interacting particles in a random potential. December 1996. to appear in: Phys. Rev. Lett., March 1997

96-21 G. Windisch. Two-point boundary value problems with piecewise constant coefficients: weak solution and exact discretization. December 1996.

96-22 M. Jung, S. V. Nepomnyaschikh. Variable preconditioning procedures for elliptic problems. December 1996.

97-01 P. Benner, V. Mehrmann, H. Xu. A new method for computing the stable invariant subspace of a real Hamiltonian matrix or Breaking Van Loan's curse? January 1997.

97-02 B. Benhammouda. Rank-revealing 'top-down' ULV factorizations. January 1997.

97-03 U. Schrader. Convergence of Asynchronous Jacobi-Newton-Iterations. January 1997.

97-04 U.-J. Görke, R. Kreißig. Einflußfaktoren bei der Identifikation von Materialparametern elastisch-plastischer Deformationsgesetze aus inhomogenen Verschiebungsfeldern. March 1997.

97-05 U. Groh. FEM auf irregulären hierarchischen Dreiecksnetzen. March 1997.

97-06 Th. Apel. Interpolation of non-smooth functions on anisotropic finite element meshes. March 1997

97-07 Th. Apel, S. Nicaise. The finite element method with anisotropic mesh grading for elliptic problems in domains with corners and edges.

97-08 L. Grabowsky, Th. Ermer, J. Werner. Nutzung von MPI für parallele FEM-Systeme. March 1997.

97-09 T. Wappler, Th. Vojta, M. Schreiber. Monte-Carlo simulations of the dynamical behavior of the Coulomb glass. March 1997.

The complete list of current and former preprints is available via
`http://www.tu-chemnitz.de/sfb393/sfb97pr.html`.