Triangulation of 3D Domains
(with Quad-Hexa Meshing Support)

User Guide

November 27, 2016

Daniel Rypl

Czech Technical University in Prague
Faculty of Civil Engineering, Department of Mechanics

Thákurova 7, 166 29, Prague
Czech Republic

e-mail: drypl@fsv.cvut.cz
http://mech.fsv.cvut.cz/~dr/dr.html

Contents

1 Model Representation

The model is described by a boundary representation and consists of the following model entities: **vertices**, **curves**, **surfaces**, **patches**, **shells** and **regions**. Topologically, each region is formed by a set of not self-intersecting boundary surfaces, patches, and shells, each of which is bounded by a set of curves. Each curve is given by two end vertices. Moreover, each boundary surface, patch, and shell points out to the regions on the side of its outer and inner normal. A curve keeps list of surfaces, patches, and shells sharing that curve. Similarly, a vertex stores the list of curves sharing that vertex. This basic topology is further restricted by the geometry of model entities. Both curves and surfaces are based on free-form representation in terms of tensor-product polynomial entities. This limits the number of curves bounding a surface to four. The number of curves bounding a patch or a shell is not limited (but must be at least two). While the patch is a planar entity (trimmed plane) or a nonplanar entity close to a plane, the shell is constrained to a background surface (trimmed surface). To enhance the modelling capability an entity-to-entity fixation concept has been introduced. Generally, each model entity may be fixed to another model entity of the same or higher dimension. However, the fixed entity is not allowed to coincide with the boundary of the parent entity. Each model entity keeps the list of entities fixed to it. No further topological information is required for the description of a valid non-manifold domain of almost arbitrary complexity.

The model also offers representation of **interfaces** between pairs of geometrically and topologically identical model entities using the model interface entities. Interface entity is bounded by two model entities (except regions) of the same type. Geometry of interface entity is defined as the space between the pairs of bounding model entities. Note that there are no restrictions on this space.

Currently, rational Bezier entities are employed for free-form curves and surfaces representation. This allows to represent exactly conics and quadrics by entities starting with an order of three (quadratic curves and biquadratic surfaces).

The rational Bezier curve has the form

$$\boldsymbol{P}(t) = \frac{\sum_{i=0}^{n} \omega_i \boldsymbol{P}_i B_i^n(t)}{\sum_{i=0}^{n} \omega_i B_i^n(t)}, \tag{1}$$

where $\boldsymbol{P}(t)$ is the point on the curve, $\boldsymbol{P}_i$ are Bezier control points, ω_i are weights of Bezier control points, $B_i^n(t)$ stand for Bernstein polynomials, t denotes an independent variable varying in range from 0 to 1, and n is the curve degree. The curve order is equal to $n + 1$. $\boldsymbol{P}_0$ and $\boldsymbol{P}_n$ correspond to model vertices while the remaining points form the control polygon of the curve. They determine the bow of the curve and need not generally lie on the curve. The first and last segments of the control polygon coincide with the curve tangent in the starting and ending vertices respectively.

The rational Bezier surface can be written in a similar form

$$\boldsymbol{P}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m}\omega_{ij}\boldsymbol{P}_{ij}B_i^n(u)B_j^m(v)}{\sum_{i=0}^{n}\sum_{j=0}^{m}\omega_{ij}B_i^n(u)B_j^m(v)}, \tag{2}$$

where $\boldsymbol{P}(u,v)$ is the point on the surface, $\boldsymbol{P}_{ij}$ are Bezier control points, ω_{ij} are weights of Bezier control points, $B_i^n(u)$ and $B_j^m(v)$ stand for Bernstein polynomials, u and v denote independent parameters varying in range from 0 to 1, and n and m are surface degrees (orders are equal to $n+1$ and $m+1$) in u and v parametric directions, respectively. If the control points are arranged in a matrix $(n+1)\times(m+1)$ then the corner points correspond to model vertices, the side points correspond to control polygons of model curves bounding the surface, and the remaining points form the control polygon of the surface and need not generally lie on the surface.

Bernstein polynomial can be expressed as

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \tag{3}$$

or recursively as

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t), \tag{4}$$

where $B_0^0 = 1$.

The ordinary Bezier entities can be derived from rational Bezier entities when all weights are set to 1.

Two types of model entities are distinguished. The **physical** ones which are designated for the actual discretization and the **virtual** ones which serve as auxiliary for geometry description or mesh size specification. Note that there are some restrictions on the fixation between virtual and physical entities.

2 Mesh Size Specification

Three levels of mesh size specification are considered

- the global mesh size specification,
- the local mesh size specification, and
- the adaptive mesh size specification.

The global mesh size specification uses global weight functions to control the mesh size description over the domain. The local mesh size specification concept prescribes the desired mesh size variation on model entities. In the adaptive mesh size specification strategy, various mesh size sources, built according to the preceding problem analysis, are used.

Currently only the local and adaptive mesh size specifications are implemented. The local mesh size specification consists of three concepts

- the required mesh size specification,
- the curve count-based specification, and
- the curvature-based mesh size control.

The first concept is used to explicitly prescribe the mesh size (using keyword ***size***) at individual model entities. Mesh size specification is stored at each vertex and at each control point of any curve or surface. These values are used to extract the mesh size specification on a curve or surface. Moreover, each model entity (except vertices) stores an upper bound limit on the mesh size which is not allowed to be exceeded.

Similar expressions to the ones describing the geometry of rational Bezier curves and surfaces are used to interpolate the local mesh size specification at control points over the curve and surface. The mesh size extracted from a curve mesh size specification has the form

$$msz(t) = \frac{\sum_{i=0}^{n} \omega_i msz_i B_i^n(t)}{\sum_{i=0}^{n} \omega_i B_i^n(t)}, \tag{5}$$

where $msz(t)$ is the required mesh size at point $\boldsymbol{P}(t)$ on the curve, msz_i are the mesh size specifications at Bezier control points, and the other symbols have the same meaning as in Eq. (1). A similar formula can be written for the extraction of the required mesh size on a surface

$$msz(u,v) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} \omega_{ij} msz_{ij} B_i^n(u) B_j^m(v)}{\sum_{i=0}^{n} \sum_{j=0}^{m} \omega_{ij} B_i^n(u) B_j^m(v)}, \tag{6}$$

where $msz(u,v)$ is the required mesh size at point $\boldsymbol{P}(u,v)$ on the surface, msz_{ij} are the mesh size specifications at Bezier control points. The remaining variables have the the same meaning as in Eq. (2).

In the second concept, the upper bound limit on the mesh size along a curve is defined via the desired number of segments on the curve (using keywords ***density***, ***nosplit*** and ***count***).

Finally, the last concept is employed to enable an accurate representation of a curve or surface by its discretization even if no particular mesh size is required. The criterion is based on the ratio (curvature rate) between the appropriate mesh size and the radius of curvature at a given location on the curve or surface. The default ratio is equal to 1, which corresponds to the discretization of a circle to 6 segments of the same length equal to the radius of the circle. The curvature-based mesh size control is performed implicitly. Its suppression may result in an unrecoverable error during mesh generation.

The adaptive mesh size specification is based on a background mesh with the mesh size specification at nodes and linear interpolation over the elements (edges, triangles, quadrilaterals, tetrahedra, pyramids, wedges, and hexahedra). The background mesh is independent of the actual model and may cover the whole model or only some of its parts. The format of the background mesh is described in Section 6.

3 Model Input Data Format

The input data consist of a set of keywords and appropriate numeral or literal values associated with them. The first keyword on the line is specific and is obligatory. The order of following keywords is more or less compulsory but must respect some built-in logic. A unique positive identification number has to be assigned to each model entity. This number is then used to reference this model entity. Note that the numbering of model entities of different type is independent. The individual model entities must be ordered in the input file in such a way that any referenced model entity must already exist. Keywords may be typed in upper case, lower case, or mixed case. Everything on a line behind a # sign is treated as a comment. Any number of blank spaces may be used between the keywords and numbers. Empty lines are ignored. Too long lines may be split using the backslash. Note that each part of the splitted commented line must be marked by the # sign to comment it thoroughly. No further input data formating is required.

Note that some features are enabled or disabled if the source code is compiled with certain directives. The following table gives a list of compilation directives and associated input file keywords:

Compiler directive	**Keyword(s)**	**in input record of**
T3D_INTERFACE	all	interface
T3D_COINCIDE	*coincide*	vertex, curve, surface, patch, and shell
T3D_CURVE_SWAPPING	*swap*, *angle*	curve
T3D_CURVE_SMEARING	*smear*, *angle*	curve
T3D_MIRRORING	*mirror*, *weak*, *simple*, *octree*	curve, surface, patch, and shell
T3D_PROPERTY	*nodeprop*	node, curve, surface, patch, shell, and region
T3D_PROPERTY	*elemprop*	curve, surface, patch, shell, and region
T3D_PROPERTY	*elemgroup*	curve, surface, patch, shell, and region
T3D_QUAD_HEXA_MESHING	*quad*	surface, patch and shell
T3D_QUAD_HEXA_MESHING	*hexa*	region
T3D_QUAD_HEXA_MAPPING	*map*, *lcs*	surface, patch, shell, and region
T3D_QUAD_HEXA_MAPPING	*duplicate*	curve, surface, patch and shell
T3D_QUAD_HEXA_MAPPING	*transition*	curve
T3D_QUAD_HEXA_MAPPING	*diagonal*	surface, patch, and shell

Compiler directive	**Keyword(s)**	**in input record of**
T3D_QUAD_HEXA_ISO	*iso*, *dir*	patch and region
T3D_QUAD_HEXA_ISO	*diaginal*	patch
T3D_EXTRUDE	*extrude*, *weak*, *simple*, *octree*	region
T3D_NONPLANAR_PATCH	*epsilon*	patch
T3D_MASTER_SLAVE	*master*	vertex, curve, surface, patch and shell
	slave	
	nomaster	
	noslave	
T3D_MASTER_SURFACE	*master*	surface, patch and shell
T3D_BND_MESH	all	bnd_mesh
T3D_BND_MESH	all	bnd_vertex, bnd_curve, and bnd_surface
T3D_QUAD_HEXA_SUPPORT	*bnd_quads*, *bnd_quad*, *normal_quad*	bnd_surface

Note that compiler directives *T3D_INTERFACE*, *T3D_QUAD_HEXA_MESHING*, *T3D_QUAD_HEXA_MAPPING*, *T3D_QUAD_HEXA_ISO* and *T3D_EXTRUDE* are ignored if directive *T3D_QUAD_HEXA_SUPPORT* is not defined. Envoke T3d with *–dir* command line option to print the overview of compilation directives.

Note that quadratic bubble elements are disabled unless the code is compiled with *T3D_BUBBLE_ELEMENT* directive.

The following notation will be used in the description of input records for individual model entities:

{ } - obligatory parameter	| - logical XOR
[ ] - optional parameter	|| - logical OR
( ) - repeated parameter	\ - line break
$_{\text{in}}$# - integer number	@ - single quoted string
$_{\text{fp}}$# - floating point number	

Note that there exists a limit on the length of the input file records, unless compiler directive T3D_FUNCTION_INPUT is used. Split the too long lines in the input file using the line break character \. Check the input file line length limit by envoking T3d with *–dir* command line option before processing input file with very long records.

3.1 Input Record of a VERTEX

```
Vertex in# { xyz fp# fp# fp#  || ( fixed { vertex in#  |                              \
                                            curve in#  [ t fp# ]  |                   \
                                            surface in#  [ uv fp# fp# ] } ) }         \
          [ size { fp#  | def  | uni  | ver  | cur  | sur }  [ * fp# ] ]             \
          [ weight fp# ]                                                              \
          [ factor fp# ]                                                              \
          [ property in# ]                                                            \
          [ virtual ]                                                                 \
          [ hidden ]                                                                  \
          [ output { yes  | no } ]                                                    \
          [ coincide vertex ( in# ) ]                                                 \
          [ slave  | master ]                                                         \
          [ noslave ]                                                                 \
          [ nomaster ]                                                                \
          [ nodeprop @ ]
```

Vertex, as the compulsory keyword, is followed by its identification number. A vertex may be determined either by three coordinates preceded by keyword ***xyz*** or by fixation to either of vertex, curve, or surface. This is expressed by keyword ***fixed*** followed by the appropriate entity keyword (*vertex*, *curve*, or *surface*) and its identification number. When the fixation to curve or surface is used parametric coordinates preceded by keyword *t* in the case of curve or keyword *uv* in the case of surface must be provided otherwise the parametric position of the vertex on the curve or surface is calculated using its coordinates specified after keyword *xyz* or inherited from the parent vertex. If the position of the vertex on a parent entity is overdetermined the redundant information is used to avoid ambiguity of the vertex position. Note that the original position (given by real coordinates) of the vertex and its position on the parent entity (given by parametric coordinates) are not allow to differ by more than user defined epsilon specified on the command line (see command line option *–e*). A vertex may be fixed simultaneously to a vertex and curve or to a vertex and surface. In this case, repeated use of keyword *fixed* is allowed. A size specification may be assigned to a vertex. This can be done using keyword ***size*** followed by a concrete number or a special keyword, optionally followed by a multiplication factor preceded by an asterisk. Special keyword *def* stands for a default mesh size specified on the command line when running the program (see command line option *–d*), *uni* denotes the uniform mesh size specified on the command line (see command line option *–u*), *ver*, *cur*, and *sur* is used in the case of fixation when the mesh size is to be the same as on the appropriate parent model entity (vertex, curve, or surface) at vertex location. By default, a vertex fixed to a parent entity inherits mesh size from that entity. In the case of a multiple fixation to vertex and curve or surface the mesh size of parent curve or surface is used. If no size is specified and the vertex is not fixed to any entity the default size from command line is used. Only positive mesh size is allowed at vertices. Note that the size assigned to a vertex is used for interpolation of mesh size over curves and surfaces sharing that vertex. The unit weight is assumed for all non-fixed vertices unless a weight is specified after keyword *weight*. Only positive weight specification is accepted. A vertex fixed to a parent entity inherits weight from that entity. Note that this inherited weight can be overridden by new weight specification (keyword ***weight***) only in the case of vertex to vertex fixation. Keyword ***factor*** is used to specify the vertex mesh size multiplication factor. Note that this factor is not used when interpolating mesh size specification over curves and surfaces sharing the vertex. Setting it to zero will cause excluding this vertex from local mesh size control. Negative mesh size factors are not accepted. Default value of mesh size factor is equal to 1. An integer property number may be assigned to a vertex after keyword ***property***. Keyword ***virtual*** marks a vertex as virtual. Keyword ***hidden*** marks it as a hidden one, which means that the location of the mesh node associated with this vertex is not fixed and may change during the smoothing process. The output of node generated at vertex position may be enforced or suppressed by setting *yes* or *no* after ***output*** keyword. By default, each physical non-hidden vertex is designated for output, except those fixed vertices with no physical ancestor designated for output. Suppressed output for a vertex bounding a physical curve or fixed to a physical curve designated for output is ignored. If generation of elements on a model entity between the vertex being just defined and other close vertices (already defined and bounding that entity) is to be prevented, the other vertices should be specified after keywords ***coincide*** *vertex*. Neither from the coinciding vertices is allowed to be fixed to physical vertex or vertex with physical ancestor. Keywords ***slave*** and ***master*** are designated (but not limited to) for use with structured meshes to extend the modelling capabilities of otherwise limited topology concept. A vertex may be marked as slave to enable merging of separate meshes or as master to enable its fixing to a model entity without explicit fixation declaration. If keyword *slave* is used the mesh space is searched for the nearest node (not connected by an edge to the node classified to the just defined vertex) which is, if sufficiently close, used to replace in the mesh connectivity the node classified to the just defined vertex. Note that nodes classified to model entity marked either slave or nomaster are excluded from the search. Similarly, keyword *master* enforced search through the mesh space for the nearest node (classified to model entity marked neither master nor noslave and not connected to the nodes classified to the just defined vertex) which is, if sufficiently close, replaced in the mesh connectivity by the node classified to the just defined vertex. The proximity criterion is based on the comparison of the distance between the corresponding nodes and 10 % of the average mesh size required at location of the nodes. If the distance of the next closest candidate from the node classified to the just defined vertex is not larger than the distance between the corresponding nodes by at least 1 % of the local mesh size, the shortest edge connected to any of corresponding nodes is identified. If the 10 % of the length of that edge is smaller then the difference between the distances of the first two candidates from the node classified to the just defined vertex, an error concerning ambiquity is issued. Note that the local mesh size is derived from the octree and can be therefore much different from both the mesh spacing prescribed at the vertex and the real mesh spacing (e.g. when mapping is applied). This can lead to master/slave node identification failure which can be prevented by appropriate mesh size specification. Keyword *nomaster* can be used to prevent node at the just defined vertex to replace node classified to an entity marked as slave. Similarly, keyword *noslave* can be employed to prevent node at the just defined vertex to be replaced by the node classified to an entity marked as master. Note that except simultaneous use of keywords *slave* and *master*, all master-slave related keywords can be used together in any combination. If the slave vertex is marked as coinciding with other vertices, the master node is searched only at these coinciding vertices. Note that similar behaviour does not hold for master vertex which is coinciding with other vertices, in which case ordinary search is performed. Nesting of master-slave relations is not supported. This implies that a particular node cannot be merged simultaneously with a master vertex and a slave vertex. Note that when using master or slave relationship the output of element boundary element entities and boundary associated elements (see Sections 7 and 8 for details) are not fully consistent (because the classification of mesh entities to model entities is overriden by the slave and master relationships). Keyword ***nodeprop*** (a supplement to property specification) defines a single quoted string that is assigned to the vertex and that relates to the node at the vertex. Note that this string is not part of the output and can be accessed only via special function call (when T3d is executed as a subroutine).

In the current implementation, the keyword *hidden* is ignored.

3.2 Input Records of a CURVE

```
Curve in# { vertex in# in# }                                                    \
        [ order in#  || fixed { curve | surface } in# ]                         \
        [ size { fp# | def | uni | cur | sur } [ * fp# ] ]                       \
        [ density { fp# | def } [ * fp# ] ]                                      \
        [ factor fp# ]                                                          \
        [ rate { fp# | def } [ * fp# ] ]                                         \
        [ nosplit fp# ]                                                         \
        [ property in# ]                                                        \
        [ virtual ]                                                             \
        [ hidden ]                                                              \
        [ polysize ( [ * in# ] { fp# | def | uni } ) ]                           \
        [ count in# ]                                                           \
        [ output [ elem | node ] { yes | no } ]                                 \
        [ coincide curve ( in# ) ]                                              \
        [ bassoc { yes | no } ]                                                 \
        [ swap { yes | no } [ angle fp# ] ]                                     \
        [ smear { yes | no } [ angle fp# ] ]                                    \
        [ mirror in# [ weak | simple ] [ octree { yes | no } ] ]                \
        [ duplicate in# ]                                                       \
        [ equidistant ]                                                         \
        [ transition { fp# | def } { fp# | def } ]                              \
        [ slave | master ]                                                      \
        [ noslave ]                                                             \
        [ nomaster ]                                                            \
        [ nodeprop @ ]                                                          \
        [ elemprop @ ]                                                          \
        [ elemgroup in# ]

Polygon in# { xyz fp# fp# fp# | poly in# }                                      \
          [ size { fp# | def | uni | cur } [ * fp# ] ]                           \
          [ weight fp# ]
Polyend
```

Curve, as the compulsory keyword, is followed by its identification number. Identification numbers of the starting and ending vertices are specified after keyword ***vertex*** and determine the curve orientation. The curve order may be specified after keyword ***order***. The curve may be fixed to a curve or to a surface which is expressed by keyword ***fixed*** followed by the appropriate entity keyword (*curve*, or *surface*) and its identification number. If the curve order is not provided and the curve is not fixed to any entity the default value 2 is used as the order and no consequent polygon records are expected. When the curve is fixed to an entity, the curve order may but needs not be specified. In the former case, curve order must be greater or equal to the order of the appropriate parent entity or set to zero (the order of the parent entity is used in that case). When the curve is fixed to a surface its end vertices must lie on the same parametric curve of the surface (the user supplied epsilon is used as a tolerance) and must be fixed to surface itself or to a curve top parent of which is fixed to the surface. A physical curve fixed to a physical solid surface is not allowed to be intersected by another physical curve fixed to the same surface. The orientation of the fixed curve is independent of the orientation of the parent model entity. The size specification preceded by keyword ***size*** is also similar to the one in Section 3.1 but two differences should be mentioned. Firstly, the size specification is treated as an upper bound of the required size extracted from the mesh size specification of curve control points and secondly, if the specification is missing or zero value is specified, no default value is used and the upper bound limit is not applied (unless a nonzero value is propagated from any incident entity of next higher dimension). For all model entities, the (nonzero) size is recursively propagated to boundary entities of the next lower dimension until a smaller (nonzero) size is encountered. Keyword ***density*** controls the relative mesh density along the curve by setting the upper bound of the curve mesh size as the ratio between the length of the curve and given value (generally corresponding to desired number of segments) which is specified as a concrete number or as default value (controlled by the command line option *–t*) using a special keyword *def*, optionally followed by a multiplication factor preceded by an asterisk. For example, using value 0.5 means that the mesh size around the curve will be set to the double of the curve length. The value of the density has no upper bound limit and must not be negative. If the curve mesh size is specified in multiple ways (for example using the density specification together with the mesh size interpolated from points of control polygon), the smaller nonzero value is considered at each particular location. Keyword ***factor*** specifies the curve mesh size multiplication factor which is applied to both curve mesh size specification and mesh size extracted from curve control points as well. When set to zero, the curve is excluded from local mesh size control. Keyword ***rate*** is used to define the accuracy of the geometrical representation of the curve by its discretization in terms of a ratio between the appropriate mesh size and radius of curvature at any location on the curve. The rate is specified as a concrete number or as default value (controlled by the command line option *–w*) using a special keyword *def*, optionally followed by a multiplication factor preceded by an asterisk. The value of the rate has no upper bound limit and must not be negative. Keyword ***nosplit*** can be used to indirectly specify average mesh density along the curve below which the curve (or its part between subsequent vertices) is discretized by a single segment and above which it is discretized by proporcionally reduced number of segments. For example, using the value 0.25 means that the curve will be discretized by a single segment whenever the octree mesh size control yields four (= 1.0 / 0.25) or less segments, otherwise the number of segments dictated by the octree will be reduced by factor 0.25. The value of the non-splitting density must not be negative, the value larger than one is irrelevant. Zero value prevents the splitting for any mesh density. This specification is applied only to curves without the prescribed number of segments bounding physical surface, patch, or shell entity subjected to unstructured discretization and not bounded by any physical surface, patch, or shell entity subjected to mapped meshing. Note that the discrepancy between the size of generated segment and density of the underlying octree data structure may lead to poor quality elements or mesh generation failure. The meaning of keywords ***property*** and ***virtual*** is the same as explained in Section 3.1. Keyword ***hidden*** marks curve as a hidden one which allows the repositioning of mesh elements and nodes of this curve during the smoothing process without respecting geometrical restriction of their fixation to this curve. The ***polysize*** keyword is used for fixed curves only if the inheritance of mesh size specification from the parent model entity is not desirable. In that case, the size specification must be provided for each internal control point in subsequent order. The repetition number preceded by an asterisk may be used if the specification is the same for several subsequent control points. The meaning of keywords *def* and *uni* is the same as described in Section 3.1. Keyword ***count*** can be used to enforce division of the curve to a specified number of segments. The count specification is reflected only by physical curves not bounding any physical solid surface, patch, or shell and not being fixed to other physical model entity. When using support for quadrilateral and hexahedral meshing, keyword *count* can be also used to prescribe tessellation of quad mappable surface, patch, or shell or of hexa-mappable region along the curve. Note that in such a case, the final number of segments may be modified due to possible inconsistency of specification on associated curves during the tessellation propagation over quad and hexa-mappable model entities. If the tessellation count is not specified and cannot be propagated from another curve, it is calculated from the mesh size specification. Note that for curves with count specification, the the other mesh size control mechanism (as *size*, *density*, etc.) are not applied unless the count specification is (for any reason) ignored, in which case a warning is issued. The output of segments on the curve may be enforced or suppressed by setting *yes* or *no* after ***output*** keyword. By default, curves bounding a physical solid surface, patch, or shell designated for output are not designated for output. The physical curve is not subjected to the discretization if the curve itself and neither from the physical entities or interfaces of higher dimension being bounded by the curve or to which the curve is fixed is designated for output. This enables to suppress discretization of curves bounding a surface, patch, shell, or region not designated for output without explicitly preventing their output. If the discretization of those curves should not be suppressed, either their output must be enforced or the surface, patch, shell, or region must be marked as *virtual*. Note however, that physical curve marked as *slave* or *master* (see bellow) or being prototype of mirroring is subjected to the disretization even if not designated for output. Optionally, there may be used keywords *output* ***elem*** *no* (or *output* ***node*** *yes*) to suppress output of elements while not preventing the discretization of the surface. This is useful if a physical curve not bounding any physical surface, patch, or shell (thus undesirably designated for output by default) is assumed to be merged (thus its discretization is necessary) with a slave curve bounding a physical surface, patch, or shell. Note that specification *output elem yes* is equivalent to *output yes* and specification *output node no* is equivalent to *output no*. Similarly as in Section 3.1, if generation of elements on a model entity between the curve just being defined and other close curves (already defined and bounding that entity) is to be prevented, the other curves should be specified after keywords ***coincide*** *curve*. Neither from the coinciding curves is allowed to be fixed to physical curve or curve with physical ancestor. Keyword ***bassoc*** followed by specification *yes* or *no* controls separate output of triangles and quadrilaterals (ids) classified on surfaces, patches, and shells bounded by the curve. Note that a specific command line option (see option *–A*) controls whether all or none of the physical curves will be by default designated for output of associated elements. Keyword ***swap*** enables or disables by setting *yes* or *no* swapping of curve segments in a quadrilateral formed by two adjacent triangles classified to two different planar model entities (surface, patch, or shell) sharing that curve and subjected to triangular meshing. The curve must be two-manifold and the two incident model entities must possess outer normal of the same orientation. The segment swapping is not performed if the angle of the outer normals of those two model entities is greater than the value specified by keyword *angle* following swap specification. Keyword ***smear*** enables or disables by setting *yes* or *no* smearing of nodes (including vertices) of curve shared by two different planar model entities (surface, patch, or shell) and subjected to triangular meshing. The curve must be two-manifold and the two incident model entities must possess outer normal of the same orientation. The node smearing is not performed if the angle of the outer normals of those two model entities is greater than the value specified by keyword *angle* following smear specification. Swap and smear specification is intended to improve quality of solid discretization of regions described by polyhedrons with high number of faces. Note that the angle value defaults to 45 degs. Keyword ***mirror*** followed by a prototype curve id enables mirroring of prototype curve discretization by the curve being defined (typically used when periodic boundary conditions are required). Note that the mirroring is exact only if the two curves are geometrically identical (including location of fixed vertices). In other cases, the mirroring yields generally only similar meshes or an error if the segments of the prototype curve cannot be accommodated on the mirror curve. Keyword ***weak*** following mirror curve specification weakens the mirroring in terms of not using strictly the same parametric coordinates of nodes on prototype curve for nodes on mirror curves. This is usefull either when the node distribution on the mirror copy curve should reflect the local mesh density specification, if there are no vertices fixed to it, or when the number of edges between the subsequent vertices fixed to it should be the same on both curves. The latter case implies that there is the same number of fixed vertices on both curves and that the vertices are ordered in the same way. The exact mirroring can be enforced irrespectably whether the prototype and mirror curves are geometrically identical by using keyword ***simple*** after mirroring specification, in which case strictly the same parametric coordinates are applied. This can be advantageously employed when the prototype and mirror curves are geometrically almost the same (for example due to round off errors related to coordinates and weights defining the vertices of the curve control polygon). Note however that enforcing simple mirroring for geometrically different curves may lead to the deterioration of the mesh quality or even to meshing failure. Also note that the keywords *weak* and *simple* are mutually exclusive. If the mirror curve is bounding or is fixed to a surface, patch, or shell which is subjected to unstructured meshing or which is bounding a region subjected to unstructured meshing, then the octree along the curves need to be mirrored as well in order to ensure that the elements on both curves are in agreement with the local octree spacing. The octree mirroring is performed automatically if the mirroring curves are geometrically identical or if simple mirroring is enforced. Generally, the octree mirroring can be enforced or suppressed by setting *yes* or *no* after keyword ***octree*** within mirroring specification. Note however that enforcing octree mirroring for geometrically (and especially parametrically) different curves may lead to undesirable (sometimes also pathological) octree refinement. Undesirable octree refinement may be also observed when portions of the octree along the curves mutually interfere. Keyword ***duplicate*** followed by a counterpart curve id enables propagation of tessellation of the structured mesh and equidistant request (see keyword *equidistant* below) from and to the counterpart curve. Thus there is no need to specify count and/or equidistant request on both curves. The counterpart curves should be identical (topologically and geometrically) otherwise the request is ignored (a warning is issued) and the mesh conformity might be violated. A compatible mesh along duplicated curves is generally produced only if the curves are spatially equivalent. Note that for poorly parameterized curves the exact compatibility of the mesh is guaranteed only if the curves are, except of topological, geometrical and spatial identity, also of the same orientation (with respect to the spatial location). When a curve is to be discretized by segments of equal length, keyword ***equidistant*** should be used. Note that the equidistant node distribution request is accepted only by the top physical curves. If there is a vertex fixed to such a curve, the equidistant request is ignored for this curve and it is propagated to all its parts including fixed curves. When using support for quadrilateral and hexahedral meshing, the equidistant request may be specified also for fixed curves. If there is a vertex fixed to such a fixed curve, the equidistant request is ignored for this curve and if there exist a chain of subcurves fixed to it and joining its end vertices, it is propagated to all these fixed subcurves. Note that on curves bounding surfaces, patches, and shells subjected to unstructured meshing, the equidistant request is checked to comply with the required mesh size distribution along the curve. In the case of large discrepancies, the equidistant request is ignored. Note that the equidistant node distribution along a curve is achieved only along curves with constant (including infinite) curvature. For curves with variable curvature, the differences in lengths of individual segments increases with the growing change in the curvature and decreases with the number of segments. Keyword ***transition*** enables to control the mesh transition between the structured and the unstructured mesh. The first parameter, ranging from 1 to 6, corresponds to the maximal allowable aspect ratio of unstructured elements in the transition from the fine structured to the coarse unstructured mesh. The second parameter, ranging from 1 to 2, corresponds to the maximal allowable aspect ratio of unstructured elements in the transition from the coarse structured to the fine unstructured mesh. The default value of either of the transition parameters can be used by using keyword *def*. Note that the default values of transition parameters can be overridden by command line option *–J*. Keywords ***slave*** and ***master*** have the similar meaning as those described in Section 3.1, with the only difference that all nodes on the curve are subjected to that process. Note that for each edge on the slave or master curve there must be localized master or slave counterpart on other entity. The meaning of keywords ***noslave*** and ***nomaster*** is the same as described in Section 3.1. Also the meaning of keyword ***nodeprop*** is the same as explained in Section 3.1. Similar meaning has keyword ***elemprop*** which specifies a single quoted string that is assigned to the curve and relates to the elements on the curve. Keyword ***elemgroup*** can be used to associate the elements (edges) generated on the curve with an integer type. Note however, that similarly as the *nodeprop* and *elemprop* strings, also this number can be accessed only via a special function call (when T3d is executed as a subroutine).

A curve may be degenerated into a single point (collapsed curve) if it is bounded by vertices located at that point and if all control points are also located at that point. This is naturally achieved if the same vertices are used to bound the curve. If different vertices are used these vertices must have a common physical parent vertex. Collapsed curve is not allowed to be fixed to a surface or non-collapsed curve. If a vertex is fixed to a collapsed curve with different vertices the parameter t must be specified.

When generating structured quadrilateral meshes on two only partially adjacent model entities (each of them is incident to a different part of the shared curve), keyword *count* can be effectively used to control the tessellation only if it is specified for each part of the shared curve (this implies that there must be a curve fixed to each part of the parent curve, even that used only just by one of the model entities). Otherwise the structured meshing request on one of the model entities will be ignored.

The primary purpose of keyword *duplicate* is to ensure generation of compatible structured mesh over several separate parts of the domain which are not connected by a common model entity. Note that the meshes along the interface corresponding to the missing common entity will be generally compatible only if counterpart entities will be identical topologically and geometrically, and if they will be at the same location or with equidistant request. However if the mirroring is supported the same effect may be achieved by using keyword *mirror*. Moreover, keyword *mirror* can be applied also to unstructured meshes and it yields compatible meshes along the interface even if the counterpart entities are not at the same location (but they must be still topologically and geometrically identical). Note, that a natural compatibility is obtained in the case of unstructured meshes even without the keyword *mirror* if the counterpart entities along the interface are topologically, geometrically, and spatially identical.

A set of internal control points must be provided for each non-fixed curve of degree $n > 1$ unless the automatically computed location of all or some of the control points should be employed. It means that input record of such a curve may be followed by up to $n-1$ records for individual internal control points of the curve. The number of the control point in the range from 1 to $n-1$ specified after keyword ***Polygon*** is used as the control point sequence number with respect to the curve orientation and can be referenced only in the context of the current curve. The control point may be specified by three coordinates preceded by keyword ***xyz*** or may be associated with any already specified polygon control point using keyword ***poly*** followed by the sequence number of that control point. Keywords ***size*** and ***weight*** have a similar meaning to the one explained in Section 3.1 but even zero or negative values of size and weight specification can be used. Note however that the denominator in Eqs (1), (2), (5), and (6) must not be negative or zero. If size specification is omitted and the default zero value is not overidden by any other default specification (for example using the command line options for default or uniform mesh size (see command line options *–d* and *–u*), a warning is issued. If zero or negative specification is used explicitly, no warning is generated. If not all internal control points are specified, the record starting with keyword ***Polyend*** must be used as last.

In the current implementation, keyword *hidden* is ignored, unless only two planar model entities (physical, solid, non-hidden surface, patches, and shells) with the same property and with collinear normals of the same orientation are sharing the given curve with no physical parent or child. Note that using hidden curves has an impact on the performance because the surfaces, patches, and shells sharing those curves are subjected to an additional smoothing.

3.3 Input Records of a SURFACE

```
Surface in# { curve in# in# in# in# }                                          \
            [ order in# in# || fixed surface in# ]                             \
            [ size { fp# | def | uni | sur} [ * fp# ] ]                        \
            [ factor fp# ]                                                     \
            [ rate { fp# | def} [ * fp# ] ]                                    \
            [ property in# ]                                                   \
            [ virtual]                                                         \
            [ hidden]                                                          \
            [ hole]                                                            \
            [ output [ elem | node] { yes | no } ]                             \
            [ polysize ( [ * in# ] { fp# | def | uni } ) ]                     \
            [ ( coincide { surface | patch | shell} ( in# ) ) ]                \
            [ bassoc { yes | no } ]                                            \
            [ quad]                                                            \
            [ map { yes | no } ]                                               \
            [ lcs in# in# ]                                                    \
            [ diagonal { none | positive | negative | point fp# fp# fp# } ]    \
            [ equidistant]                                                     \
            [ mirror in# [ weak | simple] [ octree { yes | no } ] [ smooth { yes | no } ] ]   \
            [ duplicate in# ]                                                  \
            [ ( attract fp# { all | vertex ( in# ) | curve ( in# ) } ) ]       \
            [ slave | master]                                                  \
            [ noslave]                                                         \
            [ nomaster]                                                        \
            [ nodeprop @]                                                      \
            [ elemprop @]                                                      \
            [ elemgroup in# in# ]

Polygon in# in# { xyz fp# fp# fp# | poly in# in# }                             \
                [ size { fp# | def | uni | sur} [ * fp# ] ]                    \
                [ weight fp# ]
Polyend
```

Surface, as the compulsory keyword, is followed by its identification number. Identification numbers of the four bounding curves are specified after keyword ***curve*** in clockwise order when viewing the surface against its (outer) normal (see below for its definition). The first and second curves determine the u and v directions, respectively, on the surface with the origin at their common vertex. The orientation of surface (outer) normal is then given by the vector product of vectors tangent to the parametric curves in u and v directions and oriented in the positive u and v directions, respectively. The orientation of bounding curves is not relevant. No two adjacent curves are allowed to be degenerated into a single point and no two opposite or adjacent curves are allowed to be the same or coinciding. This disables creation of surfaces degenerated into a curve or point. The surface orders in u and v directions may be specified after keyword ***order***. The surface may be fixed to a surface which is expressed by keywords ***fixed*** *surface* and the identification number of the parent surface. If surface orders are not provided and the surface is not fixed to any surface the orders are extracted from the curves bounding the surface. When the surface is fixed to the surface, the orders may but need not be specified. In the former case, order in any direction must be greater or equal to the order of the parent surface or set to zero (the order of the parent surface is used in that case). When the surface is fixed to the surface its bounding curves must be fixed to the parent surface itself. They are not allowed to be fixed to or be curves bounding the parent surface unless all four curves bounding the parent surface are used to bound the fixed surface, in which case the fixed surface is handled as a copy of the parent surface (sharing with it all boundary curves and vertices). This may be used to expand the parent surface without the need to specify its control polygon points. The ordering of bounding curves of the fixed surface must be in agreement with bounding curves ordering of the parent surface. Since physical curves fixed to a physical solid surface are not allowed to intersect each other, the physical surfaces fixed to the same physical solid surface cannot overlap. The meaning of keywords ***size***, ***factor***, ***rate***, ***property***, ***virtual***, ***hidden*** and ***polysize*** is the same as explained in Section 3.2. It should be mentioned however that the smallest of both principal radii of curvature on the surface is considered when the rate is specified. Only surface shared by two solid physical regions of the same property may be hidden. Also note that for surfaces and curves with nonzero size the (nonzero) value of factor is recursively propagated to the boundary entities of the next lower dimension until a smaller (nonzero) factor is encountered. Keyword ***hole*** is used to specify that the currently defined surface forms a hole. The output of triangles and quadrilaterals on a surface may be enforced or suppressed by setting *yes* or *no* after ***output*** keyword. By default, surfaces bounding a physical solid region are not designated for output. The physical surface is not subjected to the discretization if the surface itself and neither from the physical entities or interfaces of higher dimension being bounded by the surface or to which the surface is fixed is designated for output. This enables to suppress discretization of surfaces bounding a region not designated for output without explicitly preventing their output. If the discretization of those surface should not be suppressed, either their output must be enforced or the region must be marked as *virtual*. Note however, that physical surface marked as *slave* or *master* (see bellow) or being prototype of mirroring is subjected to the disretization even if not designated for output. Optionally, there may be used keywords *output* ***elem*** *no* (or *output* ***node*** *yes*) to suppress output of elements while not preventing the discretization of the surface. This is useful if a physical surface not bounding any physical region (thus undesirably designated for output by default) is assumed to be merged (thus its discretization is necessary) with a slave surface bounding a physical region. Note that specification *output elem yes* is equivalent to *output yes*, and specification *output node no* is equivalent to *output no*. The meaning of keyword ***polysize*** is the same as explained in Section 3.2. Keep in mind, however, that the control points on a surface are ordered in such a way that the first index (corresponding to u direction) is running faster. Similarly as in Section 3.1, if generation of elements on a model entity between the surface just being defined and other close surfaces, patches, or shells (already defined and bounding that entity) is to be prevented, the other surfaces, patches, or shells should be specified after appropriate model entity keyword (*surface*, *patch*, or *shell*) preceded by keyword ***coincide***. Keyword ***bassoc*** followed by setting *yes* or *no* controls separate output of tetrahedra, pyramids, wedges, and hexahedra (ids) classified to region bounded by the surface. Note that a specific command line option (see option *–A*) controls whether all or none of the physical surfaces will be by default designated for output of associated elements. Generation of unstructured quad-dominant mesh may be specified by keyword ***quad***. Generation of structured quad mesh can be enforced or suppressed by keyword ***map*** followed by setting *yes* or *no*. Note that if directive *T3D_QUAD_HEXA_MESHING* is not defined, keyword *quad* is equivalent to keywords *map yes*. The structured mesh can be built only on quad-mappable surface with four non-collapsed boundary curves without any fixed physical curve but with few fixed vertices (larger amount or inappropriate location of fixed vertices may cause generation of unstructured mixed mesh or failure). If the surface is not quad-mappable, either unstructured trinagular mesh or unstructures quad-dominant mesh, if keyword *quad* was used, is produced. Note that fixation of vertices and curves to surface with structured mesh request can be alternatively accomplished via the *master* keyword when defining the vertex or curve. The tessellation of structured mesh may be controlled by count specification on boundary curves. The tessellation count is automatically propagated over opposite curves (in a recursive manner) and needs not be therefore specified for each curve. Note that the quad mapping request is generated automatically if the surface is bounding a hexa-mappable region with hexa mapping request (see Section 3.6). The local coordinate system controlling the ordering of mapped quads can be defined by keyword ***lcs*** followed by identification numbers of two neigbouring curves bounding the surface. The rows of structured quads are generated along the first curve in the direction from the vertex shared with the second curve. The node ordering of each quad is counterclockwise with respect to the surface normal and starts at the node topologically closest to the surface vertex shared by those two curves. If there are vertices fixed to the surface or convexity check (specified via command line option *–K*) is prescribed, the above node ordering is guaranteed for each quad only if the resulting mesh is strictly structured. Keyword ***diagonal*** can be used to generate structured triangular mesh by diagonal splitting each of the structured quad into two triangles. The two possible diagonal choices can be specified by keyword *positive* and *negative*, respectively, following the keyword *diagonal*. The positive diagonal is formed by the first and the third quad nodes, the negative diagonal is the other one. Note that these two choices are unambiguous only if the local coordinate system is specified (otherwise the quad ordering and consequently also the orientation of the positive and negative diagonal is mesh dependent). Alternatively, the choice of the diagonal can be controlled by keyword *point* preceded by keyword *diagonal* and followed by three coordinates of a point. Then the quad diagonal is employed that is formed by the node, which distance from that point is the smallest. Note that this specification may be ambiguous if the point is located on the symmetry plane of any of the quad edges. The diagonal splitting is prevented if keyword *none* is used after keyword *diagonal*. Note that diagonal splitting request is ignored if the accommodation of fixed vertices results in lost of structured character of the mesh. Keyword ***equidistant*** can be used to make the structured mesh regular (without taking into account the required mesh size distribution) by propagating the equidistant request to all boundary curves. Keyword ***mirror*** followed by a prototype surface id enables mirroring of prototype surface discretization by the surface being defined. The prototype surface must be topologically identical and geometrically similar with the surface being defined otherwise the mirror request is ignored. Note that the mirroring is exact only if the mirrored surfaces are geometrically identical (including location of fixed curves and vertices in the interior and even on the boundary). If they are not geometrically identical, the mirroring is based on a mapping between the parameterization of both surfaces derived from the least squares fit of all constraining nodes. In this case however, an invalid mesh (with inverted elements) may be produced if the constraints do not allow to properly accommodate elements from the prototype surface. The mirroring using the parameterization mapping may be enforced even for geometrically identical surfaces if keyword ***weak*** is applied after the mirroring specification. On the other hand, simple mirroring may be enforced using keyword ***simple*** after mirroring specification in which case the parameterization mapping is omitted and strictly the same parametric coordinates of nodes on the prototype surface for nodes on mirror surface. Note however that enforcing simple mirroring for geometrically different surfaces may lead to the deterioration of the mesh quality or even to meshing failure. If the mirror surface is bounding or fixed to a region which is subjected to unstructured meshing then the octree along the surfaces need to be mirrored as well in order to ensure that the elements on both surfaces are in agreement with the local octree spacing. The octree mirroring is performed automatically if the mirroring surfaces are geometrically identical or if simple mirroring is enforced. Generally, the octree mirroring can be enforced or suppressed by setting *yes* or *no* after keyword ***octree*** within mirroring specification. Note however that enforcing octree mirroring for geometrically (and especially parametrically) different surfaces may lead to undesirable (sometimes also pathological) octree refinement. Undesirable octree refinement may be also observed when portions of the octree along the surfaces mutually interfere. The weakly mirrored surface mesh is generally subjected to an additional smoothing. This may lead to not negligibly different meshes of almost identical prototype and mirrored surfaces. To alleviate this effect, the additional smoothing of mirrored surface mesh can be suppressed by using ***smooth*** *no* within mirror specification. Alternatively it is possible to prescribe additional smoothing to mesh of the prototype surface by using *smooth yes* within a fictitious mirroring specification (using zero id for the prototype surface) when defining the prototype surface. The latter approach is preferred as it leads generally to slightly better mesh quality. Note, however, that it is not possible to use keywords *simple*, *weak* and *octree* in the fictitious mirroring specification. The mirror and weak requests are propagated to all curves bounding the prototype surface or being fixed to it. Note however that this is not the case for simple and octree requests. Keyword ***duplicate*** followed by a counterpart surface id enables propagation of tessellation of the structured mesh and equidistant request from and to the counterpart surface. Thus there is no need to specify quad mapping and/or equidistant request on both surfaces (or even on neither from them if the request is propagated at least to one of them from hexa-mappable region bounded by the surface). The counterpart surfaces must be topologically identical to enable propagation of the keyword *duplicate* to boundary curves. If they are also geometrically identical mesh conformity is ensured. Duplication request on quad non-mappable surfaces is ignored. Keyword ***attract*** is used to define an attraction (≥ 0) or a repulsion (≤ 0) coefficient for particular boundary nodes during mesh smoothing process (on unstructured mesh only) of the surface. These may be all boundary nodes (including internal boundary, even that defined later) if keyword *attract* is followed by keyword *all*, or nodes at particular vertices or on particular curves, defined after keyword *vertex* or keyword *curve*, respectively, preceded by keyword *attract*. Note that the coefficient must be in range from -1 (exclusive) to 10 (inclusive). Zero attraction coefficient prevents any attraction (or repulsion). Attraction coefficients close to range limits may cause smoothing failure. Note that attraction may be specified repeatedly within the same surface definition. If a particular entity (vertex or curves) appears in more than just a single specification (*all* specification exclusive), only the first one is considered (without any warning being issued). This enables to prescribe attraction (using however the same coefficient) to nodes on all internal boundaries defined after the surface itself different from the attraction of nodes on external boundary. Note that the ordering of *all* specification within remaining *attract* specifications is irrelevant. No warning is issued if a vertex or curve not bounding the surface is used to define attraction on the surface. The attraction specification is ignored if the surfaces is bounding a physical region. Keywords ***slave***, ***master***, ***noslave***, ***nomaster*** have the same meaning as described in Section 3.2. When using keyword *slave*, surface (i.e. entity of the same type) on which the master node should be searched for can be specified using the keyword *coincide*. This may be useful if there are several identical surfaces sharing the same location (for example when modelling interface that is connected to the rest of the body using master-slave concept in which case there are altogether four such surfaces) and keyword *nomaster* cannot be effectivelly applied. Note that the output of elements on the conciding surface is suppressed unless explicitly requested by keyword *output*. Keywords *slave* and *master* can be used to simulate nonmanifols models in which the defined surface (if marked as slave) or other surface, patch, or shell (if the defined surface is marked as slave) is shared by more than one region on any side. Note however that in such a case the output of boundary associated elements on such entity is suppressed. Keywords ***nodeprop***, and ***elemprop*** have the same meaning as explained in Section 3.2. Also keyword ***elemgroup*** has the same meaning as described in Section 3.2 with the only difference that two integer numbers associated to triangular and quadrilateral elements (in this order) must be supplied.

A set of internal control points must be provided for each non-fixed surface of degrees greater than 1 in both directions unless the automatically computed location of all or some of the control points should be employed. It means that input record of such a surface of degree $n \times m$ may be followed by up to $(n-1)(m-1)$ records for individual internal control points of the surface. The numbers of the control point (in u direction ranging from 1 to $n-1$ and in v direction ranging from 1 to $m-1$) specified after keyword ***Polygon*** are used as the control point sequence numbers and can be referenced only in the context of the current surface. The control point may be specified by three coordinates preceded by keyword ***xyz*** or may be associated with any already specified polygon control point using keyword ***poly*** followed by the sequence numbers of that control point. Keywords ***size*** and ***weight*** have the same meaning as explained in Section 3.2. If not all internal control points are specified, the record starting with keyword ***Polyend*** must be used as last.

Planar surface must be convex. Note that using hidden surfaces has an impact on the performance because the regions sharing those surfaces are subjected to an additional smoothing.

3.4 Input Record of a PATCH

```
Patch in# { normal fp# fp# fp# | ref fp# fp# fp# }                          \
          { ( boundary curve ( in# ) ) }                                    \
          [ ( subpatch ( in# ) ) ]                                          \
          [ ( fixed { vertex | curve } ( in# ) ) ]                          \
          [ size { fp# | def | uni } [ * fp# ] ]                            \
          [ factor fp# ]                                                    \
          [ property in# ]                                                  \
          [ virtual ]                                                       \
          [ hidden ]                                                        \
          [ hole ]                                                          \
          [ output [ elem | node ] { yes | no } ]                            \
          [ ( coincide { surface | patch | shell } ( in# ) ) ]              \
          [ bassoc { yes | no } ]                                           \
          [ quad ]                                                          \
          [ map { yes | no } ]                                              \
          [ lcs in# in# ]                                                   \
          [ diagonal { none | positive | negative | point fp# fp# fp# } ]   \
          [ equidistant ]                                                   \
          [ mirror in# [ weak | simple ] [ octree { yes | no } ] [ smooth { yes | no } ] ]  \
          [ duplicate in# ]                                                 \
          [ ( attract fp# { all | vertex ( in# ) | curve ( in# ) } ) ]      \
          [ slave | master ]                                                \
          [ noslave ]                                                       \
          [ nomaster ]                                                      \
          [ iso [ tria | quad ] { yes | no } [ dir fp# fp# fp# ] ]          \
          [ origin fp# fp# fp# ]                                            \
          [ epsilon fp# ]                                                   \
          [ nodeprop @ ]                                                    \
          [ elemprop @ ]                                                    \
          [ elemgroup in# in# ]
```

Patch, as the compulsory keyword, is followed by its identification number. The (outer) normal vector of the patch may be specified after keyword ***normal***. If this is not the case, the normal is computed automatically in the least-squares sense with respect to the patch geometry and its orientation is adjusted with respect to the reference point specified after keyword ***ref*** which is assumed to be out of the patch plane and on the side pointed to by the patch (outer) normal. The list of signed identification numbers of curves bounding the patch (from outside or inside) is preceded by keywords ***boundary*** *curve*. The positive number indicates the anticlockwise orientation of the curve (given by its starting and ending vertex) with respect to the patch when viewing it against the outer normal. The negative number indicates the clockwise orientation of the curve. The sign of identification number of collapsed curve forming the boundary of the patch is irrelevant (as the curve itself). The list of identification numbers of patches surrounded by the currently defined patch may be specified after keyword ***subpatch***. Each subpatch must be coplanar with the patch, however its normal orientation is arbitrary. If a subpatch is touching the patch along its boundary curve, this curve must be specified also in the list of curves bounding the patch (including proper orientation). Furthermore, identification numbers of vertices and curves inside the patch can be enumerated after appropriate model entity keyword (*vertex* or *curve*) preceded by keyword ***fixed***. The meaning of keywords ***size***, ***factor***, ***property***, ***virtual***, ***hidden***, ***hole***, ***output***, ***coincide***, ***bassoc***, ***quad***, ***map***, ***lcs***, ***diagonal***, ***equidistant***, ***mirror***, ***weak***, ***simple***, ***octree***, ***duplicate***, ***slave***, ***master***, ***noslave***, ***nomaster***, ***nodeprop***, ***elemprop***, and ***elemgroup*** is the same as described in Section 3.3. Also keyword ***attract*** has similar meaning as described in Section 3.3, but since all external as well as internal boundary entities (vertices and curves) of patch has to be defined before the patch itself, different attraction specification can be used for each of these entities.The factor specification is effective only if size specification is provided. Only patch shared by two solid physical regions of the same property is allowed to be hidden. Keyword ***iso*** enables generation of ideal equilateral elements of the same size (defined by keyword *size*) inside of the patch surrounded by unstructured mesh along the patch boundary. The type of iso elements may be specified by keyword ***tria*** or ***quad*** following the *iso* keyword, otherwise the iso elements are of the same type as requested for elements in the surrounding unstructured mesh. The orientation of the iso elements may be prescribed by a vector (in the patch plane) after keyword ***dir*** following the *iso* specification. If the orientation is not given, an appropriate one is automatically calculated. The iso elements are formed only if the size specification is not too large with respect to patch dimensions, otherwise a warning is issued. Keep in mind that in order to guarantee good quality of unstructured elements, the iso elements are not placed closer to patch boundary than approximately the double of their size. Note that the keyword *diagonal* can be also used to split iso quads into right angle iso-sceles triangles. The full control over the choice of diagonal is available only if the orientation of iso quads is specified. The patch iso meshing request is ignored if the patch is to be mirrored or if structured quad mesh is requested and the patch is quad-mappable. Generally, the patch is assumed to be ideally planar. In the case that the patch deviates slightly from a plane, the position of its pseudo-plane is automatically located making the maximum deviations on both sides equal, unless the position is explicitly given by keyword ***origin*** followed by three coordinates. In any case, if the deviation of any of vertices bounding the patch or fixed to the patch from the pseudo-plane exceeds the user defined tolerance (see command line option *–e*) the patch is considered as nonplanar and can be treated only if the source code has been compiled with the directive *T3D_NONPLANAR_PATCH*. To avoid accidental deviation from the plane (typically due to the manual input data preparation), the maximum deviation is not allowed to exceed value specified after keyword ***epsilon***. If this value is set to zero the user defined epsilon is used. The nonplanar patch can be successfully handled only if boundary curves and fixed vertices and curves projected perpendicularly to its pseudo-plane form a topologically valid patch. Note that in the case of nonplanar patch, its subpatches need not be coplanar with it but their plane or pseudo-plane is not allowed to differ from the pseudo-plane by more than 45 degs. The actual shape of the nonplanar patch is interpolating smoothly the individual boundary curves and nonsmoothly fixed vertices and curves but the user has no explicit control over the shape. This makes the nonplanar patch inappropriate for the modelling of surfaces with exact geometries. If higher order elements are used, mid-nodes classified to nonplanar patch are not projected to the patch and their position is given by simple average of corresponding element corner nodes. Note that nonplanar patch cannot be processed if smoothing is disabled (see command line option *–S*). Note also that support of iso elements on nonplanar patch is not available.

Note that using hidden patches has an impact on the performance because the regions sharing those patches are subjected to an additional smoothing.

3.5 Input Record of a SHELL

Shell $_{\mathrm{in}}\#$ { *bgsurface* $_{\mathrm{in}}\#$ } \\
{ (*boundary curve* ($_{\mathrm{in}}\#$)) } \\
[(*subshell* ($_{\mathrm{in}}\#$))] \\
[(*fixed* { *vertex* | *curve* } ($_{\mathrm{in}}\#$))] \\
[*size* { $_{\mathrm{fp}}\#$ | *def* | *uni* } [* $_{\mathrm{fp}}\#$]] \\
[*factor* $_{\mathrm{fp}}\#$] \\
[*rate* { $_{\mathrm{fp}}\#$ | *def* } [* $_{\mathrm{fp}}\#$]] \\
[*property* $_{\mathrm{in}}\#$] \\
[*virtual*] \\
[*hidden*] \\
[*hole*] \\
[*output* [*elem* | *node*] { *yes* | *no* }] \\
[(*coincide* { *surface* | *patch* | *shell* } ($_{\mathrm{in}}\#$))] \\
[*bassoc* { *yes* | *no* }] \\
[*quad*] \\
[*map* { *yes* | *no* }] \\
[*lcs* $_{\mathrm{in}}\#$ $_{\mathrm{in}}\#$] \\
[*diagonal* { *none* | *positive* | *negative* | *point* $_{\mathrm{fp}}\#$ $_{\mathrm{fp}}\#$ $_{\mathrm{fp}}\#$ }] \\
[*equidistant*] \\
[*mirror* $_{\mathrm{in}}\#$ [*weak* | *simple*] [*octree* { *yes* | *no* }] [*smooth* { *yes* | *no* }]] \\
[*duplicate* $_{\mathrm{in}}\#$] \\
[(*attract* $_{\mathrm{fp}}\#$ { *all* | *vertex* ($_{\mathrm{in}}\#$) | *curve* ($_{\mathrm{in}}\#$) })] \\
[*slave* | *master*] \\
[*noslave*] \\
[*nomaster*] \\
[*nodeprop* @] \\
[*elemprop* @] \\
[*elemgroup* $_{\mathrm{in}}\#$ $_{\mathrm{in}}\#$]

Shell, as the compulsory keyword, is followed by its identification number. The background surface of the shell is specified after keyword ***bgsurface***. Note that the background surface is also part of the model and will be discretized unless marked as hole or virtual. The list of signed identification numbers of curves bounding the shell (from outside or inside) is preceded by keywords ***boundary*** *curve*. The positive number indicates the anticlockwise orientation of the curve (given by its starting and ending vertex) with respect to the shell when viewing it against the outer normal. The negative number indicates the clockwise orientation of the curve. The sign of identification number of collapsed curve forming the boundary of the shell is irrelevant (as the curve itself). Orientation of the outer normal vector of the shell is given by the background surface. The list of identification numbers of shells surrounded by the currently defined shell may be specified after keyword ***subshell***. Furthermore, identification numbers of vertices and curves inside the shell can be enumerated after appropriate model entity keyword (*vertex* or *curve*) preceded by keyword ***fixed***. The meaning of keywords ***size***, ***factor***, ***property***, ***virtual***, ***rate***, ***hidden***, ***hole***, ***output***, ***coincide***, ***bassoc***, ***quad***,

map, ***lcs***, ***diagonal***, ***equidistant***, ***mirror***, ***weak***, ***simple***, ***octree***, ***duplicate***, ***attract***, ***slave***, ***master***, ***noslave***, ***nomaster***, ***nodeprop***, ***elemprop***, and ***elemgroup*** is the same as described in Section 3.3. The factor specification preceded by keyword is effective only if size specification is provided. Only shell shared by two solid physical regions of the same property is allowed to be hidden. Note that in the case of mirroring, the topological identity and geometrical similarity relates also to background surfaces.

Note that using hidden shells has an impact on the performance because the regions sharing those shells are subjected to an additional smoothing.

3.6 Input Record of a REGION

Region $_{\text{in}}\#$ { (*boundary* { *surface* | *patch* | *shell* } ($_{\text{in}}\#$)) } \
[(*subregion* ($_{\text{in}}\#$))] \
[(*fixed* { *vertex* | *curve* | *surface* | *patch* | *shell* } ($_{\text{in}}\#$))] \
[*size* { $_{\text{fp}}\#$ | *def* | *uni* } [* $_{\text{fp}}\#$]] \
[*factor* $_{\text{fp}}\#$] \
[*property* $_{\text{in}}\#$] \
[*virtual*] \
[*hole*] \
[*output* { *yes* | *no* }] \
[*hexa*] \
[*map* { *yes* | *no* }] \
[*equidistant*] \
[*extrude* [{ *surface* | *patch* | *shell* } $_{\text{in}}\#$] \
[*weak* | *simple*] [*octree* { *yes* | *no* }]] \
[*lcs* $_{\text{in}}\#$ $_{\text{in}}\#$ $_{\text{in}}\#$] \
[*iso* { *yes* | *no* } [*sz* { $_{\text{fp}}\#$ | *def* | *uni* } [* $_{\text{fp}}\#$]] \
[*dir1* $_{\text{fp}}\#$ $_{\text{fp}}\#$ $_{\text{fp}}\#$ *dir2* $_{\text{fp}}\#$ $_{\text{fp}}\#$ $_{\text{fp}}\#$]] \
[*nodeprop* @] \
[*elemprop* @] \
[*elemgroup* $_{\text{in}}\#$ $_{\text{in}}\#$ $_{\text{in}}\#$ $_{\text{in}}\#$]

Region, as the compulsory keyword, is followed by its identification number. The list of signed identification numbers of surfaces, patches, and shells bounding the region is specified after appropriate model entity keyword (*surface*, *patch*, or *shell*) preceded by keyword ***boundary***. The positive number means that the outer normal of the surface, patch, or shell points out of the region, the negative number indicates that the outer normal points inside the region. The list of identification numbers of regions surrounded by the region may be specified after keyword ***subregion***. If a subregion is touching the region along its boundary surface, patch, or shell, this boundary entity must be specified also in the appropriate list of entities bounding the region (including proper orientation). Furthermore, the identification numbers of vertices, curves, surfaces, patches, and shells inside the region can be enumerated after appropriate model entity keyword (*vertex*, *curve*, *surface*, *patch*, or *shell*) preceded by keyword ***fixed***. The meaning of keywords ***size***, ***factor***, ***property***, and ***virtual*** is the same as in Section 3.4. Keyword ***hole*** indicates that the currently defined region is a hole. The output of tetrahedra, pyramids, wedges, and hexahedra in region may be enforced or suppressed by setting *yes* or *no* after ***output*** keyword. By default, each physical solid region is designated for output. Note that the physical region is not subjected to the discretization if it is not designated for output. Generation of hexa-dominant unstructured region mesh may be specified by keyword ***hexa***. Generation of structured hexa mesh can be enforced or suppressed by keyword ***map*** followed by setting *yes* or *no*. Note that if directive *T3D_QUAD_HEXA_MESHING* is not defined, keyword *hexa* is equivalent to keywords *map yes*. The structured mesh can be built only on hexa-mappable regions topologically similar with a hexahedron without any fixed physical entity, with six quad-mappable boundary model entities (surface, patch, or shell), each one without any fixed curve or vertex. The quad mapping request is automatically generated on all boundary entities of the hexa-mappable region with the hexa mapping request. Note that the tessellation of structured mesh may be controlled by count specification on boundary curves. The tessellation count is automatically propagated over opposite curves (in a recursive manner) of boundary model entities and needs not be therefore specified for each curve. A semi-structured region mesh composed of prismatic elements (wedges and hexas) can be prescribed by keyword ***extrude***. An extrudable region must be bounded by two opposite topologically identical base entities (surfaces, patches, or shells) and by a set of quad mappable lateral surfaces, patches, and shells. There may be a physical curve, and/or quad mappable physical surface, patch, or shell fixed to the region spanning the whole region in the direction of the extrusion. However, there may be no physical vertex fixed inside the region. Note that fixation of vertices and curves to region with structured or semi-structured mesh request can be accomplished via the *master* keyword when defining the vertex or curve. The keyword ***extrude*** can be optionally followed by the specification of the base entity using one of keywords ***surface***, ***patch***, or ***shell*** with the entity identification number. If the base entity is not specified, it is detected automatically unless the region is found to be not extrudable. Note that the choice of the base entity determines the direction of the extrusion and may have also a significant impact on the quality of the mirroring between the base entities. The extrusion of structured meshes is not supported. In such a case, the request for the semi-structured mesh is automatically replaced by the request for the structured mesh. The keywords ***weak***, ***simple***, and ***octree*** after extrusion specification is used for the mirroring of the base entities and has therefore the same meaning as explained in Section 3.3. Note however that this mirroring request is only implicit and may be reversed or even replaced by other indirected multiple mirroring requests. Thus if the mirroring direction between the base entities is of primary importance it is recommended to explicitly prescribe the mirroring between that pair of entities. Note that simple and octree request in an extrusion specification without the base entity specification may be dangerous if the extrusion direction is ambiguous.

The meaning of keywords ***equidistant***, ***nodeprop***, and ***elemprop*** is the same as described in Section 3.3. Also keyword ***elemgroup*** has the similar meaning as described in Section 3.3 with the only difference that four integer numbers associated to tetrahedral, pyramid, wedge, and hexahedral elements (in this order) must be supplied. The local coordinate system controlling the ordering of mapped hexahedra can be defined by keyword ***lcs*** followed by identification numbers of three curves bounding the region, sharing a common vertex and forming a right hand sided coordinate system. The structured hexahedra are generated in layers along the third curve in the direction from the common vertex. Within each layer, the rows of structured hexahedra are generated along the first curve in the direction from the common vertex. The node ordering of each hexahedron starts at the node topologically closest to the common vertex and follows the counterclockwise ordering within bottom and top base of the hexahedron. The meaning of keyword ***iso*** is similar as in Section 3.4. The orientation of the iso elements may be prescribed by two orthogonal vectors after keyword ***dir1*** and ***dir2*** within keyword *iso* specification. If the orientation is not specified, an appropriate one is automatically calculated if the region is bounded by planar model entities only. Otherwise Cartesian coordinate system is used. The size of iso elements can be prescribed after keyword ***sz*** (using either a concrete number or a special keyword (*def* or *uni*), optionally followed by a multiplication factor preceded by an asterisk) within the *iso* keyword block. If this specification is not provided, the size of iso elements is taken as the default mesh size specified (after keyword *size*) for the region. Note that the request for the generation of iso elements is ignored if neither specification is available, if the available specification is zero or if zero region or mesh size multiplication factor (see command line option *–v*) is applied. The iso elements are formed only if the size specification is not too large with respect to region dimensions, otherwise a warning is issued. The decrease of the size of the iso elements needs not necessarily fix the problem as the size of transition zone (to accommodate variable size elements) may increase in such a case. Note that during the discretization of the transition zone around the iso elements, the default region mesh size is ignored. The region iso meshing request is ignored if structured hexa mesh is requested and the region is hexa-mappable satisfying all other requirements for structured mesh generation.

In the current implementation, generation of unstructured hexahedral meshes is not supported. The pyramid elements are used only in the the region subjected to unstructured tetrahedral meshing with boundary or fixed surface, patch, or shell containing quadrilateral elements or in the region subjected to iso meshing. Wedge elements are used in extruded region only. Generation of iso elements is supported only for hexahedrons. Region with extrusion request is considered as not extrudable if there is a curve bounding surface, patch, or shell fixed to that region not directly fixed to its base entity. Note that the list of boundary surfaces of the region and the list of vertices, curves, and surfaces fixed to the region may also comprise boundary mesh vertices, curves, and surfaces.

3.7 Input Record of an INTERFACE

Interface $_{\text{in}}\#$ { *type* { *vertex* | *curve* | *surface* | *patch* | *shell* } $_{\text{in}}\#$ $_{\text{in}}\#$ } \
  [*property* $_{\text{in}}\#$] \
  [*virtual*] \
  [*mirror* [*weak* | *simple*] [*octree* { *yes* | *no* }]] \
  [*elemfree*] \
  [*output* { *yes* | *no* }] \
  [*nodeprop* @] \
  [*elemprop* @] \
  [*elemgroup* $_{\text{in}}\#$ [$_{\text{in}}\#$]]

Interface, as the compulsory keyword, is followed by its identification number. The type of the interface is specified by appropriate model entity keyword (*vertex*, *curve*, *surface*, *patch*, or *shell*) preceded by keyword ***type*** and followed by identification numbers of pairs of entities forming the interface. Note that the first/second entity from the pair is considered as interface master/slave entity (mind the difference between keywords *master* and *slave* in preceeding sections). The master and slave interface entities must be topologically sufficiently detached. This means that at least a single pair of corresponding (master and slave) nodes (end nodes on edges or corner nodes on triangles and quads) must be distinct nodes. Note that this limitation is relevant only if linear elements are generated. The keywords ***weak***, ***simple***, and ***octree*** following keyword ***mirror*** are used for the mirroring of the interface entities and have therefore the same meaning as explained in Section 3.2 if the interface is of type ***curve*** or in Section 3.3 if the interface is of type *surface*, *patch*, or *shell.* The keyword ***elemfree*** prevents the creation of interface elements and only the nodal connectivity corresponding to interface elements is stored. This is useful when renumbering is applied and the connectivity between the model entities forming the interface should be considered (eg. in the case of periodic boundary conditions). Note however that the renumbering obtained for model with an element free interface is generally different from that obtained for the same model with standard interface. This is consequence of the sensitivity of nodal renumbering algorithm on connectivity ordering. Note that the sides of the interface can be formed by entities marked as slave or master. Interface of type *surface*, *patch*, or *shell* is not allowed, however, to have entities marked as slave on both sides simultaneously. The meaning of keywords ***property***, ***virtual***, ***output***, ***nodeprop***, and ***elemprop*** is the same as in Section 3.6. Keyword ***elemgroup*** can be used to associate the interface elements with an integer type. In the case of the interface between pairs of vertices or curves, only one integer element type corresponding to edge and quadrilateral element, respectively, is expected. In the case of the interface between the pairs of surfaces, patches, and shells, two integer types associated to wedge and hexahedral elements (in this order) are expected. Note that elemgroup numbers can be accessed only via a special function call (when T3d is executed as a subroutine).

The discretization of interface entity is formed by a single layer of elements between the topologically similar meshes on pairs of model entities forming the interface. This is simply achieved if one of those model entities is mirror of the other. If this mirroring is not explicitly prescribed by the user, it is automatically defined with the master entity being the prototype and the slave being the mirror. This implies that the pair of model entities forming the interface must comply with all the criteria relevant for mirroring of model entities of particular type. In the case of interface between vertices, there are no criteria at all. Note that the first/second half of nodes of an interface element (see node numbering in Section 7) are always located on the master/slave side of the interface.

3.8 Input Records Related to Boundary Mesh

In this section, input records for description of model entities based on boundary mesh consisting of triangular and eventually (if support for quad-hexa meshing is enabled) quadrilateral elements are described. Note that these entities are not allowed to interact (be bounded, form boundary of, be fixed to, be parent of) with ordinary model entities except region and interface.

3.8.1 Input Record of Boundary Mesh File

Bnd_mesh { file_name }

Bnd_mesh, as the compulsory keyword, is followed by the name of file with boundary mesh description. The file name can be given by absolute path, in which case it starts by a slash (on Unix/Linux platforms) or by a drive letter followed by colon (on MS Window/DOS platforms), or by relative path, in which case the file name is appended after the last slash (on Unix/Linux platforms) or backslash (on MS Window/DOS platforms) in the input file name, if there is any, or it is used alone, if there is no one. Note that boundary mesh file record must precede all other records related to boundary mesh. The format of the boundary mesh file is described in Section 4.

3.8.2 Input Record of Boundary Mesh VERTEX

```
Bnd_vertex in# { bnd_node in# }                                   \
               [ property in# ]                                   \
               [ output { yes | no } ]                            \
               [ coincide vertex ( in# ) ]                        \
               [ nodeprop @]
```

Bnd_vertex, as the compulsory keyword, is followed by its identification number. Note that conflict with numbering of ordinary vertices must be avoided. The boundary mesh vertex is defined by a boundary mesh node, identification number of which is specified after keyword ***bnd_node***. The meaning of all remaining keywords is the same as explained in Section 3.1.

3.8.3 Input Record of Boundary Mesh CURVE

```
Bnd_curve in# { bnd_edges in# bnd_edge ((in#) | (in#- in#)) }   \
              [ property in# ]                                   \
              [ output { yes | no } ]                            \
              [ bassoc { yes | no } ]                            \
              [ coincide curve ( in# ) ]                         \
              [ nodeprop @]                                      \
              [ elemprop @]                                      \
              [ elemgroup in# ]
```

Bnd_curve, as the compulsory keyword, is followed by its identification number. Note that conflict with numbering of ordinary curves must be avoided. The boundary mesh curve is defined by the number of boundary mesh edges following keyword ***bnd_edges*** and by the enumeration of individual boundary mesh edges forming the curve preceded by keyword ***bnd_edge***. The enumeration has the form of combination of individual identification numbers and their continuous ranges given by pair $_{\text{in}}\#$- $_{\text{in}}\#$, where the first numbers corresponds to the starting boundary mesh edge id and the second number corresponds to the ending boundary mesh edge id. Note that the order in which the boundary mesh edges are enumerated is independent of the actual ordering of the edges on the boundary mesh curve. Also note that the orientation of individual boundary mesh edges is irrelevant. Keep in mind, however, that the boundary mesh edges must form a single continuous manifold curve. The meaning of all remaining keywords is the same as explained in Section 3.2. Note that boundary mesh curve needs not be bounded by any boundary mesh vertex no matter whether it is closed or open. On the other hand, boundary mesh node shared by boundare mesh edges forming different boundary mesh curves must form a boundary mesh vertex. If an open boundary mesh curve is not bounded by vertices, a warning about violated B-rep consistency is issued.

3.8.4 Input Record of Boundary Mesh SURFACE

Bnd_surface $_{\text{in}}\#$ { *bnd_faces* $_{\text{in}}\#$ *bnd_face ((*$_{\text{in}}\#$*)* | *(*$_{\text{in}}\#$- $_{\text{in}}\#$*))* | \
  bnd_quads $_{\text{in}}\#$ *bnd_quad ((*$_{\text{in}}\#$*)* | *(*$_{\text{in}}\#$- $_{\text{in}}\#$*))* } \
  [*normal_face* $_{\text{in}}\#$ || *normal_quad* $_{\text{in}}\#$] \
  [*property* $_{\text{in}}\#$] \
  [*output* { *yes* | *no* }] \
  [*bassoc* { *yes* | *no* }] \
  [*coincide* { *surface* ($_{\text{in}}\#$) | *patch* ($_{\text{in}}\#$) | *shell* ($_{\text{in}}\#$) }] \
  [*nodeprop* @] \
  [*elemprop* @] \
  [*elemgroup* $_{\text{in}}\#$ $_{\text{in}}\#$]

Bnd_surface, as the compulsory keyword, is followed by its identification number. Note that conflict with numbering of ordinary surfaces must be avoided. The boundary mesh surface is defined by the set of boundary mesh triangles and/or boundary mesh quadrilaterals forming the surface. The set of triangles is given by the number of boundary mesh triangles following keyword ***bnd_faces*** and by the enumeration of individual boundary mesh triangles preceded by keyword ***bnd_face***. Similarly, the set of quadrilaterals is given by the number of boundary mesh quadrilaterals following keyword ***bnd_quads*** and by the enumeration of individual boundary mesh quadrilaterals preceded by keyword ***bnd_quad***. The enumeration of boundary mesh triangles and quadrilaterals is done in the same manner as the enumeration of boundary mesh edges described in Section 3.8.3. Note that the order in which the boundary mesh triangles and quadrilaterals are enumerated is independent of the actual ordering of the triangles and quadrilaterals on the boundary mesh surface. Keep in mind, however, that the boundary mesh triangles and quadrilaterals must form a single continuous manifold surface. If not all the triangles and quadrilaterals on the boundary mesh surface have the same orientation of the outer normal, the orientation of the outer normal of the whole surface must be defined by the identification number of a single triangle or quadrilateral following keyword ***normal_face*** or ***normal_quad***, respectively, otherwise an error is issued. The meaning of all remaining keywords is the same as explained in Section 3.3. Note that boundary mesh surface needs not be bounded by any boundary mesh curve no matter whether it is closed or open. On the other hand, boundary mesh edges shared by boundare mesh faces and/or quads forming different boundary mesh surfaces must form a boundary mesh curve. Note also that boundary mesh consisting of quadrilaterals can be adopted only if the source code has been compiled with directive *T3D_QUAD_HEXA_SUPPORT*.

4 Boundary Mesh Input Data Format

There are no keywords in the boundary mesh input data file (the keywords used in following subsection are just for better understanding). Everything on a line behind a # sign is treated as a comment. Any number of blank spaces may be used between individual numbers. Empty lines are ignored.

The first record contains the number of nodes, edges, triangles, and quadrilaterals (use zero if the support for quad-hexa meshing is not enabled).

bnd_nodes *bnd_edges* *bnd_trias* *bnd_quads*

The second record contains the largest ids of nodes and elements in the subsequent lists of boundary mesh nodes and elements. If zero is used for the largest node and/or element id, then it is assumed that the (local) numbering of boundary mesh nodes and/or elements (individually for each type of element) is continuous starting from 1. If nonzero value is used for the largest node and/or element id, then the (global) numbering of boundary mesh nodes and/or elements (irrespectably to their type) is arbitrary within the range from 1 to the largest id.

max_nodes_id *max_elem_id*

For each boundary mesh node, the following record describing the position of the node is expected

node_id *coord_x* *coord_y* *coord_z*

Boundary mesh edges, triangles, and quadrilaterals are described by the following records

edge_id	*nd_ids(1 – 2)*
tria_id	*nd_ids(1 – 3)*
quad_id	*nd_ids(1 – 4)*

which are repeated *bg_edges*, *bg_trias*, and *bg_quads* times. Note that the nodes and elements do not have to be ordered according to their identification number. However, the ordering nodes – edges – trias – quads is obligatory. Only linear boundary elements are supported. Note that not all boundary mesh nodes and elements must be referred to in the actual input file. No matter whether local or global numbering of boundary mesh nodes and/or elements is used, the numbering of nodes and elements in the mesh produced by t3d is continuous starting from 1 (or eventually shifted by a value defined by the appropriate command line option (see *–f* option in Section 8) with no relation to the ordering of nodes and elements in the boundary mesh.

5 Mesh Size Input Data Format

The format of the local mesh size control which is part of model description has been explained in Section 3.

The adaptive mesh size control is based on a background mesh. The required mesh size is specified at nodes of this mesh and is interpolated over the individual elements of the background mesh. The following elements can form the background mesh

- linear edge (2 nodes),
- linear triangle (3 nodes),
- linear quadrilateral (4 nodes),
- linear tetrahedron (4 nodes),
- linear pyramid (5 nodes),
- linear wedge (6 nodes),
- linear hexahedron (8 nodes).

The background mesh is generally independent of the model being discretized but it can be (more likely) tightly connected to it. It is usually identical with the mesh generated during the previous step of an adaptive analysis. The adaptive mesh size input data file is provided by the application analysis code but its manual creation (as an startup mesh size control) is also possible. The description of the background mesh input data format is provided in the following section.

6 Background Mesh Input Data Format

There are no keywords in the background mesh input data file (the keywords used in following subsection are just for better understanding). Everything on a line behind a # sign is treated as a comment. Any number of blank spaces may be used between individual numbers. Empty lines are ignored.

The background mesh input data file can contain either the description of background mesh in a single file or the specification how many independent background mesh files, each one describing a separate background mesh, are to be processed. In the latter case, the format of the first record of the background mesh input data file contains the neutral mesh type identifier (zero), the number of separate background mesh files and their naming type.

0 number_of_bgmesh_files naming_type

The naming type can take three different values – 0, 1, and -1. For naming type equal to 0 or 1, the file names of individual background mesh files are expected to be derived from the name of the background mesh input data file by extending it using a numerical suffix corresponding to the serial number of the particular background mesh file. The numbering of individual background mesh files must be continuous starting either from 0 or from 1, according to the naming type. This naming convention is motivated by the rank numbering in a parallel analysis, because the feature of multiple background mesh files was originally introduced to support parallel adaptive analysis. For naming type equal to -1, the names of the individual background mesh files are present directly in the background mesh input data file on subsequent lines. Note that each name must be specified on a separate line using either an absolute path or using a path relative to the working directory (from which T3d is executed).
The first record of the background mesh file describing a mesh contains the mesh type (3 - triangular/tetrahedral mesh, 4 - quadrilateral/hexahedral mesh, 7 - mixed mesh), element degree (1 - linear, 2 - quadratic, 3 - quadratic bubble), and optionally also the specification of the range of values (1 - positive values only, 0 - non-negative values only, -1 - all real values) the mesh size at individual nodes of the background mesh can attain. If the range specification is not provided, default zero value is used.

mesh_type elem_degree range_spec

The second record of the background mesh file then contains the overall numbers of background nodes and elements in one of the following formats according to the mesh type.

- triangular/tetrahedral mesh (mesh type = 3)

  bg_nodes bg_edges bg_trias bg_tetras

- quadrilateral/hexahedral mesh (mesh type = 4)

  bg_nodes bg_edges bg_quads bg_hexas

- mixed mesh (mesh type = 7)

  bg_nodes bg_edges bg_trias bg_quads bg_tetras bg_pyrams bg_wedges bg_hexas

For each background node, the following record describing the position of the node and associated mesh size is expected. The zero or negative mesh size is treated (if allowed by the range specification) as size corresponding to the size of model extent.

  node_id coord_x coord_y coord_z mesh_size

Background edges, triangles, quadrilaterals, tetrahedra, pyramids, wedges, and hexahedra are described by the following records

edge_id	*nd_ids(1 – 2)*	*[mid_nd_ids(?)]*
tria_id	*nd_ids(1 – 3)*	*[mid_nd_ids(?)]*
quad_id	*nd_ids(1 – 4)*	*[mid_nd_ids(?)]*
tetra_id	*nd_ids(1 – 4)*	*[mid_nd_ids(?)]*
pyram_id	*nd_ids(1 – 5)*	*[mid_nd_ids(?)]*
wedge_id	*nd_ids(1 – 6)*	*[mid_nd_ids(?)]*
hexa_id	*nd_ids(1 – 8)*	*[mid_nd_ids(?)]*

which are repeated *bg_edges*, *bg_trias*, *bg_quads*, *bg_tetras*, *bg_pyrams*, *bg_wedges*, and *bg_hexas* times. The mid-nodes related to quadratic and quadratic bubble elements are ignored. Background nodes as well as backround elements of each type must be numbered continuously starting from 1, but within the block of nodes or elements of the same type, they can be ordered arbitrarily. However, the ordering nodes – edges – trias – quads – tetras – pyrams – wedges – hexas is obligatory.

7 Mesh Output Format

The output of the mesh data is formatted in such a way that it can be read by another application either in a fixed or free format. There are no keywords in the mesh output format (the keywords used in this section are just for better understanding). The output is organized in blocks separated by a free line (if there is no output in a particular block the corresponding free line is omitted as well). The corresponding fixed format in Fortran style is enclosed in parenthesis at the end of the same line. If the description of a particular record of output format is too long it is split to several lines. In that case the corresponding format in Fortran style is split equivalently. The actual number of items on output is provided in parenthesis if necessary. Since the actual output is controlled by a command line option (see *–p* option in Section 8), items of the output format which are dependent on that option are enclosed in brackets. If some of them are not requested for the output the corresponding part of the format should be skipped.

Note that some features of the output are enabled or disabled if the source code is compiled with certain directives. The output of neighbouring elements is supported only if the code has been compiled with the *T3D_OUTPUT_NEIGHBOUR* directive.

Each mesh entity is always classified to the particular model entity of the lowest possible dimension (irrespectably whether this entity is hidden or not). The integer types are assigned to individual model entities (including interfaces) as follows

- entity type 1 = ***vertex***
- entity type 2 = ***curve***
- entity type 3 = ***surface***
- entity type 4 = ***region***
- entity type 5 = ***patch***
- entity type 6 = ***shell***
- entity type 7 = ***interface***

Block No 1 contains mesh type (3 - triangular/tetrahedral mesh, 4 - quadrilateral/hexahedral mesh, 7 - mixed mesh), element degree (1 - linear, 2 - quadratic), applied renumbering type (0 - none renumbering, 1 - node renumbering, 2 - element renumbering), and output type (see *–p* option in Section 8) followed by the number of nodes and elements in one of the following formats according to the mesh type.

- triangular/tetrahedral mesh (mesh type = 3)

mesh_type elem_degree renum_type output_type	(4I10)
nodes edges trias tetras	(4I10)

- quadrilateral/hexahedral mesh (mesh type = 4)

mesh_type elem_degree renum_type output_type (4I10)
nodes edges quads hexas (4I10)

- mixed mesh (mesh type = 7)

mesh_type elem_degree renum_type output_type (4I10)
nodes edges trias quads tetras pyrams wedges hexas (8I10)

Block No 2 consists of records for each node in one of the following formats depending on the node primary classification to the parent model entity of the lowest possible dimension

- vertex (entity type = 1)

node_id coords(3) ent_type ent_id ent_prop (I10, 3F15.6, I5, I10, I10,
[sec_clasfs [ent_type ent_id ent_prop]*(sec_clasfs)]* [I5, (sec_clasfs)[I5, I10, I10]])

- curve (entity type = 2)

node_id coords(3) ent_type ent_id ent_prop (I10, 3F15.6, I5, I10, I10,
[sec_clasfs [ent_type ent_id ent_prop]*(sec_clasfs)]* [I5, (sec_clasfs)[I5, I10, I10]]
[tangent(3)] [par_coords(1)] [5X, 3F10.6], [5X, F10.6])

- surface (entity type = 3)

node_id coords(3) ent_type ent_id ent_prop (I10, 3F15.6, I5, I10, I10,
[sec_clasfs [ent_type ent_id ent_prop]*(sec_clasfs)]* [I5, (sec_clasfs)[I5, I10, I10]]
[normal(3)] [par_coords(2)] [5X, 3F10.6], [5X, 2F10.6])

- region (entity type = 4)

node_id coords(3) ent_type ent_id ent_prop (I10, 3F15.6, I5, I10, I10,
[sec_clasfs [ent_type ent_id ent_prop]*(sec_clasfs)]* [I5, (sec_clasfs)[I5, I10, I10]]

- patch (entity type = 5)

node_id coords(3) ent_type ent_id ent_prop (I10, 3F15.6, I5, I10, I10,
[sec_clasfs [ent_type ent_id ent_prop]*(sec_clasfs)]* [I5, (sec_clasfs)[I5, I10, I10]]
[normal(3)] [par_coords(2)] [5X, 3F10.6], [5X, 2F10.6])

- shell (entity type = 6)

node_id coords(3) ent_type ent_id ent_prop (I10, 3F15.6, I5, I10, I10,
[sec_clasfs [ent_type ent_id ent_prop]*(sec_clasfs)]* [I5, (sec_clasfs)[I5, I10, I10]]
[normal(3)] [par_coords(2)] [5X, 3F10.6], [5X, 2F10.6])

If output of nodal parameter(s), tangent, or normal is requested for quadratic boundary non-conforming mesh, the parameter(s), tangent, or normal at mid-edge nodes are computed as if boundary conforming mesh would have been requested. Note that output of nodal parameter(s), tangent, or normal for quadratic bubble boundary non-conforming mesh is ignored in the current version.
Also note that nodes classified to a vertex or curve are always classified to the most top physical parent vertex or curve, respectively. This also implies that the *nodeprop*, *elemprop*, and *elemgroup* specifications described in Section 3 are relevant at the most top physical parent model entity. Except for the compulsory primary classification, the optional secondary classification to model entities (excluding interface entity) of dimension higher than that of the primary classification may be provided. Note that there is always zero number of secondary classifications for a region node. The parametric coordinates of a node on a patch refer to a fictitious rectangular bilinear planar surface not defined in the input model (and are provided just for completeness). The secondary classification, tangent, normal, and parametric coordinates in block No 2 are provided only if appropriate output specification (*–p* option) has been used. In the current implementation, there are no nodes classified to interface.

Block No 3 consists of records for each edge in one of the following formats depending on the element degree (linear or quadratic or quadratic bubble). Entity type is equal to 2 (curve) or 7 (interface).

- linear edge

  edge_id node_id(2) ent_type ent_id ent_prop (I10, 2I10, I5, I10, I10)

- quadratic and quadratic bubble edge

  edge_id node_id(3) ent_type ent_id ent_prop (I10, 3I10, I5, I10, I10)

Note that edges classified to a curve are always classified to the most top physical parent curve. The node numbering of an edge element is provided in Figure 1. In the case of quadratic and quadratic bubble interface edge, there is no mid-edge node. Therefore the third node has always zero id.

Block No 4 (not present for mesh type 4) consists of records for each triangle in one of the following formats depending on the element degree (linear or quadratic or quadratic bubble). the entity type is equal to 3, 5, or 6, according to the classification of the triangle to a surface, patch, or shell, respectively.

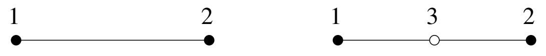

Figure 1: Node numbering of an edge element (linear element – left, quadratic and quadratic bubble element – right). Symbols: filled circle – corner node, empty circle – mid-edge node

- linear triangle

  tria_id node_id(3) ent_type ent_id ent_prop (I10, 3I10, I5, I10, I10,
  [iso_type] [ngb_elem_id(3)] [I5], [3I10]
  [bnd_curve_id(3) bnd_curve_prop(3)] [6I10])

- quadratic triangle

  tria_id node_id(6) ent_type ent_id ent_prop (I10, 6I10, I5, I10, I10,
  [iso_type] [ngb_elem_id(3)] [I5], [3I10]
  [bnd_curve_id(3) bnd_curve_prop(3)] [6I10])

- quadratic bubble triangle

  tria_id node_id(7) ent_type ent_id ent_prop (I10, 6I10, I5, I10, I10,
  [iso_type] [ngb_elem_id(3)] [I5], [3I10]
  [bnd_curve_id(3) bnd_curve_prop(3)] [6I10])

The bounding curve ids (zero means no boundary curve) and property ids are provided to simplify the boundary condition specification. The node and edge numbering of a triangular element is depicted in Figure 2. Note that the numbering of neighbouring elements is in agreement with the edge numbering. The iso type refers to the type of the element in terms of its size, shape, and orientation. Positive value stands for particular type of iso element, zero value is used for variable-sized elements. The iso type numbering is generally non-continuous and is unique only for elements classified to the same patch. Thus elements of the same iso type classified to the same patch are exactly of the same size, shape, and orientation.

Block No 5 (not present for mesh type 3) consists of records for each quadrilateral in one of the following formats depending on the element degree (linear or quadratic or quadratic bubble). the entity type is equal to 3, 5, 6, or 7 according to the classification of the quadrilateral to a surface, patch, shell, or interface, respectively.

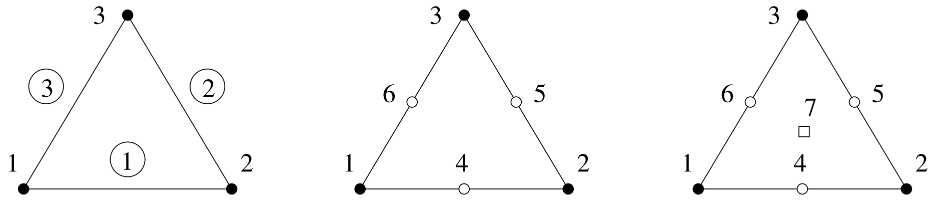

Figure 2: Node and edge numbering of a triangular element (linear element – left, quadratic element – middle, quadratic bubble element – right). Symbols: filled circle – corner node, empty circle – mid-edge node, empty square – mid-face node

- linear quadrilateral

  quad_id node_id(4) ent_type ent_id ent_prop (I10, 4I10, I5, I10, I10,
  [iso_type] [ngb_elem_id(4)] [I5], [4I10]
  [bnd_curve_id(4) bnd_curve_prop(4)] [8I10])

- quadratic quadrilateral

  quad_id node_id(8) ent_type ent_id ent_prop (I10, 8I10, I5, I10, I10,
  [iso_type] [ngb_elem_id(4)] [I5], [4I10]
  [bnd_curve_id(4) bnd_curve_prop(4)] [8I10])

- quadratic bubble quadrilateral

  quad_id node_id(9) ent_type ent_id ent_prop (I10, 8I10, I5, I10, I10,
  [iso_type] [ngb_elem_id(4)] [I5], [4I10]
  [bnd_curve_id(4) bnd_curve_prop(4)] [8I10])

The bounding curve ids (zero means no boundary curve) and property ids are provided to simplify the boundary condition specification. The node and edge numbering of a quadrilateral element is depicted in Figure 3. Note that the numbering of neighbouring elements is in agreement with the edge numbering. The iso type refers to the type of the element in terms of its shape and orientation. Non-zero value stands for particular type of iso element, zero value is used for variable-sized elements. The iso type numbering is generally non-continuous and is unique only in the frame of parent patch entity.

For the interface quadrilateral element, the first edge corresponds to the interface master curve and the third edge to the interface slave curve. Only the elements classified to the same interface are considered as neighbouring elements. Thus there are never neighbouring elements adjacent to the first and the third edge of the interface quadrilateral element. Moreover, there are also no neighbouring elements over those remaining edges that are not shared just by two quadrilateral elements classified to the same interface. Since the interface is bounded only in one direction (by opposite curves), there is no curve corresponding to the boundary in the other direction. Therefore boundary curve id, corresponding to those from the second and last edge of the interface quadrilateral element over which there is no

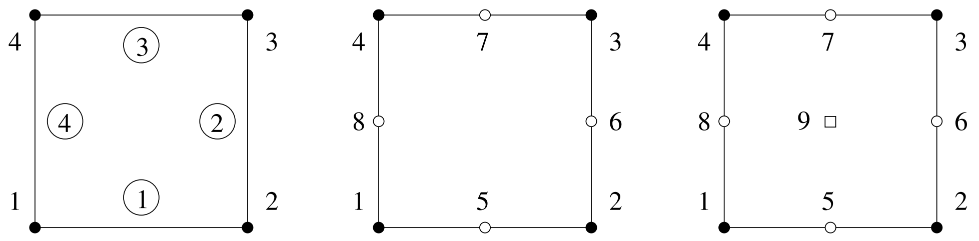

Figure 3: Node and edge numbering of a quadrilateral element (linear element – left, quadratic element – middle, quadratic bubble element – right). Symbols: filled circle – corner node, empty circle – mid-edge node, empty square – mid-face node

neighbouring element classified to the same interface, is set to -1 representing a fictitious interface boundary. In the case of the quadratic interface quadrilateral element, there are no mid-edge nodes on edges connecting the master and slave curve. Moreover, in the case of quadratic bubble interface quadrilateral element, there is also no mid-face node. Therefore zero ids are used for mid-edge nodes corresponding to the second and last edge of the quadratic and quadratic bubble interface quadrilateral element and to mid-face node of quadratic bubble interface quadrilateral element.

Block No 6 (not present for mesh type 4) consists of records for each tetrahedron in one of the following formats depending on the element degree (linear or quadratic or quadratic bubble). Entity type is equal to 4 (region).

- linear tetrahedron

tetra_id node_id(4) ent_type ent_id ent_prop	(I10, 4I10, I5, I10, I10,
[ngb_elem_id(4)]	[4I10]
[bnd_ent_id(4) bnd_ent_type(4) bnd_ent_prop(4)]	[4I10, 4I5, 4I10])
[bnd_curve_id(6) bnd_curve_prop(6)]	[6I10, 6I10])

- quadratic tetrahedron

tetra_id node_id(10) ent_type ent_id ent_prop	(I10, 10I10, I5, I10, I10,
[ngb_elem_id(4)]	[4I10]
[bnd_ent_id(4) bnd_ent_type(4) bnd_ent_prop(4)]	[4I10, 4I5, 4I10)
[bnd_curve_id(6) bnd_curve_prop(6)]	[6I10, 6I10)

- quadratic bubble tetrahedron

tetra_id node_id(15) ent_type ent_id ent_prop	(I10, 10I10, I5, I10, I10,
[ngb_elem_id(4)]	[4I10]
[bnd_ent_id(4) bnd_ent_type(4) bnd_ent_prop(4)]	[4I10, 4I5, 4I10)
[bnd_curve_id(6) bnd_curve_prop(6)]	[6I10, 6I10)

The bounding model entity ids and property ids are provided to simplify the boundary condition specification. If there is no boundary model entity/curve on a particular face/edge of a tetrahedron then corresponding boundary entity/curve id and boundary entity type are equal to zero. The node and face numbering of a tetrahedral element is displayed in Figure 4. The edge numbering starts from 1 and its ordering follows the ordering of mid-edge nodes. The numbering of neighbouring elements corresponds to the face numbering.

Block No 7 (not present for mesh type 3 or 4) consists of records for each pyramid in one of the following formats depending on the element degree (linear or quadratic or quadratic bubble). Entity type is equal to 4 (region).

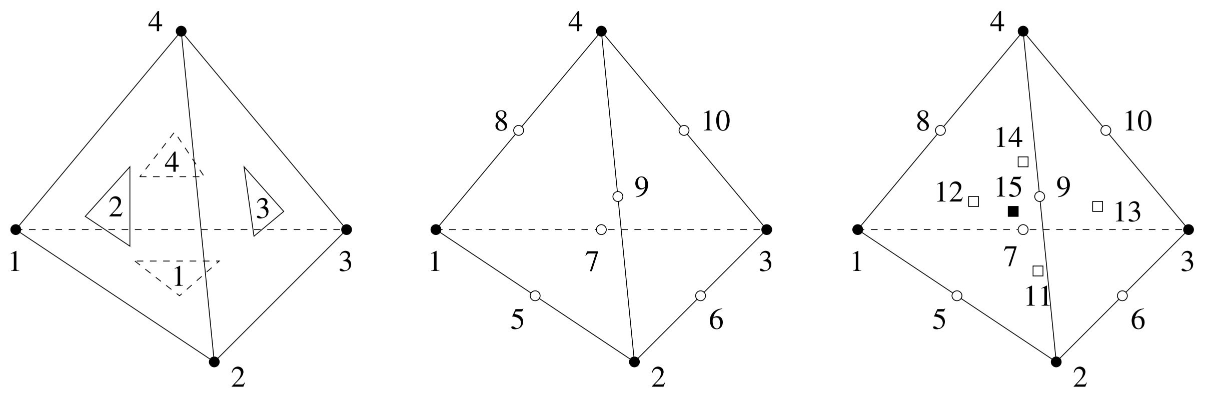

Figure 4: Node and face numbering of a tetrahedral element (linear element – left, quadratic element – middle, quadratic bubble element – right). Symbols: filled circle – corner node, empty circle – mid-edge node, empty square – mid-face node (ordering according face numbers), filled square – mid-cell node

- linear pyramid

pyram_id node_id(5) ent_type ent_id ent_prop	(I10, 5I10, I5, I10, I10,
[ngb_elem_id(5)]	[5I10]
[bnd_ent_id(5) bnd_ent_type(5) bnd_ent_prop(5)]	[5I10, 5I5, 5I10])
[bnd_curve_id(8) bnd_curve_prop(8)]	[8I10, 8I10)

- quadratic pyramid

pyram_id node_id(13) ent_type ent_id ent_prop	(I10, 13I10, I5, I10, I10,
[ngb_elem_id(5)]	[5I10]
[bnd_ent_id(5) bnd_ent_type(5) bnd_ent_prop(5)]	[5I10, 5I5, 5I10])
[bnd_curve_id(8) bnd_curve_prop(8)]	[8I10, 8I10)

- quadratic bubble pyramid

pyram_id node_id(19) ent_type ent_id ent_prop	(I10, 13I10, I5, I10, I10,
[ngb_elem_id(5)]	[5I10]
[bnd_ent_id(5) bnd_ent_type(5) bnd_ent_prop(5)]	[5I10, 5I5, 5I10])
[bnd_curve_id(8) bnd_curve_prop(8)]	[8I10, 8I10)

The bounding model entity ids and property ids are provided to simplify the boundary condition specification. If there is no boundary model entity on a particular face of a pyramid then corresponding boundary entity id and boundary entity type are equal to zero. The node and face numbering of a pyramidal element is displayed in Figure 5. The edge numbering starts from 1 and its ordering follows the ordering of mid-edge nodes. The numbering of neighbouring elements corresponds to the face numbering.

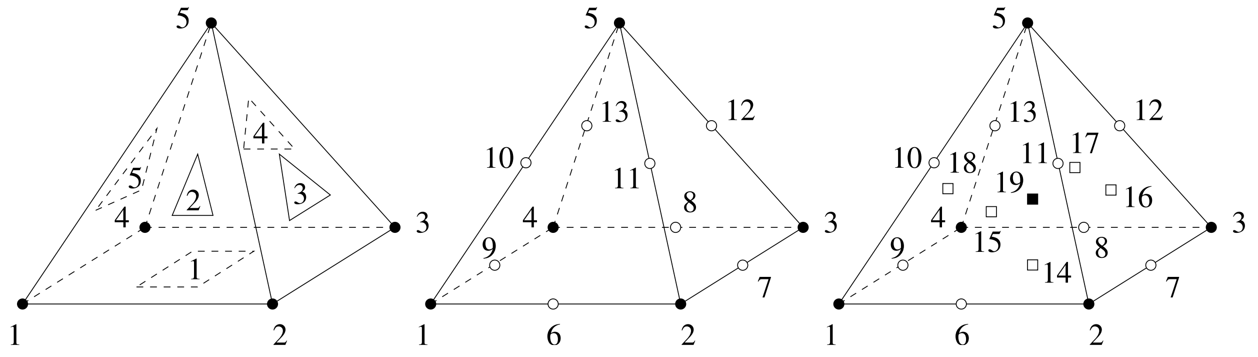

Figure 5: Node and face numbering of a pyramid element (linear element – left, quadratic element – middle, quadratic bubble element – right). Symbols: filled circle – corner node, empty circle – mid-edge node, empty square – mid-face node (ordering according face numbers), filled square – mid-cell node

Block No 8 (not present for mesh type 3 or 4) consists of records for each wedge (triangular prism) in one of the following formats depending the element degree (linear or quadratic or quadratic bubble). Entity type is equal to 4 (region) or 7 (interface).

- linear wedge

wedge_id node_id(6) ent_type ent_id ent_prop	(I10, 6I10, I5, I10, I10,
[ngb_elem_id(5)]	[5I10]
[bnd_ent_id(5) bnd_ent_type(5) bnd_ent_prop(5)]	[5I10, 5I5, 5I10])
[bnd_curve_id(9) bnd_curve_prop(9)]	[9I10, 9I10)

- quadratic wedge

wedge_id node_id(15) ent_type ent_id ent_prop	(I10, 15I10, I5, I10, I10,
[ngb_elem_id(5)]	[5I10]
[bnd_ent_id(5) bnd_ent_type(5) bnd_ent_prop(5)]	[5I10, 5I5, 5I10])
[bnd_curve_id(9) bnd_curve_prop(9)]	[9I10, 9I10)

- quadratic bubble wedge

wedge_id node_id(21) ent_type ent_id ent_prop	(I10, 15I10, I5, I10, I10,
[ngb_elem_id(5)]	[5I10]
[bnd_ent_id(5) bnd_ent_type(5) bnd_ent_prop(5)]	[5I10, 5I5, 5I10])
[bnd_curve_id(9) bnd_curve_prop(9)]	[9I10, 9I10)

The bounding model entity ids and property ids are provided to simplify the boundary condition specification. If there is no boundary model entity on a particular face of a wedge then corresponding boundary entity id and boundary entity type are equal to zero. The node and face numbering of a wedge element is displayed in Figure 6. The edge numbering starts from 1 and its ordering follows the ordering of mid-edge nodes. The numbering of neighbouring elements corresponds to the face numbering.

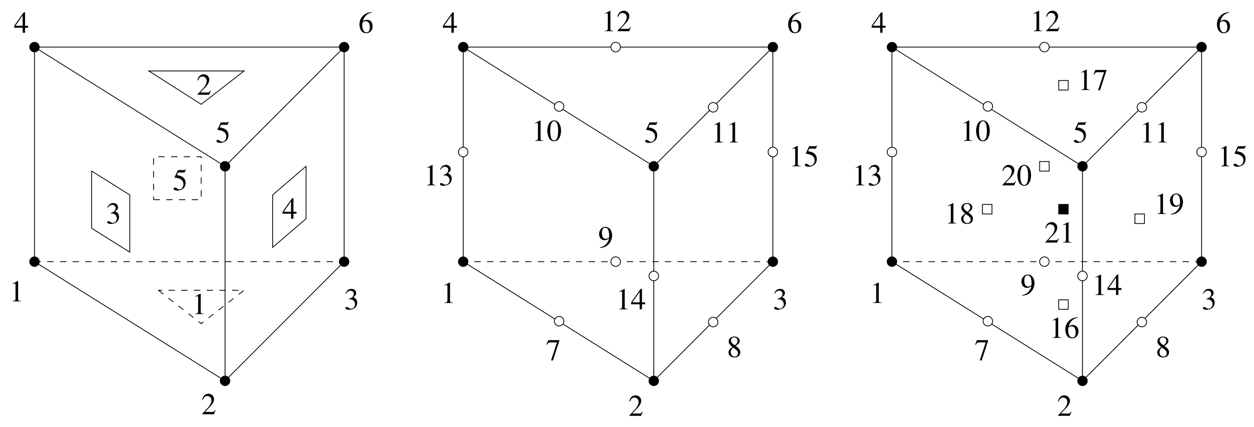

Figure 6: Node and face numbering of a wedge element (linear element – left, quadratic element – middle, quadratic bubble element – right). Symbols: filled circle – corner node, empty circle – mid-edge node, empty square – mid-face node (ordering according face numbers), filled square – mid-cell node

For the interface wedge element, the first face corresponds to the interface master entity and the second face to the interface slave entity. Only the elements classified to the same interface are considered as neighbouring elements. Thus there are never neighbouring elements adjacent to the triangular faces of the interface wedge element. Moreover, there are also no neighbouring elements over those quadrilateral faces that are not shared by two elements classified to the same interface. Since the interface is bounded only in one direction (by opposite surfaces, patches, or shells), there are no entities corresponding to the boundary in the other directions. Therefore boundary entity id, corresponding to those from the quadrilateral faces of the interface wedge element over which there is no neighbouring element classified to the same interface, is set to -1 representing a fictitious interface boundary of zero type. Note that -1 is also used as curve id for relevant interface edges bounding quadrilateral face with that fictitious interface boundary. In the case of the quadratic interface wedge element, there are no mid-edge nodes on edges connecting the master and slave entity. Moreover, in the case of quadratic bubble interface wedge element, there are also no mid-face nodes on faces connecting master and slave entity and no mid-cell node. Therefore zero ids are used for the last three mid-edge nodes of the quadratic interface wedge element and also for the last three mid-face nodes and for the mid-cell node of the quadratic bubble wedge element.

Block No 9 (not present for mesh type 3) consists of records for each hexahedron in one of the following formats depending on the element degree (linear or quadratic or quadratic bubble). Entity type is equal to 4 (region) or 7 (interface).

- linear hexahedron

hexa_id node_id(8) ent_type ent_id ent_prop	(I10, 8I10, I5, I10, I10,
[iso_type] [ngb_elem_id(6)]	[I5], [6I10]
[bnd_ent_id(6) bnd_ent_type(6) bnd_ent_prop(6)]	[6I10, 6I5, 6I10])
[bnd_curve_id(12) bnd_curve_prop(12)]	[12I10, 12I10)

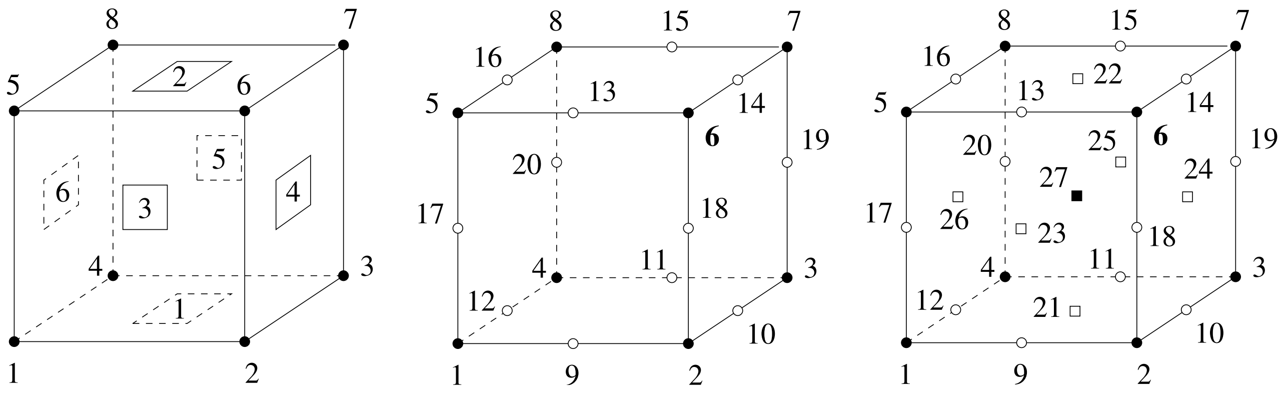

Figure 7: Node and face numbering of a hexahedral element (linear element – left, quadratic element – middle, quadratic bubble element – right). Symbols: filled circle – corner node, empty circle – mid-edge node, empty square – mid-face node (ordering according face numbers), filled square – mid-cell node

- quadratic hexahedron

hexa_id node_id(20) ent_type ent_id ent_prop	(I10, 20I10, I5, I10, I10,
[iso_type] [ngb_elem_id(6)]	[I5], [6I10]
[bnd_ent_id(6) bnd_ent_type(6) bnd_ent_prop(6)]	[6I10, 6I5, 6I10])
[bnd_curve_id(12) bnd_curve_prop(12)]	[12I10, 12I10)

- quadratic bubble hexahedron

hexa_id node_id(27) ent_type ent_id ent_prop	(I10, 20I10, I5, I10, I10,
[iso_type] [ngb_elem_id(6)]	[I5], [6I10]
[bnd_ent_id(6) bnd_ent_type(6) bnd_ent_prop(6)]	[6I10, 6I5, 6I10])
[bnd_curve_id(12) bnd_curve_prop(12)]	[12I10, 12I10)

The bounding model entity ids and property ids are provided to simplify the boundary condition specification. If there is no boundary model entity on a particular face of a hexahedron then corresponding boundary entity id and boundary entity type are equal to zero. The node and face numbering of a hexahedral element is displayed in Figure 7. The edge numbering starts from 1 and its ordering follows the ordering of mid-edge nodes. The numbering of neighbouring elements corresponds to the face numbering. The iso type refers to the type of the element in terms of its size, shape, and orientation. Positive value stands for particular type of iso element, zero value is used for variable-sized elements. The iso type numbering is generally non-continuous and is unique only for elements classified to the same region. Thus elements of the same iso type classified to the same region are exactly of the same size, shape, and orientation.

For the interface hexahedral element, the first face corresponds to the interface master entity and the second face to the interface slave entity. Only the elements classified to the same interface are considered as neighbouring elements. Thus there are never neighbouring elements adjacent to the first and second faces of the interface hexahedral element. Moreover, there are also no neighbouring elements over those remaining faces that are not shared by two elements classified to the same interface. Since the interface is bounded only in one direction (by opposite surfaces, patches, or shells), there are no entities corresponding to the boundary in the other directions. Therefore boundary entity id, corresponding to those from the last four faces of the interface hexahedral element over which there is no neighbouring element classified to the same interface, is set to -1 representing a fictitious interface boundary of zero type. Note that -1 is also used as curve id for relevant interface edges bounding quadrilateral face with that fictitious interface boundary. In the case of the quadratic interface hexahedral element, there are no mid-edge nodes on edges connecting the master and slave entity. Moreover, in the case of quadratic bubble interface hexahedral element, there are also no mid-face nodes on faces connecting master and slave entity and no mid-cell node. Therefore zero ids are used for the last four mid-edge nodes of the quadratic interface hexahedral element and also for the last four mid-face nodes and for the mid-cell node of the quadratic bubble hexahedral element.

Note that there is only one global element numbering (it means that different types of elements are not numbered independently). The output of neighbouring elements and boundary entities in blocks No 4, 5, 6, 7, 8, and 9 are provided only if appropriate output specification (*–p* option) has been used. Similarly, the output of iso type for elements in blocks 4, 5, and 9 is controlled by output specification (*–p* option). Also note that the output of the following blocks (blocks No. 10, 11, 12, 13, and 14) is dependent on the *–p* option in the similar way as was the case for the previous blocks. The actual form of the output may be checked by executing the program with *–Fout* option and appropriate *–p* option.

Block No 10 consists of the number of curves, surfaces, patches, and shells for which the list of associated elements is provided.

[curves surfaces patches shells] ([4I10])

Block No 11 contains curve entity type (equal to 2), curve id, curve property, and number of 1D elements (edges) on the curve. This first record is followed by records (for each of the curve edges) containing number of 2D elements (triangles and quadrilaterals) connected to the edge and list of ids of these connected 2D elements. The order in which these records appear on the output is identical with the edge ordering if the curve would have been designated for output. The order of connected elements is not defined. Note that interface 2D elements are not considered. If the parent entity of the connected 2D element has not been designated for the output the element does not appear in the list of connected elements. If neither from the 2D model entities connected to the curve is designated for the output, the curve does not appear in this block.

[curve_ent_type curve_id curve_prop elems] ([4I10])
[con_elems con_elems_ids(con_elems)] ([I10,(con_elems)I10])

Block No 11 is repeated for all physical curves for which the output of associated elements has not been disabled on the input (see keyword *bassoc* in Section 3.2) or during run-time and the total number of which is given in block No 6.

Block No 12 contains surface entity type (equal to 3), surface id, surface property, and number of 2D elements (triangles and quadrilaterals) on the surface. This first record is followed by records (for each of the surface elements) containing number of 3D elements (tetrahedra, pyramids, wedges, and hexahedra) connected to the surface 2D element and list of ids of these connected 3D elements. The order in which these records appear on the output is identical with the 2D element ordering if the surface would have been designated for output. The order of connected 3D elements is not defined. Note that interface 3D elements are not considered. If the region of the connected 3D element has not been designated for the output the element does not appear in the list of connected elements. If neither from the regions bounded by the surface is designated for the output, the surface does not appear in this block.

[surface_ent_type surface_id surface_prop elems] ([4I10])
[con_elems con_elems_ids(con_elems)] ([I10,(con_elems)I10])

Block No 12 is repeated for all physical solid surfaces for which the output of associated elements has not been disabled on the input (see keyword *bassoc* in Section 3.3) or during run-time and the total number of which is given in block No 6.

Block No 13 contains patch entity type (equal to 5), patch id, patch property, and number of 2D elements (triangles and quadrilaterals) on the patch. This first record is followed by records (for each of the patch elements) containing number of 3D elements (tetrahedra, pyramids, wedges, and hexahedra) connected to the patch 2D element and list of ids of these connected 3D elements. The order in which these records appear on the output is identical with the 2D element ordering if the patch would have been designated for output. The order of connected 3D elements is not defined. Note that interface 3D elements are not considered. If the region of the connected 3D element has not been designated for the output the element does not appear in the list of connected elements. If neither from the regions bounded by the patch is designated for the output, the patch does not appear in this block.

[patch_ent_type patch_id patch_prop elems] ([4I10])
[con_elems con_elems_ids(con_elems)] ([I10,(con_elems)I10])

Block No 13 is repeated for all physical solid patches for which the output of associated elements has not been disabled on the input (see keyword *bassoc* in Section 3.4) or during run-time and the total number of which is given in block No 6.

Block No 14 contains shell entity type (equal to 6), shell id, shell property, and number of 2D elements (triangles and quadrilaterals) on the shell. This first record is followed by records (for each of the shell elements) containing number of 3D elements (tetrahedra, pyramids, wedges, and hexahedra) connected to the shell 2D element and list of ids of these connected 3D elements. The order in which these records appear on the output is identical with the 2D element ordering if the shell would have been designated for output. The order of connected 3D elements is not defined. Note that interface 3D elements are not considered. If the region of the connected 3D element has not been designated for the output the element does not appear in the list of connected elements. If neither from the regions bounded by the shell is designated for the output, the shell does not appear in this block.

[shell_ent_type shell_id shell_prop elems] ([4I10])
[con_elems con_elems_ids(con_elems)] ([I10,(con_elems)I10])

Block No 14 is repeated for all physical solid shells for which the output of associated elements has not been disabled on the input (see keyword *bassoc* in Section 3.5) or during run-time and the total number of which is given in block No 6.

7.1 Binary Mesh Output

Provided the code has been compiled with the *T3D_BINARY_SUPPORT* directive, the mesh output can be also performed in binary regime. This is enforced by setting properly the output specification (see *–p* command line option). The binary output contains all the data present in the formatted output. In C terminology, all integer values are saved as *long* values excepts for values of entity type which are saved as *short* values. All floating point numbers are saved as *double* values. On 32-bit architecture the types *long*, *short*, and *double* correspond to 4-, 2-, and 8-Byte representation, respectively. Note that the binary output is generally platform dependent and therefore it is not portable.

8 Command Line Options

The program may be executed by typing

t3d [option [parameter]] . . .

on the command line prompt with the following command line options:[1]

–h - displays program usage description

–P - displays program command line parameters description

–F - displays format of input file, output file, and background mesh file;
note that the description of output file format takes into account the actually applied *–p* and *–k* options; the other options are ignored; alternatively options *–Fin*, *–Fout*, and *–Fbgm* may be used do display format of the selected files only; if the code has been compiled with the *T3D_BND_MESH* directive, option *–Fbnd* can be used to display format of boundary mesh file

–Q - silent mode (all messages to standard output and standard error channels are discarded); alternatively options *–Qout*, *–Qerr* may be used do discard messages to selected standard channel only; note that this option does not affect output to log file

–W - suppresses warning messages;
note that element quality warnings may be suppressed by compiling the code without the *T3D_QUALITY_WARNING* directive

–V - includes virtual entities into the domain extent

–C - enables curvature violation on surface and shell; (triangles and quadrilaterals with node normals differing by more than 120° are allowed);
this option is effective only if the code has been compiled with
the *T3D_VIOLATION* directive

–B - generates boundary conforming elements;
this option is relevant only for quadratic and quadratic bubble elements (see option *–k*)

–A - disables default designation of boundary model entities (curves, surfaces, patches, and shells) for output of boundary associated elements;
this option is relevant only if output of boundary associated elements is required (see option *–p*)

–S - disables nodal smoothing for surface, patch, shell, and region triangulation;
note that this option prevents removing slivers from region triangulation; this options also prevents handling nonplanar patches (an error occurs)

–H - disables support for quadrilateral and hexahedral meshing;
note that this options prevents discretization of interfaces

[1]The arguments of the parameterized options are specified symbolically in parenthesis behind the option (terms $_{\mathrm{fp}}\#$, $_{\mathrm{in}}\#$, and *string* stand for floating point number, for integer number, and for text string respectively).

–K - enables convexity check for mapped quadrilateral elements;
this option works only without option *–H* being used

–i - (string) input model file name;
if this option is not present, an error occurs unless option *–b* is used or unless the code has been compiled with the *T3D_FUNCTION_INPUT* directive and called as a subroutine (see README for details)

–o - (string) output mesh file name;
output of mesh is realized only if this option is used unless the code has been compiled with the *T3D_FUNCTION_OUTPUT* directive and called as a subroutine (see README for details)

–g - (string) output log file name;
by default no log file is produced; note that *–Q* option does not affect output to log file

–m - (string) input background mesh file name

–b - (string) binary model file name;
when being used together with option *–i* or if the code has been compiled with the *T3D_FUNCTION_INPUT* directive, binary output is performed (irrespectively, whether *T3D_FUNCTION_OUTPUT* directive is defined or not) otherwise binary input is performed;
note that binary input is successful only if the binary (output) file was created by compatible executable (on the same platform);
this options is enabled only if the code has been compiled with
the *T3D_BINARY_SUPPORT* directive

–x - (string) output graphics (Elixir) file name

–n - ($7\times$ $_{\rm in}$#) number of model entities and interfaces;
this option specifies the estimate on number of individual entities and interfaces in model representation (including virtual ones); the entities are ordered: vertices, curves, surfaces, patches, shell, regions, and interfaces; must not be negative; by default zero number of model entities of each type is assumed; if applied no model entity of corresponding type is allowed to have larger id than the specified value; need not be applied to all types of model entities

–f - ($2\times$ $_{\rm in}$#) numbering shift;
this option specifies the shift for node and element numbering; must not be negative; no numbering shift is applied by default

–L - ($_{\rm in}$#) default listing size;
this option defines the length of listing which is allocated as a contiguous block of memory; ranges from 10 to 10000; default value is 1000

–T - ($_{\rm in}$#) hash table size;
this option defines the size of hash table used to access model entities; the larger value the faster access but larger memory demands; must be positive; default value is 100

–r - ($_{\rm in}$#) renumbering type;
some cumulative combinations of the following specifications may be used

0 - no renumbering (default)
1 - renumbering of nodes
2 - renumbering of elements (disabled)
4 - fast renumbering

specification 4 is relevant only if used cumulatively with specification 1 or 2; it attempts to shorten the renumbering time by skipping passes the are not likely to produce the best renumbering; renumbering is disabled for quadratic bubble elements

–s - ($_{\text{in}}$#) type of weighting applied during the Laplacian smoothing;
this option is relevant only if option *–S* is not used; any cumulative combination of the following types may be used

0 - no weighting (default)
1 - mesh size weighting
2 - connectivity weighting

–k - ($_{\text{in}}$#) element degree;
one of the following types must be used

1 - linear (default)
2 - quadratic
3 - quadratic bubble (quadratic with mid-face and mid-cell nodes)

note that mid-edge nodes on quadratic and quadratic bubble elements and mid-face nodes on quadratic bubble elements do not follow possible curvature of model boundary unless the *–B* option is specified; quadratic bubble elements are disabled unless the code has been compiled with the *T3D_BUBBLE_ELEMENT* directive; use of quadratic bubble elements does not allow mesh renumbering

–j - ($_{\text{in}}$#) limit on the number of generated nodes;
no limit (zero value) by default

–p - ($_{\text{in}}$#) additional output specification;
any cumulative combination of the following specifications may be used

0 - no additional output (default)
1 - output of parametric coordinate(s)
(only nodes classified to a curve, surface, patch, or shell)
note that this specification is ignored if generation of quadratic bubble elements (see option -*k*) is required without request to generate boundary conforming elements (see option -*B*)
2 - output of node tangent
(only nodes classified to a curve)
note that this specification is ignored if generation of quadratic bubble elements (see option -*k*) is required without request to generate boundary conforming elements (see option -*B*)
4 - output of node normal
(only nodes classified to a surface, patch, or shell)
note that this specification is ignored if generation of quadratic bubble elements (see option -*k*) is required without request to generate boundary conforming elements (see option -*B*)

8 - output of boundary model entities
(curves to which edges of 2D elements are classified and surfaces, patches, or shells to which faces of 3D elements are classified)

16 - output of elements associated with boundary model entities
(2D elements associated with a curve and 3D elements associated with a surface, patch, or shell)

32 - output of neighbouring elements
this specification is enabled only if the code has been compiled with the *T3D_OUTPUT_NEIGHBOUR* directive

64 - output of element iso type
types are numbered from zero separately for each relevant model entity (surface, patch, shell); zero type stands for variable-sized element

128 - output of nodes in T3d native order
native ordering follows the mesh generation order: vertex, curve, surface, patch, shell, region corner (linear) nodes, curve, surface, patch, shell, region mid-edge (quadratic) nodes, surface, patch, shell mid-face (bubble) nodes, region mid-face (bubble) nodes, region mid-cell (bubble) nodes; this specification is relevant only if used together with node renumbering request (see option *–r*)

256 - output of node secondary classification(s)
(classification to model entities of higher dimension than that of primary classification)

512 - complete output of boundary model entities
(curves to which edges of 2D and 3D elements are classified and surfaces, patches, or shells to which faces of 3D elements are classified)
note that this specification is a generalized version of specification 8; therefore specification 8 is automatically activated when this specification is applied

1024 - mesh output even if zero element quality detected

2048 - perform binary output;
this specification is ignored if the code has been compiled with *T3D_FUNCTION_OUTPUT* directive and called as a subroutine (see README for details); this specification is enabled only if the code has been compiled with the *T3D_BINARY_SUPPORT* directive

4096 - simple output;
output of mesh entities in blocks No 2–9 is performed without classification to model entities and without any additional data irrespectably to remaining output specification; does not apply to binary output

8192 - split output into separate files according to individual output blocks;
files are distinguished according to suffix added to supplied output file name:

block No 1 - suffix **.numbers**
block No 2 - suffix **.nodes**
block No 3 - suffix **.edges**
block No 4 - suffix **.trias**
block No 5 - suffix **.quads**
block No 6 - suffix **.tetras**
block No 7 - suffix **.pyrams**
block No 8 - suffix **.wedges**

block No 9 - suffix **.hexas**
blocks No 10–14 - suffix **.bassoc**

files corresponding to blocks No 2–9 are created only if the number of corresponding mesh entities is nonzero; file corresponding to blocks No 10–14 is created if output of boundary associated elements is requested; does not apply to binary output

–q - ($_{\text{in}}$#) type of mesh quality report;
any cumulative combination of the following types may be used

0	- no report
1	- element quality report (default)
2	- dihedral angle report
4	- nodal connectivity report
8	- report distribution of selected quantities
16	- disable 2D element quality report
32	- disable 3D element quality report
64	- report all model entities
128	- per model entity report
256	- color 2D element quality
512	- color 3D element quality
1024	- color quality distribution

note that if none from element quality, dihedral angle, or nodal connectivity report is included in the non-zero mesh quality type then element quality report is assumed; currently element quality report is available for all elements; dihedral angle report is available only for unstructured triangles, quadrilaterals, tetrahedra, and structured quadrilaterals; nodal connectivity report is available only for unstructured triangular, quadrilateral, and tetrahedral meshes but it is not complete for 2D mixed meshes and it is unreliable for mixed 3D meshes; by default, only model entities designated for output are subjected to the quality report unless report of all model entities (specification 64) is requested; interface elements are not subjected to quality assessment; mid-nodes are ignored for quality evaluation; color quality types work only together with option *–X*; request for color quality distribution is ignored if drawing of mesh size contours (see *–M*) is required; color quality type specification can be also used to enforce visualisation of quality of boundary mesh (if drawing of boundary mesh is explicitly requested (see *–N* option)); in such a case only the default element quality is displayed

–y - ($_{\text{in}}$#) type of graphics output file;
one of the following types must be used

0	- no specific graphics output (default)
1	- Elixir graphics output
2	- VRML 2.0 graphics output
3	- VTK graphics output

note that this option is relevant only if used together with option *–x* and if the code has been compiled with the *T3D_GRAPHICS_OUTPUT* directive; note that mid-nodes of the mesh (in case of quadratic and quadratic bubble elements) are not exported

–z - ($_{\rm in}$#) graphics output specification;
some cumulative combinations of the following specifications may be used

- 0 - no graphics output specification(default)
- 1 - wireframe graphics output
- 2 - mesh size contours output
- 4 - reverted mesh size contours output
- 8 - black & white mesh size contours output

specification 1 is applied only if VRML output is required; specifications 2, 4, and 8 are irrelevant if used together with specification 1; specifications 4 and 8 are relevant only if used cumulatively with specification 2; note that this option is effective only if used together with nonzero option *–y*; this option is independent of options *–M*; note that contour visualization might not be supported by all VRML viewers; also note that some VRML viewers do not support wireframe view unless explicitly specified; this option is relevant only if the code has been compiled with the *T3D_GRAPHICS_OUTPUT* directive

–d - ($_{\rm fp}$#) default mesh size;
default mesh size is assigned to vertices and control points with zero mesh size specification if local mesh size control is applied (see *–v* option); is used in input file; must be positive

–u - ($_{\rm fp}$#) uniform mesh size;
uniform mesh size is propagated over all model entities; is used in input file; must be positive

–v - ($_{\rm fp}$#) mesh size multiplication factor;
each mesh size specification except the background mesh size specification is multiplied by this factor; must not be negative; default value is 1; zero value disables local mesh size control

–c - ($_{\rm fp}$#) default curvature rate;
it is assigned to model entities without curvature rate specification; controls the ratio between the mesh size and radius of curvature; default value is 1; if set to 0 curvature based mesh size control is disabled for entities that do not specify any curvature rate; is used in input file; must not be negative

–w - ($_{\rm fp}$#) curvature rate multiplication factor;
curvature rate specified for each curve and surface is multiplied by this factor; must not be negative; default value is 1; zero value disables the curvature based mesh size control

–t - ($_{\rm fp}$#) default curve density;
it is assigned to model curves without density specification; determines the mesh size as the ratio between the length of the curve and the density; default value

is 1; if set to 0 density based mesh size control is disabled for entities that do not specify any density; is used in input file; must not be negative

–a - ($_{\text{fp}}$#) background mesh size multiplication factor;
background mesh size specification is multiplied by this factor; must not be negative; zero value disables background mesh size control; default value is 1

–e - ($_{\text{fp}}$#) user defined epsilon;
the value of epsilon is used to resolve the geometrical ambiguity; the default value is 1.0e-5

–R - ($_{\text{fp}}$#) size of the root octant;
this option is ignored if the specified value is smaller then the largest dimension of the bounding box of the domain built in the Cartesian coordinate system; must be positive; note that the value may be increased when *–U* option is specified

–U - (3× $_{\text{fp}}$#) origin of the root octant;
note that this option may cause increase of root octant size specified with *–R* option; by default root octant origin is placed in such a way that the root octant center coincides with the center of the domain bounding box built in the Cartesian coordinate system

–J - (2× $_{\text{fp}}$#) transition factors; the first one, ranging from 1 to 6, controls the maximal allowable aspect ratio of unstructured elements in the transition from fine structured to coarse unstructured mesh; the second one, ranging from 1 to 2, controls the maximal allowable aspect ratio of unstructured elements in the transition from coarse structured to the fine unstructured mesh; default value is 1.5 for both factors

–E - ($_{\text{fp}}$#) octant size excess;
octant size excess controls the maximum ratio between the terminal octant size and the actual required element size; ranges from 1 to 1.5; default value is 1

–X - enables interactive X-window interface based on Elixir graphic library;
this option is available only when source code was compiled with the directive *T3D_ELIXIR* and all prerequisite libraries (Ckit, Elixir, X Window) and header files were available

–D - sets demonstration mode for run without user intervention;
this option works only together with option *–X*

–M - enables drawing of mesh size contours;
this option works only together with option *–X*

–N - enable drawing of background mesh;
this option works only together with option *–X*; when used together with option *–M*, mesh size contours will be applied to background mesh only; this option can be also used to enforce drawing of boundary mesh

–Y - (3× $_{\text{in}}$#) sets plotting specifications for model, mesh, and octree (in this order);
the purpose of this option is to save memory when using graphic interface;
this option works only together with option *–X*
any cumulative combination of the following types may be used for the model drawing specification

0	- do not draw model
1	- draw model vertices (default)
2	- draw model curves (default)
4	- draw model surface, patches, and shells (default)
8	- draw model regions (default)
16	- draw model interfaces (default)
32	- draw model vertex numbers (default)
64	- draw model curve numbers (default)
128	- draw model surface, patch, and shell numbers (default)
256	- draw model region numbers (default)
512	- draw model interface numbers (default)

any cumulative combination of the following types may be used for the mesh drawing specification

0	- do not draw mesh
1	- draw mesh nodes (default)
2	- draw mesh 1D elements (default)
4	- draw mesh 2D elements (default)
8	- draw mesh 3D elements (default)
16	- draw mesh node numbers (default)
32	- draw mesh 1D elements numbers (default)
64	- draw mesh 2D elements numbers (default)
128	- draw mesh 3D elements numbers (default)

one of the following types may be used for the octree drawing specification

0	- do not draw octree
1	- draw octree (default)

note that drawing of the whole model, mesh, and octree is performed by default, which corresponds to *–Y 1023 255 1* plot specification; alternatively, the drawing of the whole model and mesh can be specified by using value -1 for model and mesh drawing specification

–@ - enables phase by phase run;
this option works only if *T3D_PC_VERSION* directive is not defined

–$ - enables step by step run;
this option works only if *T3D_PC_VERSION* directive is not defined

–# - specifies element removal when discretizing regions (unstructured tetrahedral mesh);
this option works only together with options *–X* and *–$*

–ver - prints the version of T3d;
the version information is printed also if any of *–h*, *–P*, *-F*, and *–dir* command line options is specified and if T3d is invoked without any T3d relevant option

–*dir* - prints the overview of compilation directives (including input file line length limit T3D_INPUT_BUFFER_SIZE)

When a t3d parametric option is specified more than once only the last value is accepted. This is not true for file name specification which is supposed to be unique.

The following command line options are not processed by T3d and are passed to the Elixir graphic interface (if available and required)

–*display* (string) - redirects the output

–*geometry* (string) - specifies the geometry

–*fn* (string) - specifies the font

–*font* (string) - specifies the font

–*defcmap* - sets the default color map

–*defvisual* - sets the default visual

–*bestvisual* - sets the best visual

9 Terminal Output

During the runtime, the user is notified about completion of individual phases of the mesh generation, about the number of generated mesh entities, about the nominal profile before and after renumbering (if nodal renumbering was required), about the mesh quality (if quality report was required), and about the time (if running without graphic interface) and memory consumed by the mesh generation (only mesh entities are considered; the amount of consumed memory is depending on T3d specific directive used for compilation).

The number of mesh entities is provided separately for the total number of mesh entities in the mesh (total) and for the number of the mesh entities in the output of finite element mesh (FE) including interface elements.

The nominal profile is defined as number of entries in the upper triangular part of the connectivity matrix (Laplacian matrix of the mesh graph), which have at least one nonzero entry with the same column index and smaller row index, increased by number of nodes. In the case of quadratic elements, the full connectivity is considered (even if element free interface is specified). The nominal profile multiplied by square of degrees of freedom per node can be used to estimate memory requirements for characteristic matrix of the discretized governing differential equation in the finite element analysis.

The quality report depends on the actually used *–q* option. By default, it provides the arithmetic and harmonic means of selected quality quantity, the worst quality (including the number of the worst element (in parenthesis), if its quality falls into the worst quality interval), and the distribution (the first three lines of the quality report in terminal output bellow) of the quality into three quality intervals expressed as the number of elements falling into each interval (interval bounds are in parenthesis) and corresponding percentage of the total number of elements. If required, a more detailed quality distribution is generated.

An example of terminal output follows:

```
T3D - Triangulation of 3D Domains
Copyright: Daniel Rypl, 1995-2008
=================================

t3d -i wheel.in -o wheel.out -d 10 -r 1

Program started          09:48:34
Options analyzed         09:48:34
Input data analyzed      09:48:34
Octree built             09:48:34
Vertices discretized     09:48:34
Curves discretized       09:48:34
Surfaces discretized     09:48:35
Regions discretized      09:48:49
Renumbering completed    09:48:53
```

```
Discretization done       09:48:53
Output data printed       09:48:53
Program finished          09:48:53

Number of nodes:       23264 (FE)     23264 (total)
Number of edges:           0 (FE)    141272 (total)
Number of trias:           0 (FE)    227269 (total)
Number of tetras:     109266 (FE)    109266 (total)

Nominal profile:   167094801 (old)    5786950 (new)

Quality of unstructured tetrahedra:
     94122 : (1.000 - 0.750)  86.14 %
     15000 : (0.750 - 0.500)  13.73 %
       144 : (0.500 - 0.000)   0.13 %
   0.85554 : arithmetic mean quality
   0.84286 : harmonic mean quality
   0.21759 : worst quality (9554)

Real time consumed          19.77 sec
Memory consumed             44700 kB
```

10 Quality Evaluation

The quality is evaluated separately for unstructured triangular, quadrilateral, tetrahedral, and pyramid elements, for structured quadrilateral and hexahedral elements, and semistructured wedge and hexahedral elements. In the current implementation, iso elements are excluded from the quality evaluation because their quality is always equal to 1. Also quality of wedges created from hexahedra due to degeneracy caused by resolving master-slave relationship is not assessed. The mid-nodes of quadratic and quadratic bubble elements are not considered for the quality evaluation. By default only elements on the output are subjected to the quality evaluation. Note that interface elements are always excluded from quality evaluation. Also note that surfaces, patches, and shells subjected to mapping which, however, does not result in pure structured quadrilateral mesh (for example due to fixed vertices to be accommodated, due to convexity check, or due to diagonal splitting) are also excluded from quality assessment.

There are recognized three distinct quality quantities

- element shape based quality,
- dihedral angle based quality, and
- connectivity based quality.

The elements shape based quality is evaluated according to the following formulas (quality is normalized $Q \in \langle 0, 1 \rangle$)

- unstructured triangular element

  $$Q_{ut} = \frac{4}{3}\sqrt{3}\ \frac{A}{\sqrt{(a^2+b^2+c^2)/3}^{\,2}}$$

  where A is the area of the triangle and a, b, and c are the lengths of its sides. Maximum quality is obtained for equilateral triangle.

- planar unstructured quadrilateral element

  $$Q_{puq} = \frac{4}{3}\ \sqrt{Q_{ut_1} Q_{ut_2} Q_{ut_3} Q_{ut_4}}$$

  where Q_{ut_i} ($i = 1, 2, 3, 4$) are the qualities of the four unstructured triangular elements obtained by two possible diagonal splittings of the assessed quadrilateral element. Maximum quality is obtained for square. Quality of nonconvex quadrilateral is set to zero.

- nonplanar unstructured quadrilateral element

  $$Q_{nuq} = Q_{puq} \max(\cos\alpha_{13}, 0)\ \max(\cos\alpha_{24}, 0)$$

  where α_{ij} is the angle between the normals of the triangular elements obtained by splitting the assessed quadrilateral element along diagonal connecting nodes i and j.

- planar structured quadrilateral element

$$Q_{psq} = \frac{4}{\sum_{i=1}^{4} 1/(1-\cos^2\alpha_i)}\; 0.5\Big(\frac{\min(d_1,d_3)}{\max(d_1,d_3)} + \frac{\min(d_2,d_4)}{\max(d_2,d_4)}\Big)$$

  where α_i is angle between adjacent edges incident to node i and d_j is the length of j-th side. Maximum quality is obtained for rectangle. Quality of nonconvex quadrilateral is set to zero.

- nonplanar structured quadrilateral element

$$Q_{nsq} = Q_{psq} \max(\cos\alpha_{13}, 0)\, \max(\cos\alpha_{24}, 0)$$

  where α_{ij} is the angle between the normals of the triangular elements obtained by splitting the assessed quadrilateral element along diagonal connecting nodes i and j.

- unstructured tetrahedral element

$$Q_{utet} = \frac{3}{4}\sqrt[4]{108}\; \frac{V}{\sqrt{(A^2+B^2+C^2+D^2)/4}^{\,3/2}}$$

  where V denotes the volume of the tetrahedron and A, B, C, and D are the areas of its faces. Maximum quality is obtained for equilateral tetrahedron.

- unstructured pyramid element

$$Q_{up} = \frac{3}{8}\sqrt[4]{108}\; \frac{V}{\sqrt{(A^2+B^2+C^2+D^2)/4}^{\,3/2}}$$

  where V denotes the volume of the pyramid and A, B, C, and D are the areas of its triangular faces. The volume is taken as the average of volumes obtained for two pairs of tetrahedra for two possible diagonal splittings of pyramid base face. Maximum quality is obtained for equilateral pyramid.

- semistructured wedge element

$$Q_{ssw} = \frac{Q_{ut,b}+Q_{ut,t}}{2}\; \frac{\sum_{i=1}^{3} Q_{nsq,i}}{3}$$

  where $Q_{ut,b}$ and $Q_{ut,t}$ stand for the quality of bottom and top triangular bases and $Q_{nsq,i}$ denotes the quality of the i-th quadrilateral side. Maximum quality is obtained for prism with parallel equilateral bases of the same size and lateral rectangular sides of the same size. If quality of any of the bases or sides is zero, the overall wedge quality is set to zero.

- semistructured hexahedral element

  $$Q_{ssh} = \frac{Q_{uq,b} + Q_{uq,t}}{2} \frac{\sum_{i=1}^{4} Q_{nsq,i}}{4}$$

  where $Q_{uq,b}$ and $Q_{uq,t}$ stand for the quality of bottom and top quadrilateral bases and $Q_{nsq,i}$ denotes the quality of the i-th quadrilateral side. Maximum quality is obtained for prism with parallel square bases of the same size and lateral rectangular sides of the same size. If quality of any of the bases or sides is zero, the overall hexahedral quality is set to zero.

- structured hexahedral element

  $$Q_{sh} = \frac{6}{\sum_{i=1}^{6} 1/Q_{nsq,i}}$$

  where $Q_{nsq,i}$ stands for the quality of the i-th quadrilateral side. Maximum quality is obtained for cube. If quality of any of sides is zero, the overall hexahedral quality is set to zero.

The dihedral angle based quality is evaluated according to the following formulas (quality is normalized $Q \in \langle 0, 1 \rangle$)

- unstructured simplex element (triangle, tetrahedron)

  $$Q_s = \frac{\delta_{min}}{\delta_{max}},$$

  where δ_{min} is the minimal and δ_{max} the maximal dihedral angle in a particular element. Maximum quality is obtained for equilateral elements.

- quadrilateral element (unstructured and structured)

  $$Q_q = 1.0 - \max(\delta_{max} - \frac{\pi}{2}, \frac{\pi}{2} - \delta_{min}) \Big/ \frac{\pi}{2}$$

  where δ_{min} and δ_{max} are again the minimal and the maximal dihedral angle in the assessed (convex) quadrilateral element. Maximum quality is obtained for rectangular elements. Quality of nonconvex quadrilateral is set to zero.

The connectivity based quality compares the valence of individual internal nodes with the optimal valence which is exactly 6 for triangular mesh, 4 for quadrilateral mesh, approximately 12 for tetrahedral mesh, and exactly 8 for hexahedral mesh. The connectivity based quality is available only for unstructured meshes. For mixed 2D meshes, only internal nodes shared by elements of the same type are considered. For mixed 3D meshes, all internal nodes irrespectably of type of connected elements are considered which makes the results spurious.

11 Warnings and Error Messages

By default, warnings and error messages are printed on terminal output. They may be suppressed using *–Q*, *–Qout* or *–Qerr* option or redirected to a file (together with standard terminal output) using *–g* option. There are several types of errors each one with different exit code

- option error – exit code 5,
- manipulation error – exit code 10,
- memory allocation error – exit code 15,
- file manipulation error – exit code 20,
- input data error – exit code 25,
- output data error – exit code 30,
- binary input data error – exit code 35,
- binary output data error – exit code 40, and
- mesh limit error – exit code 45.

Each error contains except its type also more or less specific explanation of the error. The place where the error was generated can be identified by the line number, function name, and module name. In the case of input/output data error, the line number and the input/output file name is also provided. For binary input/ouput data error, only the binary input/output file name is given.

12 Input File Examples

The format of input file is demonstrated on a simple example of a bus shelter.

12.1 Bus Shelter

This is a simple example demonstrating the syntax of T3d input records of basic entities such as vertex, curve, surface, and patch. The vertex and curve numbering is shown in Figures 8a and 8b. The geometry of curves 22 and 24 is given by the formulas for 180 degree circular arc represented by cubic curve (see Appendix Section A.1.2). The roof of cylindrical shape is conveniently represented by a surface cubic in one direction and linear in the other one thus there are no inner control points of the surface control polygon. The vertical walls and the horizontal bench are modelled by planar patches. The complete rendered model is depicted in Figure 8c. The shaded view of the final mesh obtained via executing the command line specified at the top of the input file is displayed in Figure 8d.

The actual content of the input file follows:

```
###############
# Bus shelter #
###############

## command line
## t3d -i shelter.in -o shelter.out -d 0.5

# base vertices
vertex 1 xyz 0 0 0
vertex 2 xyz 5 0 0
vertex 3 xyz 0 2 0
vertex 4 xyz 5 2 0

# roof vertices
vertex 11 xyz 0 0 2
vertex 12 xyz 5 0 2
vertex 13 xyz 0 2 2
vertex 14 xyz 5 2 2

# columns (z-direction)
curve 1 vertex 1 11
curve 2 vertex 2 12
curve 3 vertex 3 13
curve 4 vertex 4 14 count 2

# sill and beams (x-direction)
curve 11 vertex 1 2
curve 12 vertex 11 12
curve 13 vertex 13 14

# sill, beams and arches (y-direction)
curve 21 vertex 1 3
curve 22 order 4 vertex 11 13
polygon 1 xyz 0 0 4 weight 0.3333333333
polygon 2 xyz 0 2 4 weight 0.3333333333
curve 23 vertex 12 14
curve 24 order 4 vertex 12 14
polygon 1 xyz 5 0 4 weight 0.3333333333
polygon 2 xyz 5 2 4 weight 0.3333333333

# bench vertices
vertex 21 xyz 0 0 0.5 fixed curve 1
vertex 22 xyz 4 0 0.5
vertex 23 xyz 0.5 0.5 0.5
vertex 24 xyz 4 0.5 0.5
vertex 25 xyz 0 1.5 0.5
vertex 26 xyz 0.5 1.5 0.5

# bench curves
curve 31 vertex 21 22 factor 0.5
curve 32 vertex 23 24
curve 33 vertex 25 26
curve 34 vertex 21 25 factor 0.5
curve 35 vertex 23 26
curve 36 vertex 22 24

# walls
patch 1 normal 0 1 0 boundary curve -11 1 12 -2 \
        fixed curve 31 size def
patch 2 normal 1 0 0 boundary curve 21 3 -22 -1 \
        fixed curve 34 size def
patch 3 normal 1 0 0 boundary curve 23 -24 size def

# roof
surface 1 curve 12 22 13 24

# bench
patch 11 normal 0 0 1 boundary curve 31 -32 -33 -34 35 36 size def
```

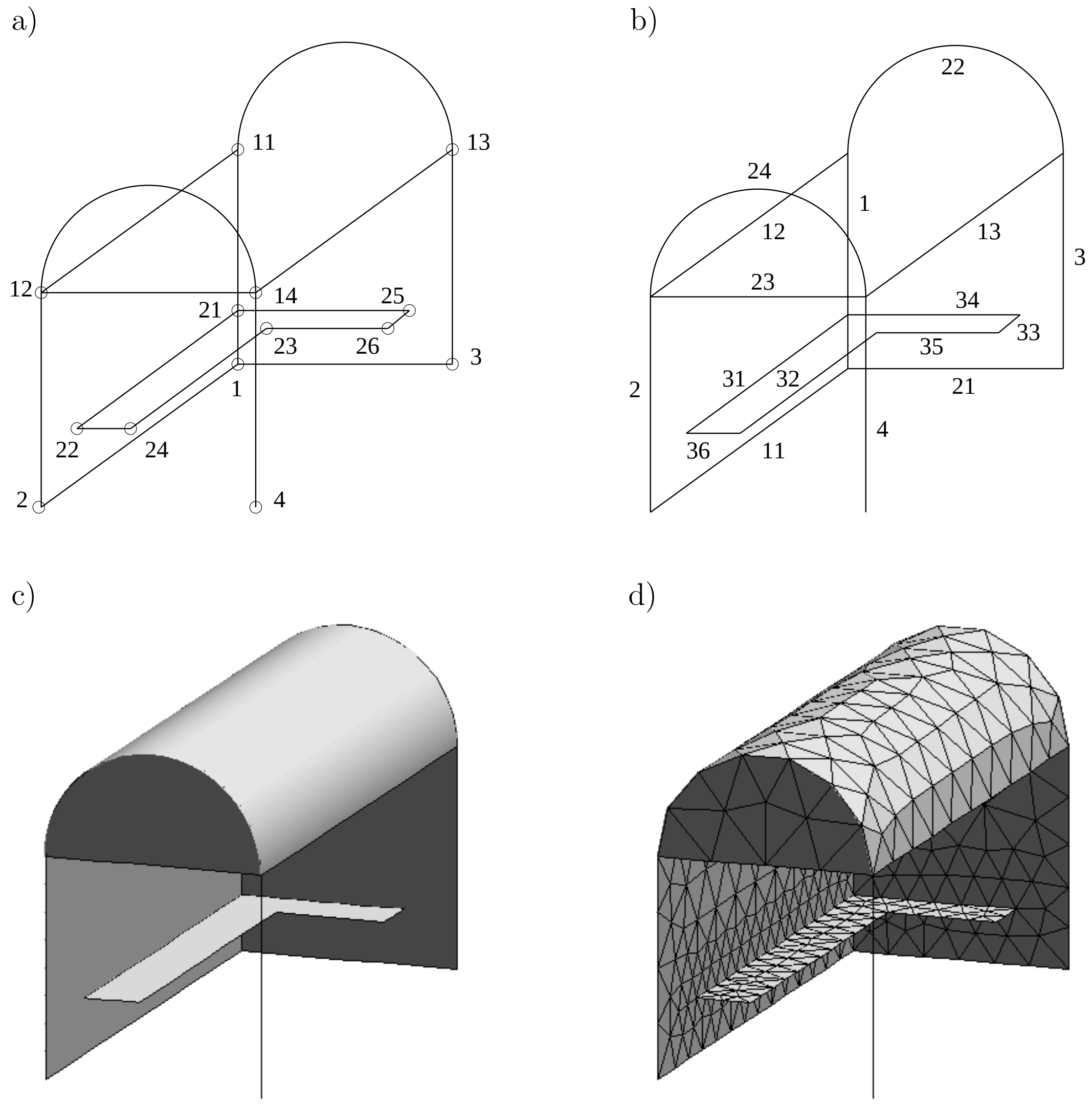

Figure 8: Bus shelter: a) vertex numbers, b) curve numbers, c) model, d) mesh.

A Modelling of Conics

In this appendix, the exact representation of conical arcs by rational Bezier curves of quadratic and cubic degree is described in terms of the geometry of the control polygon and weights of its vertices. Note that the formulas for cubic curves have been derived by expansion of a quadratic curve and that negative values of weights have been considered.

A.1 Circular Arc

A.1.1 Quadratic Curve

$$\delta = R\frac{\sin^2\frac{\alpha}{2}}{\cos\frac{\alpha}{2}}$$

$$\phi = R\sin\frac{\alpha}{2}$$

$$s = R\frac{\sin\frac{\alpha}{2}}{\cos\frac{\alpha}{2}}$$

$$\omega_0 = \omega_2 = 1 \qquad \omega_1 = \cos\frac{\alpha}{2}$$

$$\alpha \in (0; 2\pi) \setminus \{\pi\}$$

A.1.2 Cubic Curve

$$\delta = 2\,R\frac{\sin^2\frac{\alpha}{2}}{1+2\cos\frac{\alpha}{2}}$$

$$\phi = R\frac{\sin\alpha}{1+2\cos\frac{\alpha}{2}}$$

$$s = 2\,R\frac{\sin\frac{\alpha}{2}}{1+2\cos\frac{\alpha}{2}}$$

$$\omega_0 = \omega_3 = 1 \qquad \omega_1 = \omega_2 = \frac{1+2\cos\frac{\alpha}{2}}{3}$$

$$\alpha \in (0; 2\pi) \setminus \{\frac{4}{3}\pi\}$$

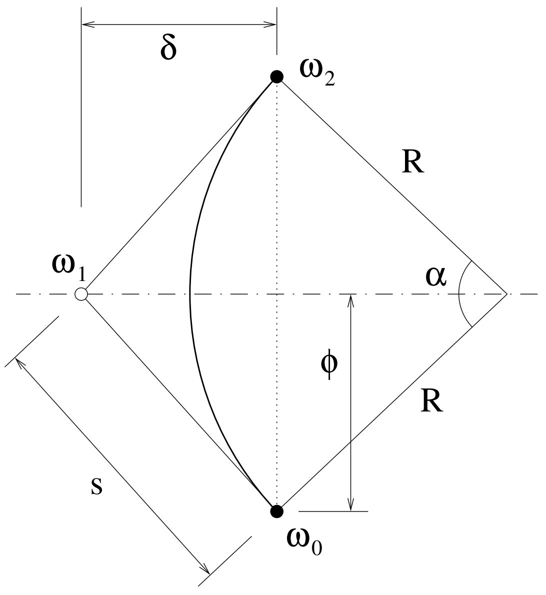

Figure 9: Circular arc - quadratic curve.

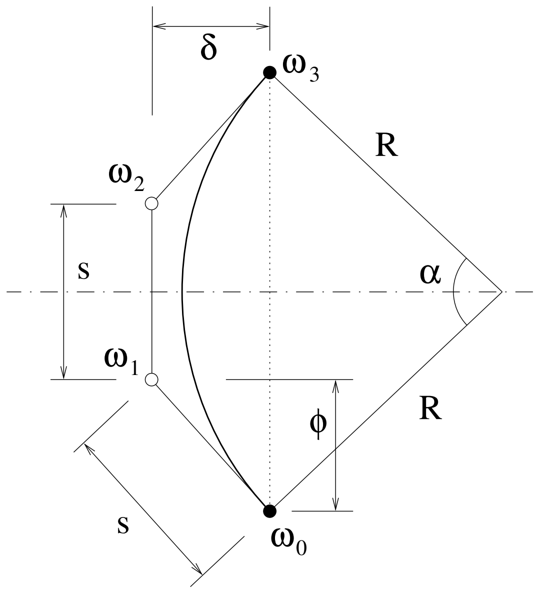

Figure 10: Circular arc - cubic curve.

A.2 Elliptic Arc

A.2.1 Quadratic Curve

$$\delta = b\frac{\sin^2\frac{\alpha}{2}}{\cos\frac{\alpha}{2}}$$

$$\phi = a\sin\frac{\alpha}{2}$$

$$s = \frac{\sin\frac{\alpha}{2}}{\cos\frac{\alpha}{2}}\sqrt{a^2\cos^2\frac{\alpha}{2} + b^2\sin^2\frac{\alpha}{2}}$$

$$\omega_0 = \omega_2 = 1 \qquad \omega_1 = \cos\frac{\alpha}{2}$$

$$\alpha \in (0; 2\pi) \setminus \{\pi\}$$

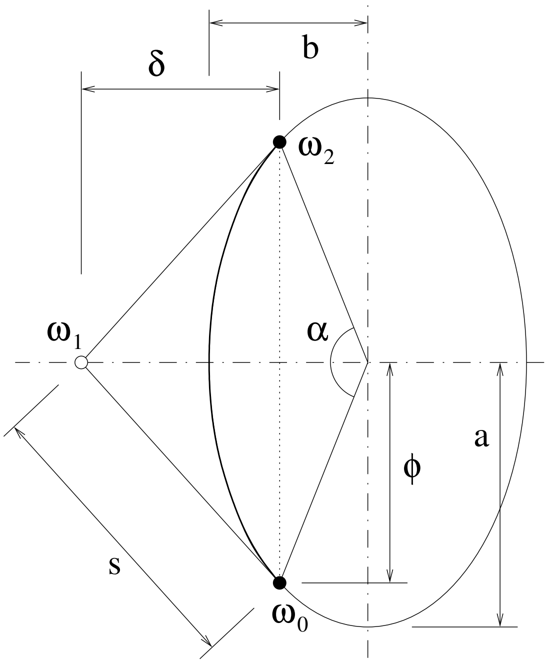

Figure 11: Elliptic arc - quadratic curve.

A.2.2 Cubic Curve

$$\delta = 2\,b\frac{\sin^2\frac{\alpha}{2}}{1+2\cos\frac{\alpha}{2}}$$

$$\phi = a\frac{\sin\alpha}{1+2\cos\frac{\alpha}{2}}$$

$$s = \frac{2\sin\frac{\alpha}{2}}{1+2\cos\frac{\alpha}{2}}\sqrt{a^2\cos^2\frac{\alpha}{2} + b^2\sin^2\frac{\alpha}{2}}$$

$$\omega_0 = \omega_3 = 1 \qquad \omega_1 = \omega_2 = \frac{1+2\cos\frac{\alpha}{2}}{3}$$

$$\alpha \in (0; 2\pi) \setminus \{\frac{4}{3}\pi\}$$

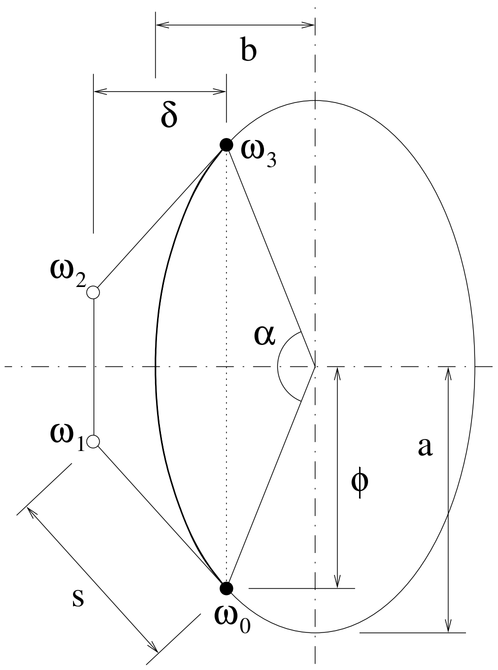

Figure 12: Elliptic arc - cubic curve.

A.3 Parabolic Arc

A.3.1 Quadratic Curve

$$\delta = 2al^2 \qquad \phi = l$$

$$s = l\sqrt{4a^2l^2 + 1}$$

$$\omega_0 = \omega_1 = \omega_2 = 1$$

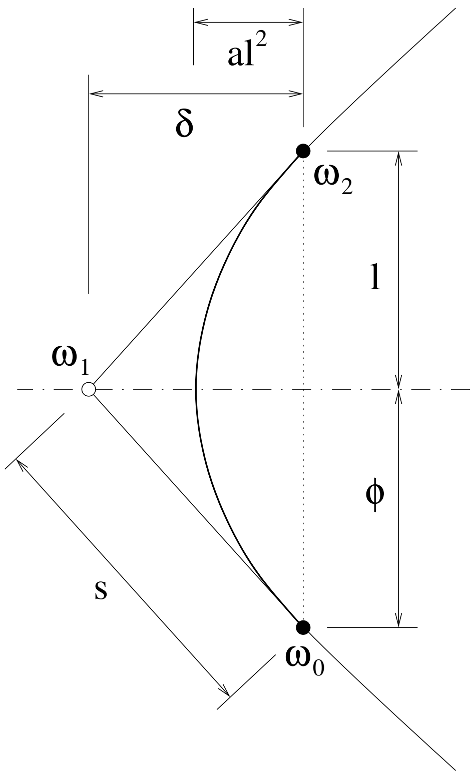

Figure 13: Parabolic arc - quadratic curve.

A.3.2 Cubic Curve

$$\delta = \frac{4}{3}al^2 \qquad \phi = \frac{2}{3}l$$

$$s = \frac{2}{3}l\sqrt{4a^2l^2 + 1}$$

$$\omega_0 = \omega_1 = \omega_2 = \omega_3 = 1$$

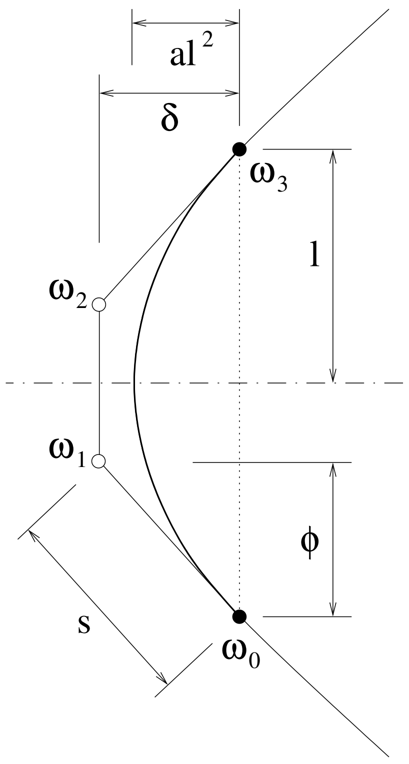

Figure 14: Parabolic arc - cubic curve.

A.4 Hyperbolic Arc

A.4.1 Quadratic Curve

$$\delta = \frac{al^2}{b\sqrt{b^2+l^2}}$$

$$\phi = l$$

$$s = \frac{l\sqrt{l^2(a^2+b^2)+b^4}}{b\sqrt{b^2+l^2}}$$

$$\omega_0 = \omega_2 = 1$$

$$\omega_1 = \frac{\sqrt{b^2+l^2}}{b}$$

A.4.2 Cubic Curve

$$\delta = \frac{2al^2}{b(b+2\sqrt{b^2+l^2})}$$

$$\phi = \frac{2l\sqrt{b^2+l^2}}{b+2\sqrt{b^2+l^2}}$$

$$s = \frac{2l\sqrt{l^2(a^2+b^2)+b^4}}{b(b+2\sqrt{b^2+l^2})}$$

$$\omega_0 = \omega_3 = 1$$

$$\omega_1 = \omega_2 = \frac{b+2\sqrt{b^2+l^2}}{3b}$$

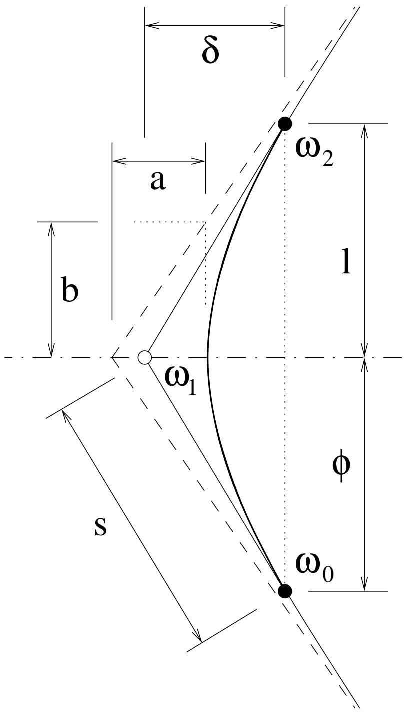

Figure 15: Hyperbolic arc - quadratic curve.

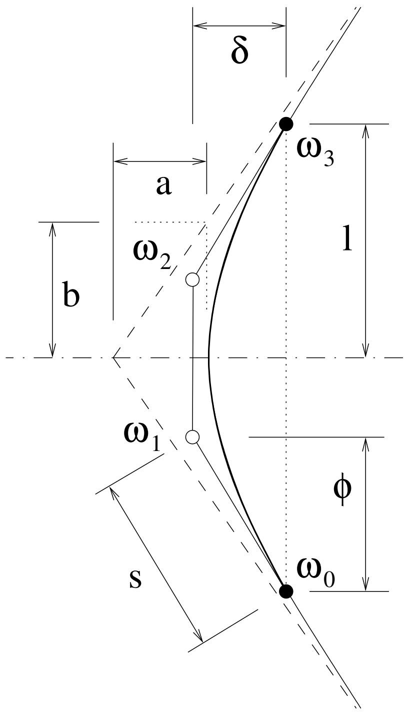

Figure 16: Hyperbolic arc - cubic curve.

B Modelling of Rotational Surfaces

B.1 Quadratic Expansion

$$\omega_\alpha = \cos\frac{\alpha}{2} \qquad \alpha \in (0; 2\pi) \setminus \{\pi\}$$

$$\omega_{00} = \omega_{20} = \omega_0 \qquad \omega_{01} = \omega_{21} = \omega_1 \qquad \omega_{02} = \omega_{22} = \omega_2 \qquad \omega_{03} = \omega_{23} = \omega_3$$

$$\omega_{10} = \omega_0\omega_\alpha \qquad \omega_{11} = \omega_1\omega_\alpha \qquad \omega_{21} = \omega_2\omega_\alpha \qquad \omega_{13} = \omega_3\omega_\alpha$$

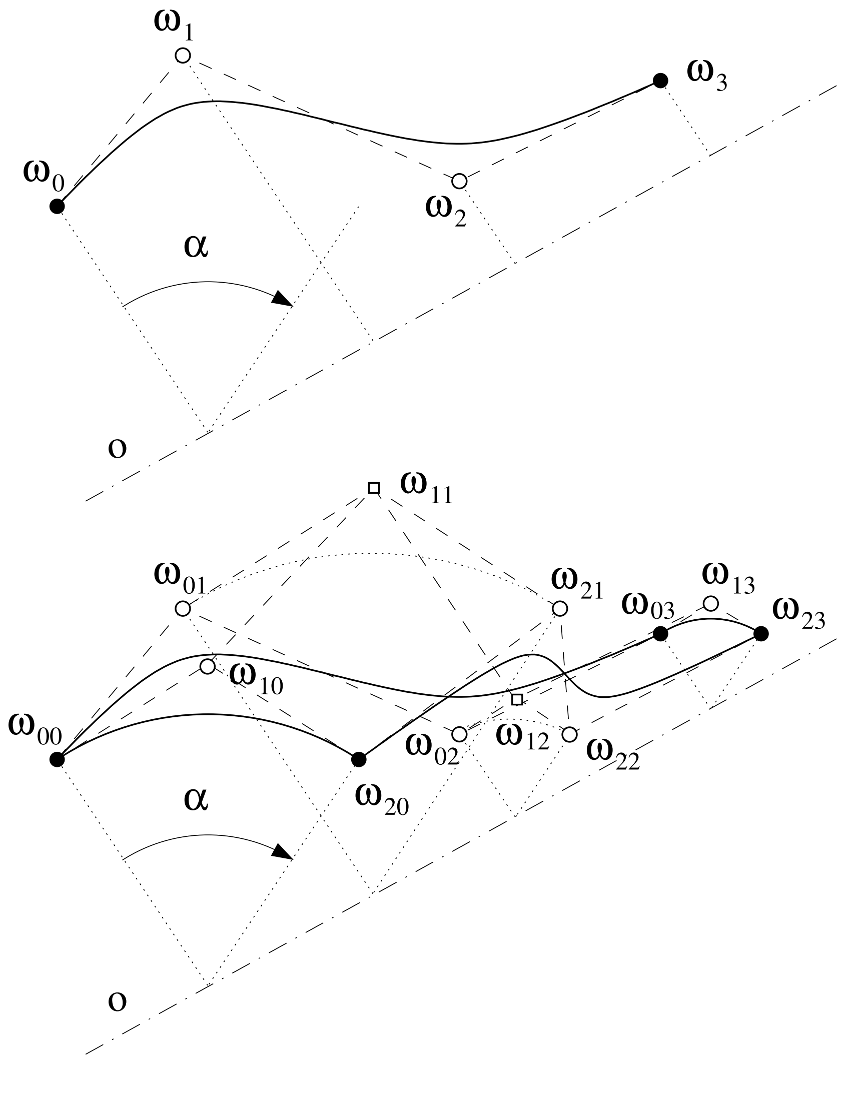

Figure 17: Rotational surface - quadratic expansion.

The positions of control polygons points $\boldsymbol{P}_{1j}$, are calculated according to the formulas in Section A.1.1. Note that the revoluted curve can be generally three-dimensional.

B.2 Cubic Expansion

$$\omega_\alpha = \frac{1 + 2\cos\frac{\alpha}{2}}{3} \qquad \alpha \in (0; 2\pi) \setminus \{\frac{4}{3}\pi\}$$

$$\omega_{00} = \omega_{30} = \omega_0 \qquad \omega_{01} = \omega_{31} = \omega_1 \qquad \omega_{02} = \omega_{32} = \omega_2 \qquad \omega_{03} = \omega_{33} = \omega_3$$

$$\omega_{10} = \omega_{20} = \omega_0\omega_\alpha \qquad \omega_{11} = \omega_{12} = \omega_1\omega_\alpha \qquad \omega_{21} = \omega_{22} = \omega_2\omega_\alpha \qquad \omega_{13} = \omega_{23} = \omega_3\omega_\alpha$$

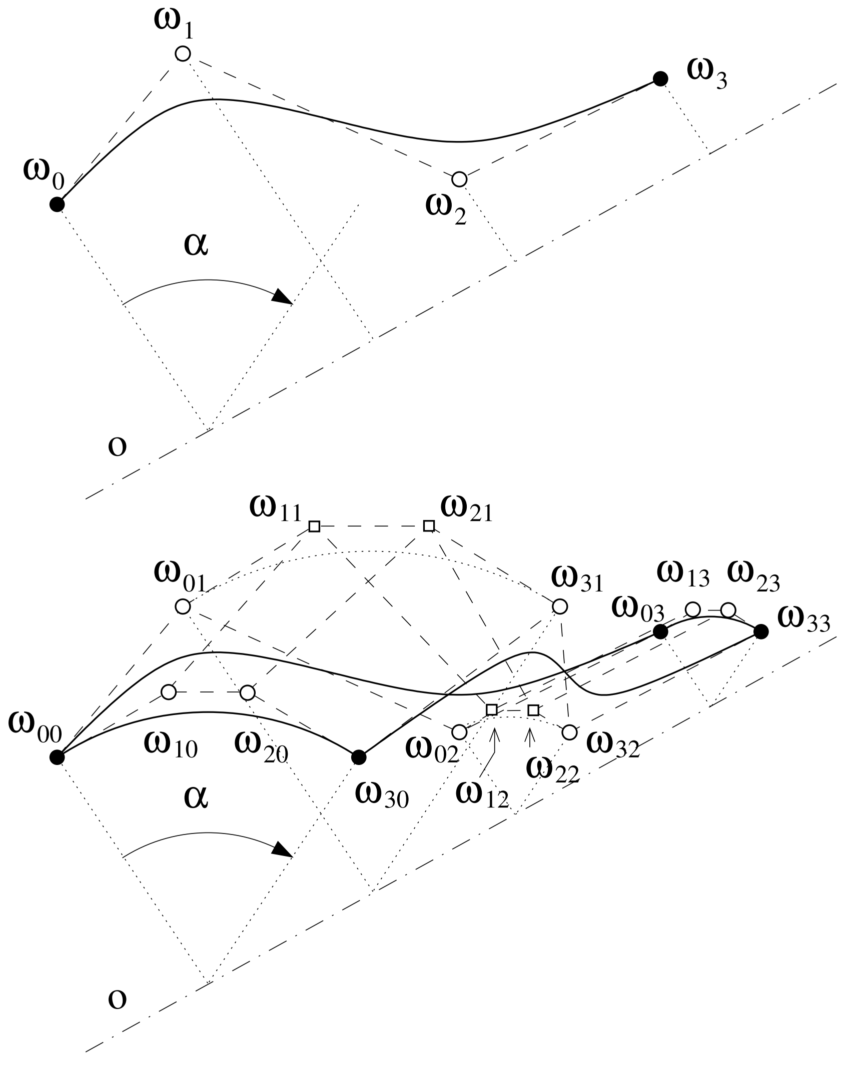

Figure 18: Rotational surface - cubic expansion.

The positions of control polygons points $\boldsymbol{P}_{ij}$, $i = 1, 2$, are calculated according to the formulas in Section A.1.2. Note that the revoluted curve can be generally three-dimensional.

C Run-Time Visualization

When running T3d with Elixir graphic interface, the bottom menu palette provides the user with all the facilities to control the run of T3d and to visualize the results. There are few useful key and mouse bindings

Accelerators

Ctrl a	fit all (all drawing windows are affected)
Ctrl p	proceed run
Ctrl s	stop run
Ctrl x	exit (no results will be printed)

Fast viewing

B1	zoom window (choose two opposite corners)
Ctrl B1	pan view
Ctrl B2	zoom view
Shift B2	fit all (only active drawing window is affected)
Ctrl Shift B1	rotate view
B3	done

Graphics selection

B1	select
Ctrl B1	select by window (choose two opposite corners)
Shift B1	select nearest point (confirm by B1 or select next one by Shift B1)
B2	accept selection
B3	reject selection

Handler control

Ctrl B3	suspend handler
B3	resume handler

B1, B2, and B3 stand for left, middle, and right mouse button.

In the case that an error in input data has been detected, the user is interactively asked for starting a graphic interface (to view at least the part of input data which has been successfully parsed). A time out is used to prevent blocking. Note that this is an optional feature and might not be available.