TARGE2

"T"riangulation of "AR"bitrary "GE"ometries in "2"D:

User's Manual

Petr Krysl

Czech Technical University in Prague,
Faculty of Civil Engineering, Dept. of Structural Mechanics,
166 29 Prague, Czech Republic,
email: pk@power2.fsv.cvut.cz

LEGAL NOTICE

Chapter 1

Introduction

1.1 Version

This is a preliminary version of the user's guide to the `targe2` program (Feb 1995 release). Please, email all your suggestions, remarks and criticisms to the author.

1.2 What `targe2` can do

The `targe2` program was designed to triangulate arbitrary planar regions. The algorithm used is of the "advancing front" family. The triangles are created one by one, and the edges bounding the inside of the domain constitute the advancing front. The heuristic criteria for placement of the triangle to be created have been modified and extended to (i) improve the quality of the triangles, and (ii) to accomodate the creation of anisotropic (directed) meshes. The `targe2` program was designed to run with very good computational complexity – the cost is in some cases a linear function of the number of triangles and in general the computational complexity of of order $O(N \log N)$ (see [1] for details).

1.2.1 Regions

The regions to be triangulated are described by their boundary curves. The only requirement posed on the boundary is that it must be closed. Therefore, the regions may be e.g. such as in Fig. 1.2.1.

The first region is composed of two subregions, touching each other in two curves. The second region of Fig. 1.2.1 is a single subregion with 4 holes through it. The third region is composed of one outer subregion, enclosing two inner ones (one of them with an additional through-hole). The fourth region is really *only one* subregion – its boundary is composed of closed curves (circles). One

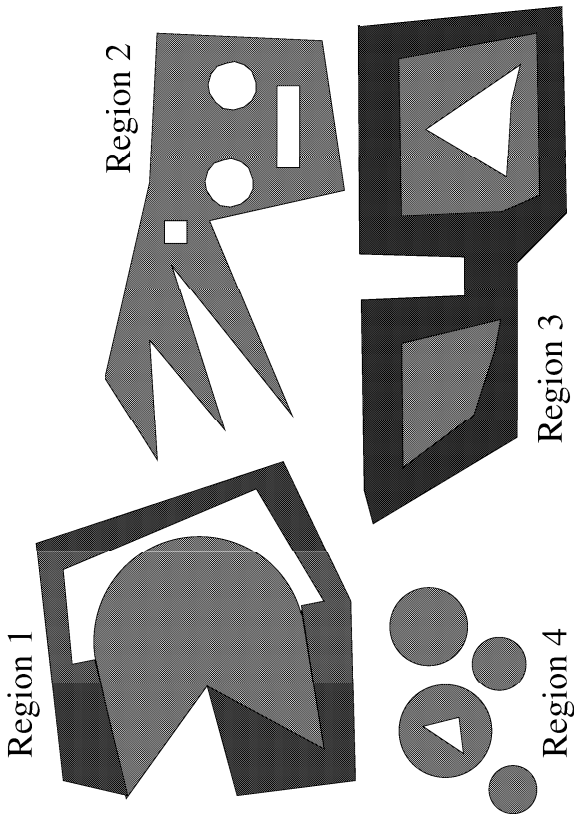

Figure 1.1: Some examples of regions that can be triangulated.

of the circles contains a triangular void. The four examples of Fig. 1.2.1 give an overview of what can be meshed with `targe2` .

1.2.2 Gradation

The triangulations can be arbitrarily graded. The `targe2` program implements two ways how to control the gradation. More details on the mesh size control (for the isotropic case) should appear in [2].

Control-point based gradation control

The gradation can be defined by specifying one constant (mesh size over all the region – this should be probably based on how good the approximation of the boundary should be), and by setting any number of refinement areas through control points. The control points influence their neighborhood through global

shape functions associated to them. Due to the global nature of the shape functions, every control point influences all all other control points and also the constant mesh size; It is therefore not possible to specify exactly the desired mesh size at a point – it always results as a merge of the constant mesh size and of the influences of the refinement areas. While this is perfectly acceptable when generating ad hoc meshes, another device must be adopted when using `targe2` in adaptive computations, where the desired gradation is estimated by evaluating optimal triangle size at the given point from the error indicator at that point.

Quadtree-based mesh control

In the case we have the desired mesh gradation specified in the form of a function over a background mesh, i.e. through interpolation from nodal values on some mesh (e.g. the previous mesh in adaptive computations), the following strategy is implemented in `targe2` . The mesh size function defined on the background mesh is accessed many times during the triangulation. To evaluate the mesh control function at a given point, we have to find the triangle enclosing the point. The search through the whole background mesh is unacceptably slow, however. While different approaches have been presented in the literature, they all have in common that a tree is used to speed up the searching. The resulting computational complexity is still not sufficiently low, unfortunately. Therefore, `targe2` approximates the mesh control information on the background mesh by a discontinuous function consisting of its values tranferred directly to the leaves of a quadtree. The access to the mesh control information is then speeded up considerably, as the searching and interpolation are not needed any more.

1.2.3 Anisotropic meshes

The strength computations of structural mechanics usually require isotropic meshes. On the other hand, there are applications where the use of directed (oriented, anisotropic) meshes is imperative (e.g. viscous flows in boundary layers, or generation of triangles on 3D surface patches). The `targe2` program implements the creation of oriented meshes. The procedure has been added to support triangulation of parametric 3D surfaces. Note, that instead of using the local-mapping approach supported by Peraire, the triangles are generated in their actual shape (all the transformations needed in the former approach are not done here)[2].

1.2.4 Smoothing

The final mesh may be smoothed either by applying the Laplacian smoother (in fact force relaxation, with springs corresponding to the edges), or the anisotropic

smoother (under investigation). Note, that the only mesh improvement techniques that have been implemented are triangle merging for vertices with only three or four incident edges. Three triangles around the vertex are collapsed into a single one, and four triangles are merged into two better shaped ones. It was not felt that other fancy mesh improvement techniques were of primary importance, as usually more than 98% of the generated triangles pass rather stringent quality test based on the ratio of circumscribed circle radius to inscribed circle radius.

1.3 How to get it

The `targe2` program is available (with all needed libraries) from the author as it is the free software. Ask for details at `pk@power2.fsv.cvut.cz`.

1.4 Installation

To install the `targe2` program you need to have the `Ckit` library (lists, stacks, hash tables and other goodies). It is all you need, in case you do not intend to use X Window System based graphic interface to monitor the triangulation and to evaluate the final mesh.

On the other hand, for those who would wish to have a graphic interface (and I strongly recommend using it), there is a one. It relies on the `Elixir` library for 3D graphics written also by the author of `targe2` . The `Elixir` library is free software and you can obtain it on request (if it is not included with `targe2`).

The installation should be relatively straightforward, see the appropriate `README` file for details.

Chapter 2

Running

The `targe2` program is run usually with a lot of options to select features or to set options if the defaults are not useful in the given circumstances. The options that are available to you are printed by `targe2` when run without any parameters or with parameter "`-?`". The printout also shows any default values for the options. The best way how to maintain the below list up-to-date seems not to be specific here about the default values used in the code.

Note, that the default values of the settings may be inspected by running `targe2` *without any command-line arguments (or with the "*`-?`*" or "*`-h`*" options).*

2.1 File options

`-i` *file* Here *file* is the name of the input file. It is assumed, that if the file named does not exist, that the file name obtained by appending the suffix ".in" could have been meant. This is the only required option.

`-n` No output (does not write any results out),

`-f` *format* Output format. *format* equal to 1 means AFRO output format, *format* equal to 2 means MOS output format.

`-o` *file* output file name. If not given, it is constructed from the input file name to end on ".out" by tearing off any "suffixes",

`-L` *file* Log-file (verbose mode might print into a log file instead).

2.2 Run modes

`-b` *path* Graphic driver path (meaningful only on PCs),

`-v` Verbose (prints some information of general interest),

`-s` Silent (does not print anything),

`-X` Use X Window System graphical interface (meaningful only on UNIX graphical workstations when the `targe2` program was configured and compiled to run with X-based interface),

`-$` Stop for each triangle in the generation loop. It makes sense only when debugging with the graphic interface.

`-g` Use MS-DOS graphic interface (meaningful only on PCs).

2.3 Generation parameters

`-D` Do not smooth mesh. Mesh smoothing means relaxation of the vertex locations here.

`-m` *num* Merge tolerance (needs to be set to be able to merge vertices of connected curves, e.g. if the dimensions of the domain to be triangulated are very small).

`-C` *num* Cell-size/pulled-edge length ratio. A good choice is the default setting. Very strongly graded meshes may require larger values (e.g. 10). Note: Large values lead to deteriorated performance.

`-S` *num* Maximum stretch of ideal-point placement ($S \geq 1$). You allow basically to generate the ideal point of the current triangle at such a distance, that the ratio of the length of the new edges is at most S times the length of the pulled edge.

`-e` *num* Mesh control shape function exponent μ (see below).

`-B` *num* Hash-function parameter β ($\beta > 1$). The default value is sufficient for gradations with the ratio of longest to shortest edge of about $500,000$. You should not set β to more than 1.2 (it influences the quality of the mesh).

`-M` *num* Magnification factor in grid rebuild. Optimal value 1.1 is the default. There should be no reason to change this.

`-O` *num* Smoother type. Laplacian relaxation of vertex location is selected by 2, anisotropic relaxation is selected by 1.

`-p` *num* Pischweitz constant (do not ever ask what is it for, you would faint, fearless art thee). But seriously: it is used currently to restrict searches through the front when generating extremely stretched elements (it is not yet mature enough to be described in more detail).

`-Q` *num* Improve quality by triangle merging, if argument equal to 1; do not merge triangles, if argument equal to 0.

2.4 Mesh control function selection

`-q` Use quadtree to control mesh size. Note that to be able to use the quadtree-based control the input file *must* start with a `REGEXT` command.

`-F` *num* use analytical function mesh control of number given.

`-E` Produce quasi-estimate of the mesh size.

2.5 Parameters specific to anisotropic meshes

`-A` generate anisotropic mesh,

`-Z` *num* Select what should be drawn to visualize the anisotropic mesh control parameters. Applies only to anisotropic meshes. Either show as Z coordinate (1=h, 2=kappa, 3=alpha) or draw mesh size control cross (4=mesh-size cross).

Chapter 3

Input format

3.1 Conventions

The input to the generator is an ASCII file, containing the definition of the region(s) to be triangulated and the mesh control parameters. The input is in free format. Everything following an exclamation mark (`!`) is considered to be a comment.

```
! square with a very strong gradation at two corners (1:500,000)
```

An input line may be continued on practically unlimited number of physical lines by ending them with a blank followed by a double dash (" `--`"). Example:

```
curve  5 discretized 0 1    --
                     0.85 1 --
                     1 1
```

The input as viewed by the input subsystem of `targe2` consists of strings and numbers (these are collectively called *tokens*). The tokens are separated by blanks (in any quantity > 1). The *numbers* are of the form recognized by `strtod()` standard ANSI function. The *strings* are the tokens not recognized as numbers. When writing the input file, case of letters is unimportant. Note, that the currently fixed maximum number of tokens per line is given in `ttypes.h` file. It should be rather large number so that it should not be necessary to adjust it.

The syntax descriptions below use the denotations `<INT>`, `<FLOAT>`, `<STRING>` and `<INT-LIST>` to stand for integer or float number, string and integer list (i.e. a series of integer numbers). The brackets (`[]`) surround an optional item. The upper-case strings in the syntax definitions are keywords, strings in lower-case are argument identifiers. Example: `<INT id>` denotes an identifier named `id` of type `<INT>`.

3.2 Input file subdivision

The input file should be subdivided into the following blocks.

1. Definition of mesh control.
2. Definition of curves.
3. Definition of subregions.
4. Definition of fixed points.

This might be important as in some instances the definition of curves relies on the mesh control information to be complete (and of course, the subregions must be specified after the boundary curves have been described).

3.3 Definition of mesh control

The mesh size control is available in two species: isotropic and anisotropic. The isotropic meshes should ideally consist of equilateral triangles, while the anisotropic meshes might be biased (oriented) to accomodate some criteria (e.g. given distortion of a parametric patch when generating mesh on a 3D surface).

3.3.1 Important notes

When the quadtree-based control is used, the input file must start by the `REGEXT` command. This is to ensure that the quadtree knows the extent to cover.

3.3.2 Isotropic mesh control

Constant mesh size

This input specification defines the constant value of the mesh size (i.e. triangle side length) over all the subregions. This constant value is blended with other mesh size "sources" (control points) to produce the resulting mesh size.

The syntax is:

```
MC-C  <FLOAT const_m_s>
```

Mesh control by refinement points

The command defines one mesh control point. The control point influences the resulting mesh size control function in the following way: By prescribing a mesh size value at a point, together with some circle in which we want to have this point "strong" influence, the program is able to merge all the defined control point influences (using global shape functions – see [2]) and to produce the mesh control function over the domain. The syntax is:

`MC-P <INT id> <FLOAT x  y  mesh_size  radius_of_influence>`

The arguments are:

`id`	Control point identifier. Unique number.
`x, y`	Coordinates of the control point.
`mesh_size`	Mesh size to be enforced at this point.
`radius_of_influence`	Radius of the circle in which we want to obtain the desired mesh size (approximately, note that there is interaction between the individual control points a consequently the resulting mesh size is in general not exactly what is prescribed).

Mesh size control function theory

The mesh size control function (*MSCF*) as generated by refinement points, is based on the following expressions: The mesh size at a given point $\mathbf{r}$, denoted as $h(\mathbf{r})$, is computed from the input data as

$$h(\mathbf{r}) = \begin{cases} \dfrac{\sum_{i=1}^{N_{cp}} h_i w_i \phi_i(|\mathbf{r}-\mathbf{r}_i|) + h_c}{\sum_{i=1}^{N_{cp}} w_i \phi_i(|\mathbf{r}-\mathbf{r}_i|) + 1} & \text{for } h_c \neq 0 \\ \dfrac{\sum_{i=1}^{N_{cp}} h_i w_i \phi_i(|\mathbf{r}-\mathbf{r}_i|)}{\sum_{i=1}^{N_{cp}} w_i \phi_i(|\mathbf{r}-\mathbf{r}_i|)} & \text{for } h_c = 0 \end{cases} \tag{3.1}$$

where $\mathbf{r}$ is the positional vector of the given point $\langle x; y\rangle$; $\mathbf{r}_i$ is the positional vector of the ith control point $\langle x_i; y_i\rangle$; h_i is the mesh size, associated to the ith control point; $\phi_i(|\mathbf{r}-\mathbf{r}_i|)$ is the shape function of the ith control point; h_c is the uniform mesh size specified for the whole region (given as $h_c = 0$ when it should not be considered); N_{cp} is the total number of control points defined; w_i is the weight of the ith control point. The shape function associated to the ith control point located at $\langle x_i; y_i\rangle$ is defined as

$$\phi_i(|\mathbf{r}-\mathbf{r}_i|) = \left[1 + \left(\frac{|\mathbf{r}-\mathbf{r}_i|}{R_i}\right)^{\mu_i} \left(\bar{\phi}^{-1} - 1\right)\right]^{-1}, \; 0 < \phi_i(r) \leq 1, \tag{3.2}$$

where μ_i is an arbitrary number, $\mu_i > 0$; R_i is a so-called *influence radius* and $\bar{\phi}$ an arbitrary number, $1 > \bar{\phi} > 0$.

To clarify the meaning of the parameters R_i and $\bar{\phi}$ let us consider (3.2) for $r = R_i$: By simple substitution we obtain $\phi(R_i) = \bar{\phi}$. It follows, that the influence radius R_i defines a circle around the ith control point such that

the shape function ϕ assumes within the circle values $\phi(r) \geq \bar{\phi}$. The value of $\bar{\phi}$ is probably best adjusted to use in manual *MSCF* specification by setting for instance $\bar{\phi} = 0.9$. The reason is, that in most cases the value of shape function being greater than 90% is sufficient to enforce the triangle size inside the influence circle as sufficiently close to what was desired. The task of controlling the mesh size thus becomes more intuitive. The specification of a circle and of the triangle size desired within the circle suffices to control the local mesh characteristics at points of interest. The interaction of the defined mesh control points takes care of the mesh size throughout the rest of the region to be covered by the triangulation.

The parameter w_i (weight ot the ith control point) needs to be discussed next. Due to the globality of the shape functions, the mesh size at the ith control point is in general *not* equal to the prescribed control parameter h_i, $h(x, y) \neq h_i$. The weights serve to reduce this discrepancy (it could make orders of magnitude). To illustrate the argument, a pair of control points is considered – first one with associated mesh size h_1, second one with mesh size $h_2 \ll h_1$. Let us assumed for simplicity, that $R_1 = R_2$ and $\mu_1 = \mu_2$. Then if the shape functions of both control points had the same weight, the larger mesh size h_1 would prevail even in the immediate neighbourhood of point $\langle x_2; y_2 \rangle$ as follows from Equation (3.2). The weight w_i of the ith control point is defined to give preference to the control points associated with smaller mesh size parameter h_k as

$$w_i = \frac{\max_{k=1,\cdots,n} h_k}{h_i}, \tag{3.3}$$

where $\max_{k=1,\cdots,n} h_k$ is the largest mesh size specified. In this way, even the control point with the smallest mesh size can enforce in its neighbourhood the desired triangle size with acceptable accuracy. Of course, it is also possible not to use the weights and to compute the control parameters from a system of linear equations, which could be obtained for instance fixing the mesh size at a number of points, or by least squares technique. This approach seems appropriate only for (semi)automatic input preparation, though.

Control parameters

As can be seen, the set of parameters $\mu_i,\ h_i,\ R_i$ varies the influence of the ith control point throughout the region. The meaning of parameters $h_i,\ R_i$ is fairly obvious, μ_i controls together with R_i how steep the gradation in mesh size is going to be. The author proposes to fix $\mu_i = 2$ for all control points – it corresponds approximately to maximal slope in mesh size control function acceptable for linear-shape-functions triangles – and to use only $h_i,\ R_i$ for mesh size control by the ith control point. Nevertheless, it might prove advantageous to set the parameter μ_i to different values at some control points – for instance to achieve steeper *MSCF* slope at some locations.

The operator of the triangulator applies the control parameters in the following way. First, the constant mesh size h_c might be specified. Its usefulness is twofold:

- It is the easiest way to specify mesh size in regions to be triangulated with uniform triangle size.
- When generating graded meshes, it is useful to specify the upper limit on triangle size that might be due e.g. to geometrical constraints.

Next, at each control point $\mathbf{r}_i$ the control parameters h_i, R_i (and μ_i, if desired) are prescribed:

- Mesh size h_i to specify the desired size at the point $\mathbf{r}_i$ and in its immediate neighbourhood,
- Influence radius R_i to specify the size of the neighbourhood of the point $\mathbf{r}_i$ in which the user wants the triangles to be approximately of the desired size h_i.
- Exponent μ_i to influence the slope in the *MSCF*.

Note, that the strategy for the placement of control points could use for instance heuristic information about singularity sources, preliminary computations or simply the user's decision.

Mesh control by refinement areas

The command below specifies a triangular refinement area. The effect of this command depends on what kind of mesh control implementation should be used: control-point based or quadtree based.

The mesh size as specified by linear interpolation on the triangle is transferred directly to the quadtree-based control function data structures. Therefore, the mesh size at a point within the triangle is given "exactly" by the interpolation from the vertex values.

For the control-point based mesh size function, the triangle is splitted into control points and their interaction leads to the final effect this refinement area will have.

The syntax:

```
MC-T <FLOAT x1  y1  x2  y2  x3  y3> <FLOAT ms1  ms2  ms3>
```

The arguments are:

`x1, y1, x2, y2, x3, y3`	Coordinates of the triangle vertices.
`ms1, ms2, ms3`	Mesh size at the vertices.

3.3.3 Anisotropic mesh control

Note that in order to use the anisotropic mesh control, the user must specify that the quadtree-based implementation of the control function must be used. The reason is, that only the quadtree-based function is able to store the additional mesh control parameters beyond mere mesh size.

The only way how to define anisotropic mesh control parameters is through a triangular control area. This may seem to be an unnecessary restriction, but remember that the preparation of practically useful anisotropic mesh gradation is best done by a tool, e.g. preprocessing program – manual approach would be very tedious at best.

Mesh control by refinement areas

The command below specifies a triangular mesh control area. Note that the quadtree based mesh function implementation is required to be able to use anisotropic mesh gradation. It is important to remember that the triangles must cover the whole region (they may overlap it in arbitrary way, but the whole domain to be triangulated must be covered) – otherwise the resulting mesh control function may show strange behaviour in areas not covered (the missing information is then deduced by averaging over neighboring quad-leaves).

The syntax:

```
MC-T <FLOAT x1  y1  x2  y2  x3  y3> <FLOAT ms1  ms2  ms3> --
     <FLOAT s1  s2  s3> <FLOAT a1  a2  a3>
```

The arguments are:

`x1, y1, x2, y2, x3, y3`	Coordinates of the triangle vertices.
`ms1, ms2, ms3`	Mesh size at the vertices. This specifies the prescribed length of a triangle side oriented parallel to lines inclined by angle `a1` (`a2`, `a3`) with respect to the x axis. Note that these sizes must be shorter than or equal to lengths in the conjugate direction – see below.
`s1, s2, s3`	Stretches at the vertices. The prescribed length of a triangle side perpendicular to lines inclined by angle `a1` (`a2`, `a3`) with respect to the x axis is computed as `ms1*s1` (`ms2*s2`, `ms3*s3`). Note, that it must always hold that $s1 \geq 1$, $s2 \geq 1$, $s3 \geq 1$.

3.4 Definition of boundary curves

The boundary curves are considered oriented entities. They are used in the definition of the subregions, so they must be oriented to specify the interior of the surrounded area.

3.4.1 Straight line segment

```
CURVE <INT id> LINE <FLOAT x1  y1  x2  y2>
```

The arguments are:

`id`	Identifier of the curve. The only requirement is that it must be storable in a `long int`, and that it must be unique among all the boundary curves that are defined by the input file.
`x1, y1, x2, y2`	Coordinates of its end-points (`x1`, `y1` is the start point). The line is oriented in the usual sense – from its start point to the other end.

3.4.2 Circle

```
CURVE <INT id> CIRCLE CENTER <FLOAT x  y> RADIUS <FLOAT r>
```

The arguments are:

`id`	Identifier of the curve. The only requirement is that it must be storable in a `long int`, and that it must be unique among all the boundary curves that are defined by the input file.
`x y`	Coordinates of the center. The circle is oriented always so that to travel along it in the positive sense, the rotation of the corresponding radial line is counterclockwise. Consequently, the following rule holds: to use a circle in the definition of the external boundary, use its identifier with plus sign (i.e. the circle is used in its inherent, positive, sense), to define a circular hole, write the identifier with a minus sign.
`r`	Radius of the circle.

3.4.3 Discretized curve

The discretized curve is simply a sequence of straight line segments. However, these segments are undivisible – they are not further subdivided into edges (in fact they *are* the edges of the triangulation). This boundary curve is useful when connecting together several subregions that are to be triangulated independently (e.g. a mapped mesh with free triangulation).

```
CURVE <INT id> DISCRETIZED <FLOAT x1  y1>...<FLOAT xN  yN>
```

The arguments are:

`id`	Identifier of the curve. The only requirement is that it must be storable in a `long int`, and that it must be unique among all the boundary curves that are defined by the input file.
`x1, y1`	Coordinates of the start point of the curve. The line is oriented in the usual sense – from its start point to the other end.
`xN, yN`	Coordinates of the Nth point of the curve (the last point).

3.4.4 Circular arc

The definition of a circular arc is rather tricky to describe. The syntax:

```
CURVE <INT id> ARC <FLOAT fx  fy  lx  ly> --
                   CENTER <FLOAT cx  cy> [REVERSED]
```

The arguments are:

`id`	Identifier of the curve. The only requirement is that it must be storable in a `long int`, and that it must be unique among all the boundary curves that are defined by the input file.
`fx, fy`	Coordinates of the start point of the arc. The arc is oriented from the start point to the end-point in such a manner, that to travel along it in the positive sense by rotating the line connecting the center with the start point into the line connecting the center with the end point, an angle less than π needs to be traversed.
`lx, ly`	Coordinates of the end point of the arc.
`cx, cy`	Coordinates of the center of the arc.

`[REVERSED]`	Optional keyword. If used, reverses the default sense as defined above, so that the arc is oriented in such manner that the angle mentioned is larger than π.

Important remark: There is a bug that was not yet attended to: The program does not handle definitions of half-circle arcs. The solution is to bias the center of the arc slightly (one-millionth of the radius is in most cases sufficient).

3.5 Definition of subregions

There is an important rule that must be followed in the definition of the boundary:

When you start from any point on the exterior boundary and follow the curves in the sense specified by the SUBREGION command (either positive, i.e. as they were defined, or negative, i.e. in the reversed direction with respect to their inherent orientation) you should travel counterclockwise and the start point has to be achieved in the end. On the other hand, to travel around a hole, you should travel clockwise.

Syntax:

```
SUBREGION <INT id> PROPERTY <INT prop> [MATERIAL <INT mat>] --
   [SUBSOIL <INT subs>] [THICKNESS <FLOAT th>] --
   [GROUP <STRING grp>] --
   BOUNDARY <INT-LIST ext> --
   [HOLE <INT-LIST hole1> [...HOLE <INT-LIST holeN>]]
```

The arguments are:

`id`	Identifier of the subregion. The only requirement is that it must be storable in a `long int`, and that it must be unique among all the subregions that are defined by the input file.
`prop`	Property identifier. It is written into the output file, if the output format is `AFRO` (=1).
`mat`	Optional material identifier. It is written into the output file, if the output format is `AFRO` (=1).
`subs`	Optional sub-soil identifier. It is written into the output file, if the output format is `AFRO` (=1).
`th`	Optional thickness value. It is written into the output file, if the output format is `AFRO` (=1).

`grp` Optional group name. It is written into the output file, if the output format is `AFRO` (=1).

`ext` List of signed curve identifiers in the order in which they define the external boundary of the subregion (see the rule above).

`hole1,...,holeN` Optional lists of signed curve identifiers in the order in which they define the holes within the subregion. Note, that any subregions enclosed in the subregion being defined are considered holes with respect to it.

property identifier

3.5.1 Note on fixed curves inside the domain

The domain to be triangulated might contain some curves, that should be included in the final triangulation in that it should be covered by edges without any topological holes (i.e. without any edge crossing the curve).

Such curves are easily incorporated by the following trick. The fixed curve may be imagined as a hole with zero area, i.e. the boundary of the hole is created of curves that are traversed in both senses. Simple example is a fixed straight line segment (say with identifier equal to 13). The (fictitious) hole is then defined as

```
subregion 1  property 1 boundary 1 2 -3 hole 13 -13
```

The resulting triangulation then respects the line segment as a fixed curve.

3.5.2 `REGEXT` command

To be able to use the quadtree-based mesh control, the input file *must* start by the `REGEXT` command. This is to ensure that the program knows which area should be covered by the quadtree. The `REGEXT` command defines a rectangle which contains completely all of the regions to be triangulated.

```
REGEXT  minx miny maxx maxy
```

The arguments are:

`minx,miny` Coordinates of the lower-left corner of the rectangle covering all of the regions to be triangulated.

`maxx,maxy` Coordinates of the upper-right corner of the rectangle covering all of the regions to be triangulated.

3.6 Definition of fixed points

In some cases it might be necessary to define fixed points inside the domain to be triangulated. These points need to be respected in that there should a vertex with each one of them. This feature is rather fragile with respect to the mesh size definition – it is easy to generate badly shaped triangles or even break down the mesh generation by setting unrealistic mesh size with respect to the (relative) locations of the fixed points.

```
FIXED <INT id> XY <FLOAT x  y> [CURVE <INT cid>]
```

The arguments are:

`id`	Identifier of the fixed point. The only requirement is that it must be storable in a `long int`, and that it must be unique among all the fixed points that are defined by the input file.
`x y`	Coordinates of the fixed point.
`cid`	Optional curve identifier. This let you include cases, where the fixed point lies on a curve. This case would be handled probably incorrectly when the program was not told about this circumstance.

3.7 Annotated example of input file

The following is a simple region definition. It is a triangle, with two fixed points and one fixed curve inside. The curve is used to define a fictitious hole, as described above.

The Fig. 3.7 shows the resulting triangulation.

```
Mc-c 0.9                  !  uniform mesh
Curve 1 line 0 3 -3 0
curve 2 arc 3 0 -3 0 center 0 2
curve 3 line 3 0 0 3
CURVE 4  line 0 2 -1.9 .7 ! fixed curve
Subregion 1  property 1 boundary 1 -2 3 -- ! split line
! fictitious
      hole 4 -4
fixed 338 xy -0.3 -0.2
FIXED 339 xy  0.3  0.4
```

Figure 3.1: Input to generate the sample mesh below.

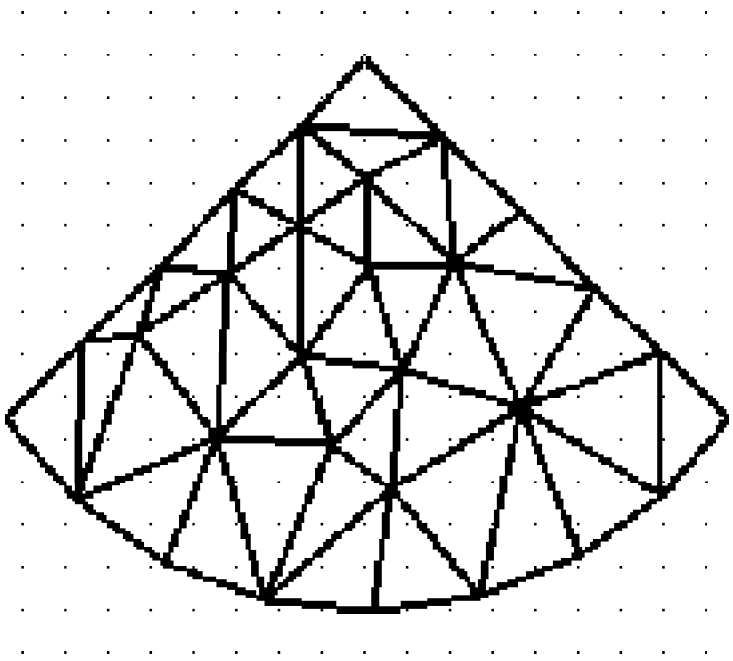

Figure 3.2: Mesh generated from the input.

Chapter 4

Output formats

There are two output formats (i) AFRO, (ii) MOS. The former was designed for use with the commercial package FEAT (of SmartSoft Ltd., Prague), the latter was designed for use with the accompanying program for generation of triangulations on 3D surface patches.

4.1 Output for AFRO

The output file is ASCII and has this format:

- First line: name of the output file.
- Second line: Two numbers – total number of vertices in the file, i.e. number of lines defining nodes of the mesh, and total number of triangles in the file.
- Third line contains only the string "*NODES".
- Then follows one line per each node with node number and its coordinates x, y.
- Next line contains the string "*TR3N".
- Follows one line per each triangle with triangle number, number of its defining nodes, material identifier, thickness, sib-soil identifier and the group string.

4.2 Output for MOS

This is still subject to modification. Best way how to find out what it actually contains is to run `targe2` for one sample and to inspect its output (do not forget to set option `-f 2`).

Chapter 5

Errors

The error reporting system is bound to undergo systematic overhaul soon. But there is the current assortment of error messages.

5.1 Error messages

Too large number of cells in the cell grid.

The number of cells in the cell grid is much too large to be represented by an integer type in the computer (see reference [1]). Repair work is in progress, but no help right now.

Parameter MAX_FRONT_LISTS must be increased.

The difference between the shortest and longest edge is too large. The internal auxilliary array overflowed. This should not occurr, but in very exceptional cases recompilation might help (increase `MAX_FRONT_LISTS` in "consts.h" parameter).

Too many tokens in input line.

What it says. Increase `MAX_TOKENS` in "ttypes.h" and recompile `targe2` .

No mesh control information.

No mesh control defined yet.

Information on mesh size demanded when there was none yet. Rewrite or complete the input file.

Invalid anisotropic multiplier.

Stretch less than one. See syntax above.

Triangle stream could not be opened.

Error reading triangle stream.

Problems with unformatted triangle stream. Non-existing, unreadable etc.

Too numerous connectivity.

Too many edges connected to a single vertex. This is currently fixed internal parameter. One way out to modify the input requirements. Governed by the parameter `MAX_CONNECTIVITY` in "ttypes.h".

Fixed-point list non-empty.

The triangulation was finished and there still remained some fixed points. It means, that the mesh is not valid. This should be rather rare exception.

No leaf in quadtree at this location.

The mesh control information was requested outside the area of the root of the quadtree. This can happen when the user sets the domain extent incorrectly by the `REGEXT` command.

Last resort failed.

Bad luck. The generator was given input specification that lead it to unsolvable deadlock. It might be also the failure of the author to find the correct answer to the problem. Let the author know of this (send the input file and the parameters, with which the program has been run).

5.2 Bug reports

Send all bug reports at the email address "pk@power2.fsv.cvut.cz". Please, include all the default settings (`targe2` prints them when run without parameters), with the parameters that you used, and with the input file.

Bibliography

[1] Krysl P., "Computational Complexity of the Advancing Front Triangulation", to appear in *Engineering with Computers* (1995).

[2] Rypl D. and P. Krysl, "Triangulation of 3D Surfaces", submitted to *Engineering with Computers* (1995).

Contents