Weierstraß-Institut
für Angewandte Analysis und Stochastik

im Forschungsverbund Berlin e.V.

Technical Report ISSN 1618 – 7776

TetGen
A 3D Delaunay Tetrahedral Mesh Generator
Version 1.2 User's Manual

Hang Si[1]

submitted: December 20th 2002

[1] Weierstrass Institute for Applied Analysis and Stochastics
Mohrenstrasse 39, D - 10117 Berlin
email: si@wias-berlin.de

No. 4
Berlin 2002

2000 *Mathematics Subject Classification.* 65N50, 65U0.

Key words and phrases. 3D Mesh generation, Delaunay property.

Edited by
Weierstraß-Institut für Angewandte Analysis und Stochastik (WIAS)
Mohrenstraße 39
D — 10117 Berlin
Germany

Fax: + 49 30 2044975
E-Mail: preprint@wias-berlin.de
World Wide Web: http://www.wias-berlin.de/

Abstract

This technical report describes the main features and the using of TetGen, a 3D Delaunay tetrahedral mesh generator. Based on the most recent developments in mesh generation algorithms, this program has been specifically designed to fulfill the task of automatically generating high quality tetrahedral meshes, which are suitable for scientific computing using numerical methods such as finite element and finite volume methods.

In this document, the user will learn how to create 3D tetrahedral meshes using TetGen's input files and command line switches. Various examples were given for better understanding.

This document describes the features of the version 1.2.

Contents

1 Introduction

1.1 What is TetGen?

TetGen is a stand-alone program for 3D Delaunay tetrahedral mesh generation. It generates the Delaunay tetrahedrization for 3D point sets, the constrained Delaunay tetrahedrization, and the conforming Delaunay tetrahedrization for piecewise linear complexes. One of the main purposes of TetGen is to generate high quality tetrahedral meshes which are suitable for scientific computing using numerical methods such as finite element and finite volume methods.

TetGen is also a small library that can be embedded into the application, if one wants to create 3D tetrahedral meshes within the application.

The TetGen library consists of a mesh data structure and a set of mesh manipulating primitives and routines, which are specifically designed to implement algorithms for 3D tetrahedral mesh generation. The goals of the present implementation are efficiency and stability. TetGen is written in C++ and uses the standard library. It was tested on 32-bit and 64-bit computers using an ANSI C++ compiler to compile it.

1.2 Features

TetGen is specialized for generating 3D Delaunay tetrahedral meshes, but is also suitable for some related tasks such as computing convex hull and Delaunay tetrahedrization for 3D point sets. The input of TetGen can be:
– a 3D point set, hence only a list of points or
– a piecewise linear complex (PLC), which is a list of points plus a list of facets.

For a 3D point set, TetGen generates its Delaunay tetrahedrization or convex hull. Figure 1 shows a sample point set of a teapot and its Delaunay tetrahedrization.

For a 3D piecewise linear complex, TetGen generates its boundary constrained Delaunay tetrahedrization and its quality conforming Delaunay mesh. The latter contains well shaped elements except slivers, it can be used for finite element and finite volume analysis.

Figure 2 provides an example of a piecewise linear complex modeling a mechanical part and its boundary constrained Delaunay tetrahedrization.

Normally, the quality (or shape) of elements of a boundary constrained Delaunay te-

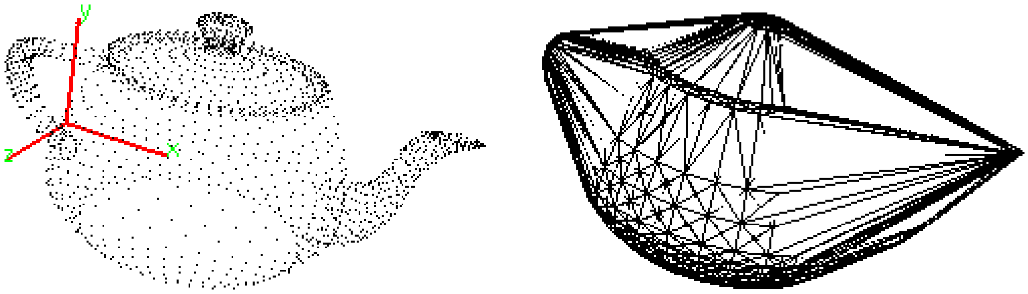

Figure 1: A 3D point set (left) and its Delaunay tetrahedrization (right), here only the convex hull faces are visible.

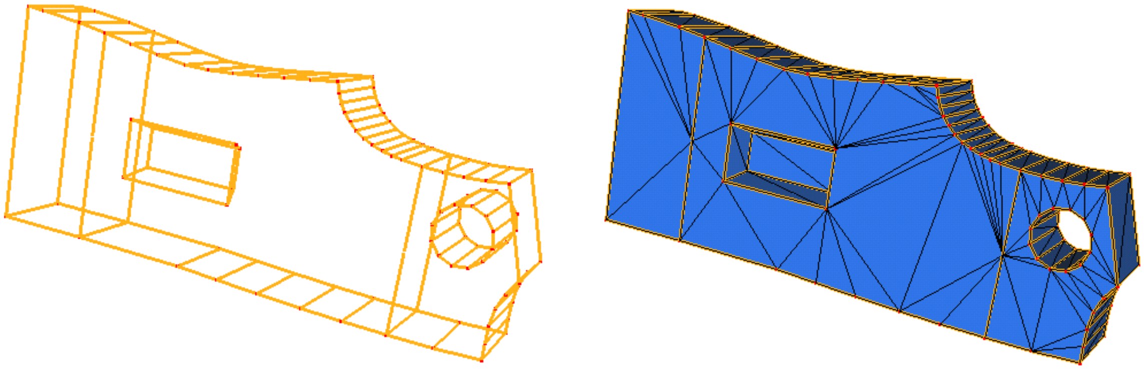

Figure 2: A piecewise linear complex (left) and its boundary constrained Delaunay tetrahedrization (right).

trahedrization is not sufficient for use within finite element or finite volume methods. TetGen is able to eliminate badly shaped elements from the mesh and replaces them with well shaped ones. This is done by inserting additional points into the current mesh. TetGen stops to add points when all the elements are satisfying a default or a user-specified quality bound, and the result is a conforming Delaunay tetrahedral mesh. Figure 3 shows the refinement results using different quality bounds for the boundary constrained Delaunay mesh of Figure 2.

In addition, TetGen can generate a quality report if the "-V" switch is used in the command line. Table 1 presents a sample output of the quality report for the mesh shown in Figure 3.

```
Mesh quality statistics:

  Smallest volume:         0.019848   |  Largest volume:           37.042
  Shortest edge:            0.70981   |  Longest edge:             7.9102
```

```
Smallest dihedral:          1.172   |  Largest dihedral:         178.04

Aspect ratio histogram:
1.1547 - 1.5       :         0    |      15 - 25         :         26
   1.5 - 2         :         0    |      25 - 50         :          5
     2 - 2.5       :         0    |      50 - 100        :          4
   2.5 - 3         :         0    |     100 - 300        :          1
     3 - 4         :      1927    |     300 - 1000       :          0
     4 - 6         :      1548    |    1000 - 10000      :          0
     6 - 10        :       241    |   10000 - 100000     :          0
    10 - 15        :        40    | 100000 -             :          0
(Tetrahedron's aspect ratio is radius of circumsphere divided
  by radius of inscribedsphere)

Dihedral Angle histogram:
      0 -  10 degrees:       100    |     90 - 100 degrees:      1924
     10 -  20 degrees:       285    |    100 - 110 degrees:      1266
     20 -  30 degrees:       775    |    110 - 120 degrees:       828
     30 -  40 degrees:      1805    |    120 - 130 degrees:       520
     40 -  50 degrees:      2966    |    130 - 140 degrees:       294
     50 -  60 degrees:      3523    |    140 - 150 degrees:       193
     60 -  70 degrees:      3367    |    150 - 160 degrees:       104
     70 -  80 degrees:      2672    |    160 - 170 degrees:        63
     80 -  90 degrees:      2047    |    170 - 180 degrees:        20

Shape histogram:

  Sliver:  29
  Needle:  0
  Spindle: 0
  Wedge:   49
  Cap:     0

There are 78 bad elements among 3792 elements.
```

Table 1: Example of quality report.

The mesh may have to describe various materials, too. TetGen allows the user to define regions in the input file. Each region is bounded by a closed set of facets, and carries a region number (integer) and a region attribute (float). While the region number is used to indicate which material is assigned to this region, different regions can have the same region number. That means they are made up by the same material. In Figure 4 the left picture shows the mechanical part containing three different regions, shown in different colors. The region attribute is used to impose a maximum volume constraint on all elements of this region. This is very useful when one needs a non-uniform resolution of the mesh. In Figure 4, the right picture shows a non-uniform mesh achieved by using the region attributes.

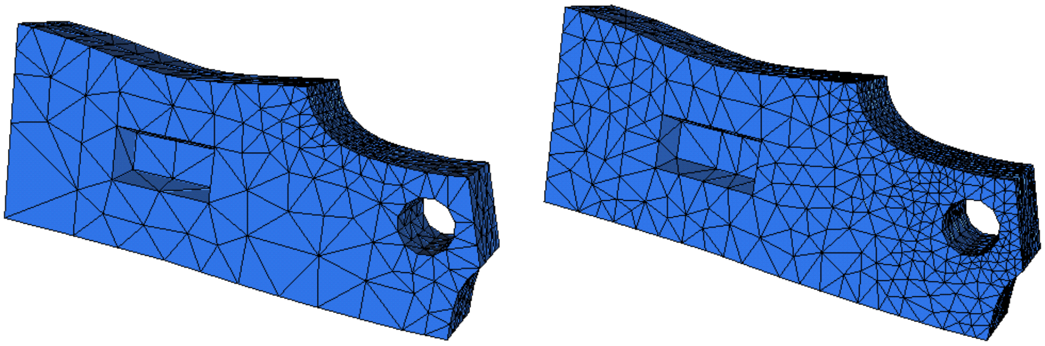

Figure 3: Quality Delaunay tetrahedral mesh generation using different quality bounds. The left one has a lower quality bound than the right one, which needs much more inserted points.

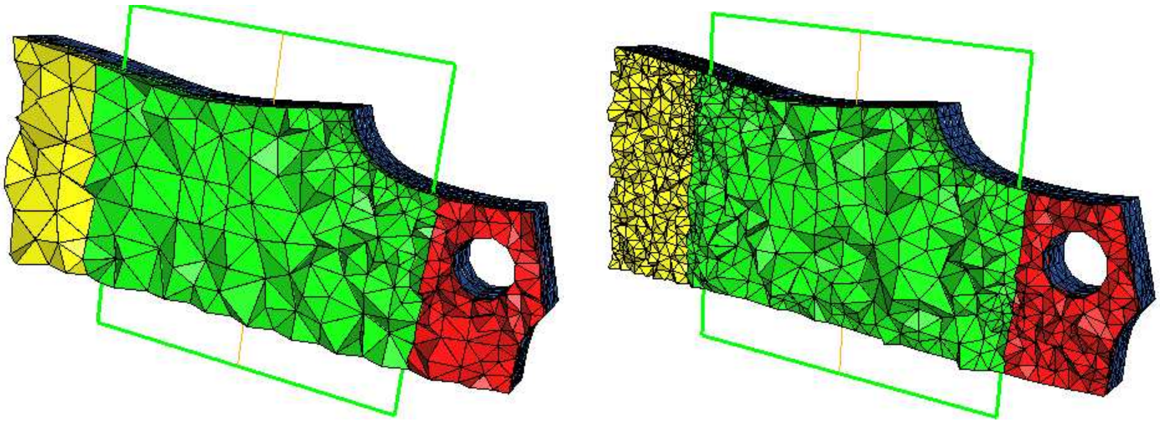

Figure 4: Illustration of the effect of the region number and region attribute. In the left picture three different regions are defined, shown in different colors, the right one shows the result of using region attributes to achieve a non-uniform mesh.

For convinience TetGen inherited many command line switches from `triangle` [1], which is a very successful 2D Delaunay mesh generator written by Jonathan Shewchuk. Although it is a 2D program, many command line switches can be straightforwardly extended to 3D, switches like "-p, -q, -A, -a, -V ..." have almost the same meaning, and the style of using command line switches is same in both programs.

The input/output files of TetGen are simple and flexible. TetGen uses ASCII files as inputs and outputs, all files have a simple and easy to understand formats. The input consists of one or two files: a list of nodes (.node), and a list of facets (.poly) or a list describing a surface mesh (.smesh). The output files are optional. By default the output is devided into a node file (.node) and an element file (.ele). Optionally

[1] Available at http://www-2.cs.cmu.edu/ quake/triangle.html

TetGen creates a face file (.face) and a neighbor file (.neigh).

For direct incorporation in other applications TetGen provides an programming interface, too. This feature is particular useful for FEM or FVM programs. A structure called 'tetgenio' (similar to the 'triangleio' structure of `triangle`) is provided: it stores the data passied to and from TetGen without involving any file.

Finally some typical execution times are given. 3D point sets are Delaunay tetrahedrized. This allows to compare the results with `Qhull`[2]. `Qhull` is a public domain program aiming at the computation of convex hulls for point sets in high dimensions. Differently sized data sets (from $10,000$ to $1,000,000$ points) are used to compare both programs. Table 2 summarizes the results.

Size of point set	10k	20k	50k	100k	200k	500k	1M
Qhull (time/sec.)	1.55	3.78	9.92	19.85	97.31	265.9	730.9
TetGen (time/sec.)	1.23	3.13	9.33	31.85	65.68	195.69	840.72
Qhull (Memory/MB)						1249	2370
TetGen (Memory/MB)						284	435

Table 2: Typical Delaunay tetrahedrization run times for Qhull3.0 and TetGen1.2. The point sets used in this test are randomly distributed points in the unit cube (generated by `robx`, a companion program of `Qhull`). This test was performed on a Compaq GS80, ev67, 731GHz, 8GB Memory.

[2] Available at http://www.geom.umn.edu/software/qhull

2 Compilation

The source code of TetGen is distributed via `http://tetgen.berlios.de`. There are two types of the package, one (tetgen*.*.zip) was created by WinZip32 for MS Windows systems, and the other (tetgen*.*.tar.gz) is a gzipped tar file for Linux/Unix systems. *.* stands for the current version number. To uncompress the archive file into a directory, for example, in Linux/Unix, you can type the following commands:

```
- gunzip tetgen*.*.tar.gz
- tar xvf tetgen*.*.tar
```

The distributed files of TetGen are listed below:

```
- README.TXT      general information.
- LICENCE         licence information.
- bin/            Directory containing the executable
                  file after compilation.
- data/           Directory with examples.
- src/            Directory with the C++ source code.
```

An ANSI C++ compiler is needed to compile the source code.

2.1 In Linux/Unix System

The easiest way to compile TetGen is to use `make`, a "makefile" is included in directory "src". Using make requires:

```
- cd src
- make
```

This will build an executable file "tetgen" in directory "bin" and a library file "libtet.a" in directory "src".

Any problem? Open the "makefile" and check the CC, CSWITCHES. The default compiler for CC is gcc. gcc must be replaced by the compiler available on the given system.

The default optimization switch for SWITCHES is -O, one should try O2, O3, ... to find the best optimization level.

2.2 In Win98/NT/2000 System

TetGen compiles as a console program "tetgen.exe" and a library "libtet.lib" on Win32 systems. It was successfully compiled with Borland C++ 5.5 and Visual C++ 6.0. The easiest way to compile TetGen on Win32 systems is to use the IDE of Borland C++ 5.0 (or later) or Visual C++ 6.0. The project files for building TetGen are included in directory "src/win32". One can load the corresponding project files at this directory and build it directly.

3 Using TetGen

When TetGen is used as a stand-alone program, it is invoked from the command line with a set of switches and a input file name. Switches are used to control the behavior of TetGen and specify the output files. In correspondence to the different switches, TetGen will generate the Delaunay tetrahedrization, the constrained Delaunay tetrahedrization or a conforming Delaunay mesh. The output will be written to the already mentioned files.

3.1 Command Line

The command syntax is:

```
tetgen [-pq__a__AfcngzBPNEIOCQVh] input_file
```

Underscores indicate: numbers may optionally follow certain switches, spaces are not permitted within the string of switches. The input file extension has to be .node or .poly or .smesh in case the -p switch is used.

3.2 Switches

It follows an overview of all the switches and a complete description. Invoking TetGen without switches and input_file prints on the screen:

```
-p  Triangulates a Piecewise Linear Complex.
-q  Quality mesh generation.
-A  Assign a region number and an attribute to each element.
-a  Applies a maximum tetrahedron volume constraint.
-f  Generates a list of tetrahedron faces.
-e  Generates a list of segments.
-c  Generates a list of convex hull or outer boundary faces.
-n  Generates a list of tetrahedron neighbors.
-g  Generates files for viewing mesh by other mesh viewer.
-z  Numbers all items starting from zero (rather than one).
-B  Suppresses output of boundary information.
-N  Suppresses output of .node file.
-E  Suppresses output of .ele file.
-I  Suppresses mesh iteration numbers.
-O  Ignores holes in .poly file.
-C  Check consistency of final mesh.
-Q  Quiet:  No terminal output except errors.
-V  Verbose:  Detailed information on what I'm doing.
-h  Help:  Detailed instructions for TetGen.
```

The -p Switch

The -p switch is used to generate a constrained Delaunay tetrahedrization for a piecewise linear complex. It performs the following jobs for you:

- Reads a file (.poly or .smesh) describing a piecewise linear complex, which can specify points, facets, holes and regions.
- Will generate a constrained Delaunay tetrahedrization fitting the input.
- If -p is not used, TetGen reads a .node file by default, and generates a Delaunay tetrahedrization for that point set.

The -q Switch

The -q switch is used to generate a quality conforming Delaunay mesh for piecewise linear complex. It controls the mesh quality. The resulting mesh contains well shaped elements measured by a default or user specified quality measure bound.

- It uses the circumcenter insertion algorithm of Ruppert and Shewchuk to refine a boundary constrained Delaunay mesh.
- The quality measure it uses is called Radius-Edge Ratio r/l, for each tetrahedron, the measure is a value of ratio between the radius r of its circumsphere to the length l of its shortest edge. This ratio is minimized by the regular tetrahedron $r/l = \sqrt{3/8} \approx 0.612$, but such measure is hard to reach in general cases. The default quality bound in TetGen is 2.0, TetGen will stop to add points to the mesh when every refinable element has a quality lower or equal to this value.
- An alternative quality bound can be specified after -q switch. For instance, -q1.414 specifies a quality bound $r/l \approx \sqrt{2}$. TetGen will not terminate for very small bounds. In practice, the algorithms used in TetGen work often successfully down to 1.1 – but they may not terminate.
- If the -q switch is used, the -p switch is automatically applied.

The -A Switch

The -A switch is used to assign a region number and a region attribute to each element. The region number identifies the material the element is mapped to, and the region attribute will impose a maximum volume constraint on all elements belonging to this region.

- When this switch is used, the "region" section defined in the .poly or the .smesh file will be read. The elements will be classified with respect to the facet bounded regions.

- If a region is not explicitly defined in the .poly or the .smesh file, its elements belong to the default region number zero with the attribute zero.
- This switch is in effect only together with the -p or the -q switch.

The -a Switch

The -a switch is used to apply a maximum tetrahedron volume constraint to all or a part of the tetrahedra. While the -q switch controls the mesh quality, this switch is used to control the mesh element size.

- If a number follows -a, it will impose a fixed volume constraint on all elements of the mesh, it means, no tetrahedron will exist in final mesh whose volume is larger than that number.
- If no number is specified, volume constraints will be read from the "region" section of the .poly or the .smesh file. Each facet-bounded region optionally contains a volume constraint, thereby enforcing a minimal tetrahedron density in that region.
- It is possible to impose both a fixed volume constraint and a varying volume constraint by invoking -a twice, once with and once without a follow on number.

The -V Switch

The -V switch specifies a verbose model when TetGen is running. More 'V's are increasing the amount of details. The main use is related to algorithm and program debugging. It provides a statistics on mesh quality and of dynamic memory usage, too. See table 1 for an example of a quality report.

The -g Switch

The -g switch is used to view results using other mesh viewing programs (Medit, Gid, and Geomview), which are publicly available. This helps to understand the mesh output. These programs are found on the websites:

```
- Medit    <http://www-rocq.inria.fr/gamma/medit>
- Gid      <http://gid.cimne.upc.es>
- Geomview <http://geomview.org>
```

With the -g switch in the command line, TetGen will generate the following additional files:

```
- *.mesh,  read by Medit;
- *.ele.gid, *.face.gid, imported by Gid;
- *.off, read by Geomview;
```

For example, the pictures of result meshes in this manual have been created by Medit.

3.2.1 Examples for Using Switches

- Example 1: **tetgen dots**
  Reads points from file the “dots.node”, computes a Delaunay tetrahedrization from this point set and writes the results to the files “dots.1.node” and “dots.1.ele ”. (“dots.1.node” is identical to “dots.node”.)
  tetgen -I dots writes the results to the file “dots.ele” instead. (No additional .node file is needed, hence it is not written.)
  tetgen -c dots also writes the convex hull faces to file “dots.1.face”.

- Example 2: **tetgen -p object**
  Reads a Piecewise Linear Complex (PLC) from file “object.poly” or “object.smesh” (and possibly “object.node”, if the points are omitted from file “object.poly” or “object.smesh”), computes a boundary constrained Delaunay tetrahedrization of this PLC and writes results to the files “object.1.node” and “object.1.ele”.

- Example 3: **tetgen -pq1.414a0.5 object**
  Reads a Piecewise Linear Complex (PLC) from the file “object.poly” or “object.smesh”, (and possibly “object.node”), generates a conforming Delaunay mesh for this PLC. Tetrahedra all have a quality measure (Radius-Edge Ratio) smaller than 1.414 and volume smaller than 0.5. Writes the mesh to the files “object.1.node” and “object.1.ele”.

3.3 File Formats

All files may contain comments prefixed by the character '#'. Points, tetrahedra, faces, holes, and regions must be numbered consecutively, starting from 0 or 1. All input files must be consistent, for instance, if the points are numbered from 1, so must be all other objects, like tetrahedra, faces, etc. TetGen automatically detects the used convention while reading the .node (or .poly or .smesh) file. When calling TetGen from another program, the -z switch specifies numbering objects from zero (default: counting starts at 1).

Remark: in the following description '#' stands for 'number' – it should not cause confusion with the comment prefix.

3.3.1 .node file

- First line: <# of points> <dimension (must be 3)> <# of attributes> <boundary markers (0 or 1)>
- Remaining line: <point #> <x> <y> <z> [attributes] [boundary marker]

The '.node' file contains a list of points. Each point contains its coordinates, attributes and optionally a boundary marker. It is used as both input and output files.

The attribute of each point is copied unchanged to the output mesh. If -q or -a is selected, each new Steiner point added to the mesh will have an attribute assigned to it by linear interpolation.

If the fourth entry of the first line is '1', the last column of the remainder of the file is assumed to contain boundary markers. Boundary markers are used to identify boundary points and points resting on segments or facets a of piecewise linear complex. The .node file produced by TetGen will contain boundary markers in the last column unless they are suppressed by the -B switch.

3.3.2 .poly file

- One line: <# of points> <dimension (must be 3)> <# of attribute> <boundary markers (0 or 1)>
- Following lines: <point #> <x> <y> <z> [attributes] [boundary marker]
- One line: <# of facets> <# of boundary markers>
- Following lines: (listing all facets data, see below)
- One line: <# of holes>
- Following lines: <hole #> <x> <y> <z>
- Optional line: <# of region>
- Optional following lines: <region #> <x> <y> <z> <region number> <region attribute>

For each facet, the format is:

- One line: <# of polygons> [# of holes] [boundary marker (0 or 1)]
- Following lines: <# of vertices> <point1> <point2> ... <point#>
- Following lines: <hole #> <x> <y> <z>

The '.poly' file represents a Piecewise Linear Complex (PLC), as well as some additional information. It is used as input file. The '.smesh' file is also used to represent a PLC and its format is much simpler than that of the '.poly' file, but is not as flexible as the .poly format.

The '.poly' file consists of four sections: the points, the facets, the holes, and the region section. The region section is optional. These sections are introduced below.

The first section lists all the points – it is identical to the format of the '.node' file. <# of points> may be set to zero to indicate that the points are listed in a seperated .node file. This has the advantage that a point set of this PLC can be easily be tetrahedralized with or without facet information.

The second section lists all the facets of the PLC. A facet is a planar boundary, however, it can be of quiet complicated shape. It may be non-convex, may have any number of sides, and may consist of holes, slits or vertices, too. A strict requirement is: all points of this facet must be coplanar. Figure 5 shows a simple facet and a more elaborated one.

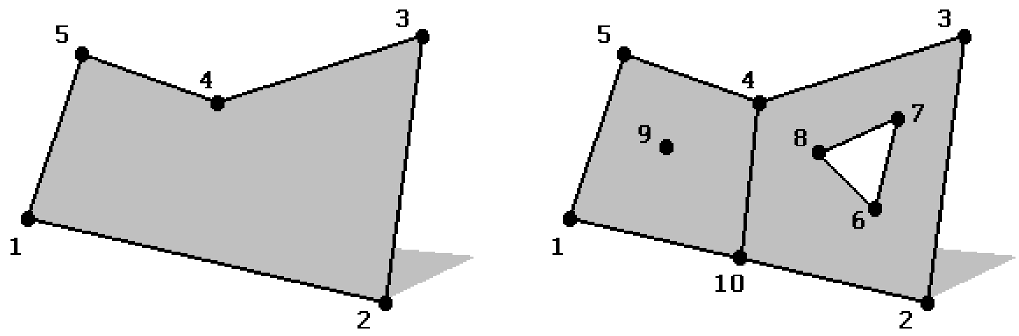

Figure 5: Examples of facets. One is simple in geometry (left), it is formed by a non-convex polygon with 5 vertices. The other is a little bit more complicated (right), it is formed by the same polygon as the left one, but additionally exist in the facet: a hole, a segment, and a point.

In the .poly file, each facet is represented by a set of cloosed polygons and holes definitions. The first line of each facet specifies the number of polygons, number of holes and a boundary marker. If no hole number and boundary marker is provieded, TetGen assumes there is no hole in this facet and the default boundary marker (0) is assigned to this facet.

For each facet, the set of polygons has to be be listed first. Each polygon is specified by giving the number of vertices in the polygon, followed by the ordered set of point indices. This list can be distributed over a number of lines. A single segment can be represented by a degenerate polygon with only two endpoints. A degenerate segment with two identical endpoints represents an isolated point. After listing all polygons holes can be specified by identifying a point inside each hole (this point describes the hole properly, as long as the orthogonal projection of this point onto the facet is inside the hole).

The following example describes the left facet in Figure 5. This facet contains 1 polygon, no hole, no boundary marker.

```
...
1  0  0
5  1 2 3 4 5  # Lists the polygon.
...
```

Now follows the description of the right facet in Figure 5. This facet contains 4 polygons, 1 hole, no boundary marker.

```
...
4  1  0
5  1 2 3 4 5  # The outer polygon.
3  6 7 8      # The inner polygon.
2  4 10       # A segment.
2  9 9        # A point.
1  x y z      # The hole point.
...
```

The description is not unique. Another valid possibility is given by:

```
...
4  1  0
4  1 10 4 5   # The left polygon.
4  10 2 3 4   # The right polygon.
3  6 7 8      # The inner polygon.
2  9 9        # A point.
1  x y z      # The hole point.
...
```

The third section lists holes in the tetrahedrization. Holes are specified by identifying a point inside each hole. After the Delaunay tetrahedrization is formed, TetGen creates holes by removing tetrahedra. The start is defined by the 'hole point'. PLC facets block the procedure. In case of non closes PLC facets whole tetrahedrization may be 'eaten' away. If two tetrahedra abutting a subface are removed, the subface itself is also ate away. Hole points have to be placed inside a region, else the rounding error determines which side of the facet is being transformed into the hole.

The optional fourth section lists regional attributes (to be assigned to all tetrahedra in a region) and regional constraints on the maximum tetrahedron volume. TetGen will read this section only if the -A switch is used or the -a switch without a number is invoked. Regional attributes and volume constraints are propagated in the same manner as holes. If two values are written on a line after the x, y and z coordinate, the former is assumed to be a regional attribute (but will only be applied if the -A switch is selected), and the latter is assumed to be a regional volume constraint (but will only be applied if the -a switch is selected). It is possible to specify just one value after the coordinates. It can serve as both an attribute and an volume constraint, depending on the choice of switches. A negative maximum volume constraint allows to use the -A and the -a switches without imposing a volume constraint in this specific region.

3.3.3 .smesh file

- One line: <# of points> <dimension (must be 3)> <# of attributes> <boundary markers (0 or 1)>
- Following lines: <point #> <x> <y> <z> [attributes] [boundary marker]
- One line: <# of facets> <boundary markers (0 or 1)>
- Following lines: <# of vertices> <point1> <point2> ... <point#> [boundary marke]
- One line: <# of holes>
- Following lines: <hole #> <x> <y> <z>
- Optional line: <# of region>
- Optional following lines: <region #> <x> <y> <z> <region number> <region attribute>

The '.smesh' file represents a special kind of PLC - the surface mesh. It consists of four sections: point, facet, hole, and region section. Only the facet section's format is different from the already defined .poly format. The facet section is a simplified format of .poly file. Each facet contains only one polygon and no hole in it. This is particularly convenient when the surface mesh is created by other programs.

For example, in Figure 5, the left facet can be described in a .smesh file as below:

```
...
5  1 2 3 4 5
...
```

Nevertheless, the right facet in figure 5 can not be described by the .smesh file format, because it contains a hole.

3.3.4 .ele file

- First line: <# of tetrahedra> <points per tetrahedron> <# of attributes>
- Remaining line: <tetrahedron #> <point> <point> <point> <point> [attributes]

The '.ele' file contains a list of tetrahedra. Each tetrahedra contains of its four corner points, defined by the indices into the corresponding .node file. Optionally it contains one or more attributes. The attributes are just like those of .node files. It is used as output file.

3.3.5 .face file

- First line: <# of faces> <boundary markers (0 or 1)>
- Following lines: <face #> <point> <point> <point> [boundary marker]

The '.face' file contains a list of faces. Each face contains the three corner points, given by indices into the corresponding .node file. The optional column of boundary markers is suppressed by the -B switch. It is used as output file.

3.3.6 .neigh file

- First line: <# of tetrahedra> <# of neighbors per tetrahedron (always 4)>
- Following lines: <tet #> <neighbor> <neighbor> <neighbor> <neighbor>

Neighbors are indices with respect to the corresponding .ele file. An index of -1 indicates a mesh boundary, hence no neighbor. TetGen can produce .neigh files (use the -n switch), but cannot read them. It is used as output file.

3.3.7 Example of File Formats

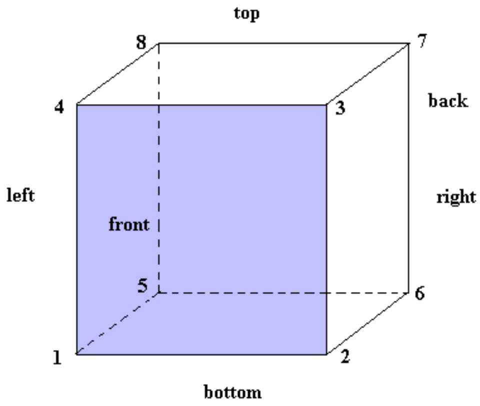

Figure 6: The geometry of a cube of size $10 \times 10 \times 10$.

.poly and .smesh File "cube.poly" listed below describes a cube of size $10 \times 10 \times 10$. Figure 6 shows the related point numbering.

```
# cube.poly file

# Nodes section
#  8 points in 3D, no attributes, no boundary marker.
8       3  0  0
1     0   0  10
2    10   0  10
3    10  10  10
4     0  10  10
5     0   0   0
6    10   0   0
7    10  10   0
8     0  10   0

# Facet section
#  6 facets, no boundary markers
6       0
# For each facet, there only have 1 polygon,
# no holes and no boundary marker
1                  # front
4    1  2  3  4
1                  # back
4    5  6  7  8
1                  # bottom
4    1  2  6  5
1                  # top
4    4  3  7  8
```

```
1                  # left
4    1  4  8  5
1                  # right
4    2  3  7  6

# Holes section
#  There is no hole in cube
0

# Region section
#  No region be defined
0
```

A second input possibility is demonstrated by the file "cube.smesh":

```
# cube.smesh file

# Nodes section
#  8 points in 3D, no attributes, no boundary marker.
8       3  0  0
1     0   0  10
2    10   0  10
3    10  10  10
4     0  10  10
5     0   0   0
6    10   0   0
7    10  10   0
8     0  10   0

# Facet section
#  6 facets, no boundary markers
6       0
# List each facet.
4    1  2  3  4    # front
4    5  6  7  8    # back
4    1  2  6  5    # bottom
4    4  3  7  8    # top
4    1  4  8  5    # left
4    2  3  7  6    # right

# Holes section
#  There is no hole in cube
0

# Region section
#  No region be defined
0
```

.node and .poly Another simple example shows the usage of .node and .poly files together to describe the input geometry (PLC). The geometry is the same as the previous one, but a rectanglar bar is removed. The modified example is shown in Figure 7. This geometry can't be described in a .smesh file, because the facets "front" and "back" contain holes.

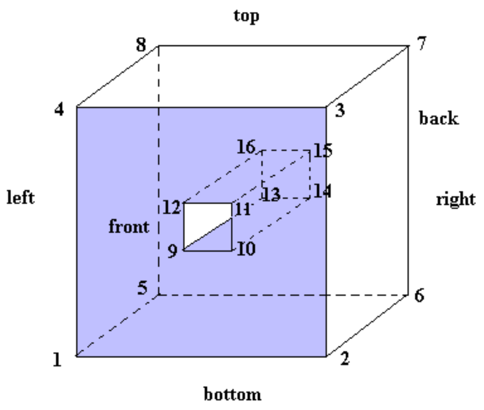

Figure 7: The modified 'cube' example.

File "cubebar.node" is used to list all points of the cube.

```
# cubebar.node file

#  16 points in 3D, no attributes, no boundary markers
16  3  0  0
1     0   0  10
2    10   0  10
3    10  10  10
4     0  10  10
5     0   0   0
6    10   0   0
7    10  10   0
8     0  10   0
9     4   4  10
10     6   4  10
11     6   6  10
12     4   6  10
13     4   4   0
14     6   4   0
15     6   6   0
16     4   6   0
```

File "cubebar.poly" is used to list all facets of the cube.

```
# cubebar.poly
```

```
# Node section
#  Set node number be zero to indicate that the
#  points are listed in ‘‘cubebar .node’’ file.
0

# Facet section
#  10 facets, no boundary markers
10  0

# facet definition starts here
#  The front and back facets have 2 polygons, 1 hole,
#  no boundary marker
2  1  0                # front
4    1  2  3  4
4    9  10  11  12
1    5.0  5.0  10.0    # hole point in front facet

2  1  0                # back
4    5  6  7  8
4    13  14  15  16
1    5.0  5.0  0        # hole point in back facet

# Other facets of cube: bottom, top, left, right
1                      # bottom
4    1  2  6  5
1                      # top
4    4  3  7  8
1                      # left
4    1  4  8  5
1                      # right
4    2  3  7  6

# Other facets of the bar: bottom, top, left, right
1                      # bottom
4    9  10  14  13
1                      # top
4    12  11  15  16
1                      # left
4    9  13  16  12
1                      # right
4    10  14  15  11

# Hole section
0

# Region section
0
```

A Algorithms Used in TetGen

This section gives a short overview of the algorithms used in TetGen by pointing to the literature.

For constructing Delaunay tetrahedrizations and convex hulls for 3D point sets, the randomized incremental flip algorithm [2] was used. This algorithm is incremental and adds a point in a sequence of flips. The 3D flip operations were first discussed in [3], and the proof that flipping succeeds if the points are added incremently can be found in [4].

TetGen applies the circumcenter insertion Delaunay refinement algorithm [9] to compute quality (conforming) Delaunay meshes from the boundary constrained Delaunay tetrahedrizations. This algorithms uses a quality measure called radius-edge ratio [5] to detect and eliminate all the badly shaped tetrahedra except slivers. The insertion of points into the circumcenter of the low quality elements transforms them by flipping into well shaped ones. This results in improved quality meshes and a grading effect with respect to different prescribed resolution in specified parts.

A heuristic boundary recovery algorithm is used to construct boundary constrained Delaunay tetrahedrization for piecewise linear complexes (PLC). This algorithm tries to recover the missing boundary edges of the PLC from its Delaunay tetrahedrization by inserting points.

Other algorithms used in TetGen include "Jump-and-Walk" point location [7] algorithm and the "Gift-wrapping" algorithm for constructing a local tetrahedrization. The mesh data structure and primitives manipulating meshes developed within the implementation of TetGen aim on memory efficiency, robustness, and speed.

B Explanations of Geometric Terms

This section gives simplified explanations of some frequently used geometric terms in the literature of 3D Delaunay mesh generation. This should help to understand this manual and to use TetGen properly. The definitions of these geometric terms can be found for instance in the book of Edelsbrunner [1].

Mesh

A **mesh** is a division of a polygonal domain Ω into simple elements. Elements can come in various dimensions and shapes. Most popular elements are triangles and quadrangles in 2D and tetrahedra and hexahedra in 3D. Let $T = \{t_i\}$ be a set of those elements, then they should cover the entire domain $\bigcup_{t_i \in T} = \Omega$. The mesh is conforming if the intersection $t_i \cap t_j$ for $i \neq j$ is either empty or a lower dimensional entity that is part of both elements, e.g. an edge that is shared by both elements.

TetGen generates Delaunay tetrahedral meshes.

Voronoi Diagram

Given a finite set of points $S \subset R^d$. For each point $P \in S$ the **Voronoi Cell** is defined as:

$$V(P) = \{x \in R^d \mid \forall Q \in S \ \|x - P\| \leq \|x - Q\|\}.$$

The **Voronoi Cell** for each point $P \in S$ is a convex polyhedron, which may extend (especially in the boundary) to infinity.

The **Voronoi Diagram** $V(S)$ is the collection of all the **Voronoi Cells** of all points in S.

Delaunay Triangulation and Delaunay Mesh

The expression `triangulation` is used in 2D and in 3D, it is also called `tetrahedrization` in 3D.

The **Delaunay triangulation** of a point set S in 3D consists of tetrahedra formed by points in S whose circumscribing spheres enclose no other points in S. This set of tetrahedra covers the convex hull of S.

One can use the Delaunay triangulation to obtain it's geometric dual, the **Voronoi diagram**. Figure 8 shows the relation of a Delaunay tetrahedron and the faces of

Voronoi cells.

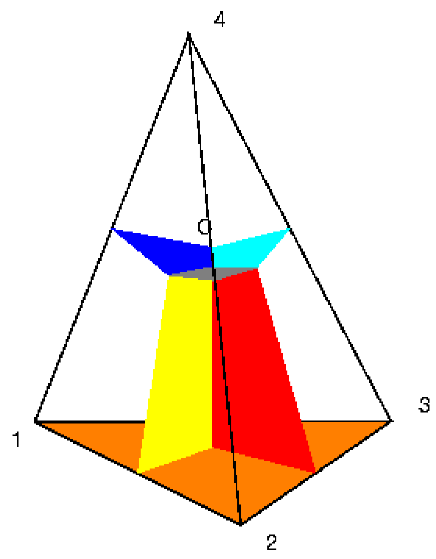

Figure 8: Delaunay tetrahedron and the parts of Voronoi polyhedra.

The **Delaunay mesh** of a 3D polygonal domain is a subdivision of this domain into tetrahedral elements. It's a subset of Delaunay triangulation of the point set of this domain.

Piecewise Linear Complex

The definition of **Piecewise Linear Complex (PLC)** is an extension of **Planar Straight Line Graph (PSLG)** in 2D [8], it is used to describ the 3D mesh domain (see [6]).

A **Piecewise Linear Complex (PLC)** is a collection of **vertices**, **segments** and **facets**. The meaning of **vertices** and **segments** are invariant in any dimension. However, **facets** can be quite complicated in shape. A facet has a planar boundary, it may have any number of boundary edges, may be non-convex, and may include holes, edges or vertices in it. Figure 5 gives two examples of facets. All points of a facet must be coplanar.

Constrained Delaunay Tetrahedrization

The definition of **Constrained Delaunay Tetrahedrization (CDT)** has a close relation to **Delaunay Tetrahedrization (DT)** and **Piecewise Linear Complex**

(PLC).

A **Constrained Tetrahedrization** of a PLC G is a tetrahedrization T of the vertices of G that includes the segments of G as a part of T, in addition, the facets of G are represented by a set of triangluar faces which are faces of tetrahedra of T. A **Constrained Delaunay Tetrahedrization (CDT)** of G is a constrained tetrahedrization of G that also has the property that it is as close to a **Delaunay Tetrahedrization** as possible.

A CDT of PLC is not a true Delaunay tetrahedrization. Some tetrahedra may be not Delaunay – the circumsphere of these elements may include other vertices of the PLC.

Conforming Delaunay Tetrahedrization

A **Conforming Delaunay Tetrahedrization** of a PLC G is a **Delaunay Tetrahedrization** of the vertices of another PLC $G1$, where $G1$ has the same geometry as G, but the vertices of $G1$ are augmented by additional vertices (called `Steiner points`) of G. These additional vertices have to be carefully chosen: the boundary (segments and facets) of G have to be reserved. In other words, the segments of G are represented by a set of contiguous edges in $G1$, and the facets of G are represented by a union of triangular faces in $G1$.

References

[1] H. Edelsbrunner, *Geometry and Topology for Mesh Generation* . Cambridge University Press, 2001, ISBN 0-521-79309-2.

[2] H. Edelsbrunner, N. Shah, *Incremental Topological Flipping Works for Regular Triangulations.* Algorithmica, 15:223-241, 1996.

[3] B. Joe, *Three-Dimensional Triangulations from Local Transformations.* SIAM J. Sci. Statist. Comput. 10:718-741, 1989

[4] B. Joe, *Construction of Three-Dimensional Triangulations Using Local Transformations.* Computer Aided Geometric Design 8:123-142, 1991

[5] G. L. Miller, D. Talmor, S. H. Teng, N. J. Walkington, *A Delaunay Based Numerical Method for Three Dimensions: Generations, Formulation, and Partition.* In "Proc. 27th Ann. ACM Sympos. Theory Comput.," 683-692, 1995

[6] G. L. Miller, D. Talmor, S. H. Teng, N. J. Walkington, *Control Volume Meshes using Sphere Packing: Generation, Refinement and Coarsening*, 5th International Meshing Roundtable (Pittsburgh, Pennsylvania), Pages 47-61, October, 1996.

[7] E. P. Mücke, I. Saias, B. H. Zhu, *Fast Randomized Point Location without Preprocessing in Two- and Three-Dimensional Delaunay Triangulations.* Proceedings of the Twelfth Annual Symposium on Computational Geometry, ACM, May 1996

[8] J. Ruppert, *A Delaunay Refinement Algorithm for Quality Two-Dimensional Mesh Generation.* Journal of Algorithms 18(3):548-585, May 1995

[9] J. R. Shewchuk, *Delaunay Refinement Mesh Generation*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, 1997. Available as Technical Report CMU-CS-97-137