6 Calling TetGen from Another Program

One can use TetGen as a library so that it can be called directly from another program. This section gives the necessary instructions for using the TetGen library. Users are supposed to be able to use TetGen, i.e., know its command line switches and the input and output file formats. We refer to Section 3 for the instructions of how to compile TetGen into a library.

6.1 The Header File

Programs calling TetGen must include the header file `tetgen.h`.

```
#include "tetgen.h"
```

It includes all data types and function declarations of the TetGen library. It defines the function `tetrahedralize()` and the data type `tetgenio`, which are provided for users to call TetGen with all its functionality. They are described in Section 6.2 and Section 6.3, respectively.

6.2 The Calling Convention

The function `tetrahedralize()` is declared as follows:

```
void tetrahedralize(char *switches, tetgenio *in, tetgenio *out,
                    tetgenio *addin = NULL, tetgenio *bgmin = NULL);
```

The parameter `switches` is a string containing the command line switches for this call. In this string, no initial dash '-' is required. The `Q` (quiet) switch is recommended in the final code. Some file output switches, like `I` and `g` are ignored.

The parameters `in` and `out`, which are two pointers pointing to objects of `tetgenio`, describing the input and the output. `in` and `out` must not be NULL.

Two additional parameters `addin` and `bgmin` may be supplied. When the switch `-i` is used, `addin` contains a list of additional vertices to be inserted. When the switch `-m` is used, `bgmin` contains a background mesh which is used to provide a mesh sizing function.

6.3 The `tetgenio` Data Type

The `tetgenio` structure is used to pass data into and out of the tetrahedralize() procedure. It replaces the input and output files of TetGen by a collection of arrays, which are used to store points, tetrahedra, boundary markers, and so forth. It is a `C++` class including data fields and functions. The data fields of `tetgenio`:

```
int firstnumber; // 0 or 1, default 0.
int mesh_dim;    // must be 3.

REAL *pointlist;
REAL *pointattributelist;
REAL *pointmtrlist;
int *pointmarkerlist;
int numberofpoints;
int numberofpointattributes;
int numberofpointmtrs;

int *tetrahedronlist;
REAL *tetrahedronattributelist;
REAL *tetrahedronvolumelist;
int *neighborlist;
int numberoftetrahedra;
```

```
    int numberofcorners;
    int numberoftetrahedronattributes;

    facet *facetlist;
    int *facetmarkerlist;
    int numberoffacets;

    REAL *holelist;
    int numberofholes;

    REAL *regionlist;
    int numberofregions;

    REAL *facetconstraintlist;
    int numberoffacetconstraints;

    REAL *segmentconstraintlist;
    int numberofsegmentconstraints;

    int *trifacelist;
    int *trifacemarkerlist;
    int numberoftrifaces;

    int *edgelist;
    int *edgemarkerlist;
    int numberofedges;
```

6.4 Description of Arrays

In all cases, the first item in any array is stored starting at index [0]. However, that item is item number `firstnumber` (0 or 1) unless the `z` switch is used, in which case it is item number '0'. Now the description of arrays follows.

pointlist

An array of point coordinates. The first point's x coordinate is at index [0], its y coordinate at index [1], and its z coordinate at index [2], followed by the coordinates of the remaining points. Each point occupies three REALs.

pointattributelist

An array of point attributes. Each point's attributes occupy `numberofpointattributes` REALs.

pointmarkerlist

An array of point markers; one int per point.

pointmtrlist

An array of metric tensors at points. Each point's tensor occupies `numberofpointmtrs` REALs.

tetrahedronlist

An array of tetrahedron corners. The first tetrahedron's first corner is at index [0], followed by its other three corners, followed by any other nodes if the '-o2' switch is used. Each tetrahedron occupies `numberofcorners` (4 or 10) ints.

tetrahedronattributelist

An array of tetrahedron attributes. Each tetrahedron's attributes occupy `numberoftetrahedronattributes` REALs.

tetrahedronvolumelist

An array of tetrahedron volume constraints; one REAL per tetrahedron. Input only.

neighborlist

An array of tetrahedron neighbors; four ints per tetrahedron. Output only.

facetlist

An array of PLC facets. Each facet is an object of type `facet` (see Section 6.4.2).

facetmarkerlist

An array of facet markers; one int per facet.

holelist

An array of holes. The first hole's x, y and z coordinates are at indices [0], [1] and [2], followed by the remaining holes. Three REALs per hole.

regionlist

An array of regional attributes and volume constraints. The first constraints' x, y and z coordinates are at indices [0], [1] and [2], followed by the regional attribute at index [3], followed by the maximum volume at index [4], followed by the remaining volume constraints. Five REALs per volume constraint. Each regional attribute is used only if the `A` switch is used, and each volume constraint is used only if the `a` switch (with no number following) is used, but omitting one of these switches does not change the memory layout.

facetconstraintlist

An array of facet maximum area constraints. Two REALs per constraint. The first one is the facet marker (cast the type to integer), the second is its maximum area bound. Note the 'facetconstraintlist' is used only for the 'q' switch.

segmentconstraintlist

An array of segment length constraints. Two REALs per constraint. The first one is the index (pointing into `pointlist`) of the node, the second is its maximum length bound. Note the 'segmentconstraintlist' is used only for the 'q' switch.

trifacelist

An array of triangular faces. The first face's corners are at indices [0], [1] and [2], followed by the remaining faces. Three ints per face.

trifacemarkerlist

An array of face markers; one int per face.

edgelist

An array of segment endpoints. The first segment's endpoints are at indices [0] and [1], followed by the remaining segments. Two ints per segment.

edgemarkerlist

An array of segment markers; one int per segment.

6.4.1 Memory Management

Two routines defined in tetgenio are used for memory initialization and cleaning. They are:

```
void initialize();
void deinitialize();
```

`initialize()` initializes all fields, that is, all pointers to arrays are initialized to NULL, and other variables are initialized to zero except the variable 'numberofcorners', which is 4 (a tetrahedron has 4 nodes). Initialization is implicitly called by the constructor of tetgenio. For an example, the following line creates an object of tetgenio named `io`, all fields of `io` are initialized:

```
tetgenio io;
```

The next step is to allocate memory for each array which will be used. In C++ the memory allocation and deletion can be done by the `new` and `delete` operators. Another pair of functions (preferred by C programmers) are `malloc()` and `free()`. Whatever you use, you must stick with one of these two pairs, e.g., 'new'/'delete' and 'malloc'/'free' cannot be mixed. For example, the following line allocates memory for `io.pointlist`:

```
io.pointlist = new REAL[io.numberofpoints * 3];
```

`deinitialize()` frees the memory allocated in objects of tetgenio by using 'delete'. It is automatically called on deletion of the tetgenio objects. If the memory was allocated by using the function `malloc()`, the user is responsible to free it. After having freed all memory, one call of `initialize()` disables the automatic memory deletion.

To reuse an object is possible: first call `deinitialize()`, then call `initialize()` before the next use.

6.4.2 The `facet` Data Structure

The `facet` data structure defined in tetgenio can be used to represent any facet of a PLC. The structure of facet shown below consists of a list of polygons and a list of hole points.

```
typedef struct {
  polygon *polygonlist;
  int numberofpolygons;
  REAL *holelist;
  int numberofholes;
} facet;
```

A polygon is again an object of type `polygon`. It consists of a list of corner points (`vertexlist`). The structure is shown below.

```
typedef struct {
  int *vertexlist;
  int numberofvertices;
} polygon;
```

The structure of a `facet` corresponds to the facet description in a `.poly` file format, described in Section 5.2.2. The front facet of Figure 23 serves an example for setting a PLC facet into an object of facet. It has two polygons, one has six vertices, and the other is a segment, no holes, the ASCII data is:

```
2
6  4 12 8 5 9 1 # front side
2  12 9
```

The following C++ code does the translation. Assume the object of tetgenio is `io` and has already be created.

```
tetgenio::facet *f;   // Define a pointer of facet.
tetgenio::polygon *p; // Define a pointer of polygon.

// All indices start from 1.
io.firstnumber = 1;

...

// Use 'f' to point to a facet of 'facetlist'.
f = &io.facetlist[i];
// Initialize the fields of this facet.
//   There are two polygons, no holes.
f->numberofpolygons = 2;
// Allocate memory for polygons.
f->polygonlist = new tetgenio::polygon[2];
f->numberofholes = 0;
f->holelist = NULL;

// Set the data of the first polygon into facet.
p = &f->polygonlist[0];
p->numberofvertices = 6;
// Allocate memory for vertices.
p->vertexlist = new int[6];
p->vertexlist[0] = 4;
p->vertexlist[1] = 12;
p->vertexlist[2] = 8;
p->vertexlist[3] = 5;
p->vertexlist[4] = 9;
p->vertexlist[5] = 1;

// Set the data of the second polygon into facet.
p = &f->polygonlist[1];
p->numberofvertices = 2;
p->vertexlist = new int[2]; // Alloc. memory for vertices.
p->vertexlist[0] = 12;
p->vertexlist[1] = 9;
```

6.5 A Complete Example

This section gives an example of how to call TetGen from another program by using the `tetgenio` data structure and the function `tetrahedralize()`. The input PLC in Section 5.4.1 (Figure 22) is used again.

The complete `C++` source code is given below. It is also available on TetGen's website:`http://www.tetgen.org/files/tetcall.cxx`. The code illustrates the following basic steps:

- at first it creates an input object `in` of tetgenio containing the data of the bar;
- then it calls function `tetrahedralize()` to create a quality mesh of the bar with output in `out`.

In addition, it outputs the PLC in the object `in` into two files (`barin.node` and `barin.poly`), and outputs the mesh in the object `out` into three files (`barout.node`, `barout.ele`, and `barout.face`).

This example can be compiled into an executable program.

- Compile TetGen into a library named `libtet.a` (see Section 3.1 for compiling);
- Save the file `tetcall.cxx` into the same directory in which you have the files `tetgen.h` and `libtet.a`;
- Compile it using the following command:

```
g++ -o test tetcall.cxx -L./ -ltet
```

which will result an executable file `test`.

The complete source codes are given below:

```
#include "tetgen.h" // Defined tetgenio, tetrahedralize().

int main(int argc, char *argv[])
{
  tetgenio in, out;
  tetgenio::facet *f;
  tetgenio::polygon *p;
  int i;

  // All indices start from 1.
  in.firstnumber = 1;

  in.numberofpoints = 8;
  in.pointlist = new REAL[in.numberofpoints * 3];
  in.pointlist[0]  = 0;  // node 1.
  in.pointlist[1]  = 0;
  in.pointlist[2]  = 0;
  in.pointlist[3]  = 2;  // node 2.
  in.pointlist[4]  = 0;
  in.pointlist[5]  = 0;
  in.pointlist[6]  = 2;  // node 3.
  in.pointlist[7]  = 2;
  in.pointlist[8]  = 0;
  in.pointlist[9]  = 0;  // node 4.
  in.pointlist[10] = 2;
  in.pointlist[11] = 0;
  // Set node 5, 6, 7, 8.
  for (i = 4; i < 8; i++) {
    in.pointlist[i * 3]     = in.pointlist[(i - 4) * 3];
    in.pointlist[i * 3 + 1] = in.pointlist[(i - 4) * 3 + 1];
    in.pointlist[i * 3 + 2] = 12;
  }

  in.numberoffacets = 6;
  in.facetlist = new tetgenio::facet[in.numberoffacets];
  in.facetmarkerlist = new int[in.numberoffacets];

  // Facet 1. The leftmost facet.
  f = &in.facetlist[0];
  f->numberofpolygons = 1;
  f->polygonlist = new tetgenio::polygon[f->numberofpolygons];
  f->numberofholes = 0;
  f->holelist = NULL;
  p = &f->polygonlist[0];
  p->numberofvertices = 4;
  p->vertexlist = new int[p->numberofvertices];
  p->vertexlist[0] = 1;
  p->vertexlist[1] = 2;
  p->vertexlist[2] = 3;
  p->vertexlist[3] = 4;

  // Facet 2. The rightmost facet.
  f = &in.facetlist[1];
  f->numberofpolygons = 1;
  f->polygonlist = new tetgenio::polygon[f->numberofpolygons];
  f->numberofholes = 0;
  f->holelist = NULL;
  p = &f->polygonlist[0];
  p->numberofvertices = 4;
  p->vertexlist = new int[p->numberofvertices];
  p->vertexlist[0] = 5;
  p->vertexlist[1] = 6;
  p->vertexlist[2] = 7;
  p->vertexlist[3] = 8;
```

```
    // Facet 3. The bottom facet.
    f = &in.facetlist[2];
    f->numberofpolygons = 1;
    f->polygonlist = new tetgenio::polygon[f->numberofpolygons];
    f->numberofholes = 0;
    f->holelist = NULL;
    p = &f->polygonlist[0];
    p->numberofvertices = 4;
    p->vertexlist = new int[p->numberofvertices];
    p->vertexlist[0] = 1;
    p->vertexlist[1] = 5;
    p->vertexlist[2] = 6;
    p->vertexlist[3] = 2;

    // Facet 4. The back facet.
    f = &in.facetlist[3];
    f->numberofpolygons = 1;
    f->polygonlist = new tetgenio::polygon[f->numberofpolygons];
    f->numberofholes = 0;
    f->holelist = NULL;
    p = &f->polygonlist[0];
    p->numberofvertices = 4;
    p->vertexlist = new int[p->numberofvertices];
    p->vertexlist[0] = 2;
    p->vertexlist[1] = 6;
    p->vertexlist[2] = 7;
    p->vertexlist[3] = 3;

    // Facet 5. The top facet.
    f = &in.facetlist[4];
    f->numberofpolygons = 1;
    f->polygonlist = new tetgenio::polygon[f->numberofpolygons];
    f->numberofholes = 0;
    f->holelist = NULL;
    p = &f->polygonlist[0];
    p->numberofvertices = 4;
    p->vertexlist = new int[p->numberofvertices];
    p->vertexlist[0] = 3;
    p->vertexlist[1] = 7;
    p->vertexlist[2] = 8;
    p->vertexlist[3] = 4;

    // Facet 6. The front facet.
    f = &in.facetlist[5];
    f->numberofpolygons = 1;
    f->polygonlist = new tetgenio::polygon[f->numberofpolygons];
    f->numberofholes = 0;
    f->holelist = NULL;
    p = &f->polygonlist[0];
    p->numberofvertices = 4;
    p->vertexlist = new int[p->numberofvertices];
    p->vertexlist[0] = 4;
    p->vertexlist[1] = 8;
    p->vertexlist[2] = 5;
    p->vertexlist[3] = 1;

    // Set 'in.facetmarkerlist'

    in.facetmarkerlist[0] = -1;
    in.facetmarkerlist[1] = -2;
    in.facetmarkerlist[2] = 0;
    in.facetmarkerlist[3] = 0;
    in.facetmarkerlist[4] = 0;
    in.facetmarkerlist[5] = 0;

    // Output the PLC to files 'barin.node' and 'barin.poly'.
    in.save_nodes("barin");
    in.save_poly("barin");
```

```
  // Tetrahedralize the PLC. Switches are chosen to read a PLC (p),
  //   do quality mesh generation (q) with a specified quality bound
  //   (1.414), and apply a maximum volume constraint (a0.1).

  tetrahedralize("pq1.414a0.1", &in, &out);

  // Output mesh to files 'barout.node', 'barout.ele' and 'barout.face'.
  out.save_nodes("barout");
  out.save_elements("barout");
  out.save_faces("barout");

  return 0;
}
```