COMPUTING
OBJECTS

CM2 TetraMesh® Iso/Aniso

Version 5.6

tutorials

Revision February 2025.

https://www.computing-objects.com

Forewords

This manual is a tutorial for the solid tetrahedron mesh generators of the **CM2 MeshTools®** SDK:

- The isotropic mesher **CM2 TetraMesh® Iso,**
- The anisotropic mesher **CM2 TetraMesh® Aniso**.

These two tools are *constrained* unstructured meshers: the boundary mesh (i.e. the triangle surface mesh), and the internal hard faces, hard edges and hard points if any, are kept (if possible) unmodified in the final mesh.

Based on a fast and robust hybrid "advancing-front" and Delaunay algorithm, they generate high quality elements with smooth grading size according to the length of the boundary elements, to the user-specified sizes or to the metric map supported by a background mesh.

They can generate very big meshes – several millions of elements – with a limited amount of memory. The speed is near independent of the number of the elements to be generated.

Option switches can be used to adapt the meshers to the various needs of the user concerning mesh refinement and optimization. In this regard, the meshers can also be used as mere optimizers of some already existing tetrahedral meshes.

Many data concerning the mesh are available upon exit: histograms of the shape qualities and the size qualities, matrix of the neighbors, number of sub-domains, meshed volume...

Like many other meshers of the library, **CM2 TetraMesh Iso/Aniso** are multi-threaded (you can select in the settings the maximum number of threads the generator can use).
The generated meshes are reproducible (same mesh with same input data and same mesh with any number of threads).

Data are exchanged with the CM2 mesh generators through vector and matrix objects (no file). Beginners should start by reading the manual **CM2 Math1 - overview** to get first views on these mathematical containers.

For a complete description of the data and settings structures used with these meshers please refer to the **CM2 TetraMesh Iso/Aniso - reference manual**.

Table of contents

Forewords 2

1. Getting started 5

Some declarations 6

Authorization of the library 6

Boundary mesh 7

Tetrahedral mesh 7

2. Simple cube 10

Some declarations 11

Boundary mesh 11

Tetrahedral mesh 13

3. Cube with an internal hard line 15

4. Cube with an internal hard node 18

5. Internal cavity 20

6. Quadratic elements & high-order nodes 23

7. Multiple meshes 25

8. Repairing the boundary mesh 26

9. Background mesh 28

10. Anisotropic meshes (CM2 TetraMesh Aniso) 31

Mesh gallery 39

This manual shows examples of 3-D tetrahedral meshings illustrating along the way some of the major options of the meshers **CM2 TetraMesh® Iso/Aniso**.

Each example starts with including the file `stdafx.h` (can be a pre-compiled header) giving access to the classes and the functions of the library (API).

The general namespace `cm2` has nested namespaces such as `cm2::vecscal`, `cm2::vecvec`, `cm2::meshtools` or `cm2::triamesh_iso`. The user can add a `using namespace cm2` directive in this `stdafx.h` file. Keeping namespaces in the user's source code can however be useful to improve the legibility and to avoid name conflicts. In the rest of this document we assume such a `using namespace cm2` directive.

File `stdafx.h`:

```
// CM2 MESHTOOLS
#include "meshtools.h"        // General purpose mesh routines.
#include "meshtools2d.h"      // To generate 1D meshes.
#include "meshtools2d.h"      // To generate 2D meshes.
#include "tetramesh_iso.h"    // CM2 TetraMesh Iso.
#include "tetramesh_aniso.h"  // CM2 TetraMesh Aniso (Section 10 only).

using namespace cm2;          // Main cm2 namespace can now be omitted.
```

Required libraries[1]:

- `cm2math1`
- `cm2misc`
- `cm2meshtools`
- `cm2meshtools2d`
- `cm2meshtools2d`
- `cm2meshtools3d`
- `cm2tetramesh_iso`
- `cm2tetramesh_aniso` *(Section 10 only)*

1 The lib names end with `_($platform)_($ver)`. For instance `cm2tetramesh_iso_x64_56.dll`.
On Windows, file extensions for the libraries are `.lib` and `.dll`. On Linux/Unix/macOS platforms, file extensions are usually .a (static archive), .so or .dylib (dynamic lib).

1. Getting started

The simple way to see **CM2 TetraMesh Iso** in action is to use a pre-existing boundary mesh (triangle mesh). We assume here we have such a boundary mesh of ready on the ASCII file `part.dat`. The expected format here is quite simple: first the coordinates of the nodes as a 3x*NP* matrix, then the connectivity of the triangle mesh as a 3x*NT* matrix[2]. For each matrix the format is:

$$\begin{array}{lllll}
n \text{ X } m \; [& & & & \\
d_{0,0} & d_{0,1} & d_{0,2} & \dots & d_{0,m-1} \\
d_{1,0} & d_{1,1} & d_{1,2} & \dots & d_{1,m-1} \\
\dots & & & & \\
d_{n-1,0} & d_{n-1,1} & d_{n-1,2} & \dots & d_{n-1,m-1} \;]
\end{array}$$

For instance, a mesh with 4 nodes and 2 triangles could write[3]:

```
3 X 4 [
0.  1.  1.  0.
0.  0.  1.  1.
0.  0.  0.5 0. ]

3 X 2 [
0  1
1  2
3  3 ]
```

The source code to generate the tetrahedral mesh writes:

```
#include "stdafx.h"
#include <iostream>
#include <fstream>

// Simple optional display handler.
static void display_hdl (void*, unsigned, const char* msg) { std::cout << msg; }

int main()
{
   std::ifstream                          istrm("part.dat");
   cm2::tetramesh_iso::mesher             the_mesher;
   cm2::tetramesh_iso::mesher::data_type  data;

   // UNLOCK THE DLL.
   cm2::tetramesh_iso::registration("Licensed to SMART Inc.", "F53EA108BCWX");

   // READ THE BOUNDARY TRIANGLE MESH.
   cm2::matio::read(istrm, data.pos);
   cm2::matio::read(istrm, data.connectB);

   // GENERATE THE 3D MESH.
   the_mesher.run(data);

   // SOME OUTPUT INFO (OPTIONAL).
   data.print_info(&display_hdl);

   // VISUALISATION (OPTIONAL).
   cm2::meshtools::medit_output("TH.mesh", data.pos, data.connectM, CM2_TETRA4);

   return 0;
}   // main
```

[2] Other supported input formats are: STL (ASCII and binary), FEMAP neutral, Nastran and Alias' Wavefront OBJ.

[3] The format for each value is free.

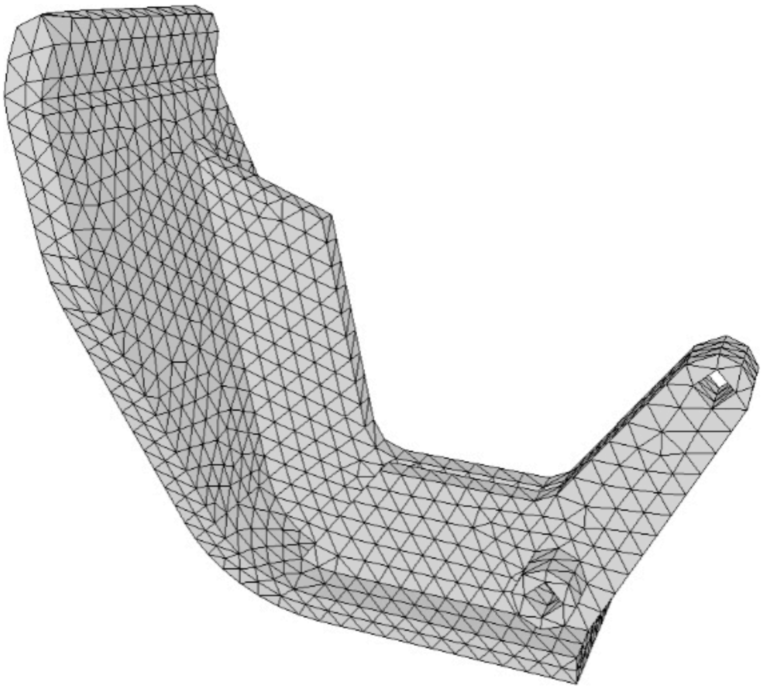


Figure 1 – A simple mechanical part.

Let us explain this program line by line.

Some declarations

A standard input stream `istrm` is built with the data file name `part.dat`. We create then an instance of the tet mesher (`the_mesher`) and a data structure (`data`) that will contain among others the coordinates of the nodes (in field `data.pos`) and the connectivity matrices (initial triangle boundary mesh in `data.connectB` and final tet mesh in `data.connectM`).

Matrix `data.pos` is a `DoubleMat` (variable-sized matrix of doubles)[4] and matrices `data.connectB` and `data.connectM` are `UIntMat`. For instance `connectB(i, j)` is the ith local node of the jth element. This integer refers to the column number in matrix `data.pos` where the coordinates of this node can be found[5].

Authorization of the library

The `tetramesh_iso` library is unlocked through a call to `cm2::tetramesh_iso::registration`. Two strings must be provided for each library: the name of your company or organization that has acquired the license and a secret code[6]. Note that both strings are case sensitive and the registration call must be made each time the library is loaded into memory and before any run of the mesher.

```
cm2::tetramesh_iso::registration("Licensed to SMART Inc.", "F53EA108BCWX");
```

[4] See manual **CM2 Math1 - overview**.

[5] Recall that array indices are zero based (from 0 to N-1).

[6] Contact `license@computing-objects.com` for any licensing inquiry.

Boundary mesh

The cm2::matio::read can load matrices following the above specified ASCII format from an input stream[7]. We use it to load the nodes' coordinates and the triangle connectivity from the same file.

```
cm2::matio::read(istrm, data.pos);
cm2::matio::read(istrm, data.connectB);
```

Tetrahedral mesh

```
the_mesher.run(data);
```

Upon exit, the matrix data.pos is bigger and contains all the new nodes generated inside the domain by the tetrahedral mesher. These new points are appended to the original matrix. The initial points of the boundary mesh are left untouched in the first columns.

The connectivity of the final mesh is stored in the matrix data.connectM, each column storing the indices of the nodes for an element. connectM(i, j) is the ith local node of the jth element.

Output information

Printed information about the generated mesh and a MEDIT[8] output file are obtained with:

```
data.print_info(&display_hdl);
cm2::meshtools::medit_output("TH.mesh", data.pos, data.connectM, CM2_TETRA4);
```

Figure 2 shows the output given by data.print_info(&display_hdl): the generated mesh is made of 2428 nodes and 8008 tetrahedrons for an initial boundary mesh of 4174 triangles and 2085 nodes (hard faces and hard nodes). The time spent in each of the three steps of the meshing process is given in seconds[9].

The first step is the generation of the front mesh (3D triangularization of the hard nodes and faces). In the second step, new nodes are generated inside the domain to get elements with good shape and size. Finally, the last step concerns geometrical and topological optimizations to improve the quality of the elements.

The formula used to compute the shape quality of a tetrahedron is given by:

$$Q_s = 6\sqrt{6}\,\frac{V}{L_{\max}\,S}$$

with:

V	Volume of the tetrahedron.
L_{max}	Length of the longest edge of the tetrahedron.
S	Total area of the four faces of the tetrahedron.

[7] This function is a template function and can be used with a wide range of streams and matrix classes. A similar cm2::matio::transpose_read function exists to read a matrix and transpose it on the fly.

[8] MEDIT is a free visualization program (link). Other output formats are: Ensight, FEMAP (neutral), Nastran, STL (ASCII or binary), VTK and Wavefront OBJ.

[9] All runs were done with x64 CM2 libs (VS 2017 MD build) on Windows® 8.1 x64 with Intel® Xeon® E3-1270 V2 3.5 GHz (4 cores with hyper-threading, turbo boost disabled). The typical speed with default settings on such a platform ranges from 8 000 tets / s. (**CM2 TetraMesh® Aniso** with background mesh) to more than 80 000 tets / s. (**CM2 TetraMesh Iso** without background mesh). Speed can be increased further by reducing the optimization level (see reference manual). Pure 3-D tetrahedrisation in CONVEX_HULL_MODE is even faster.

This quality measure ranges from zero (for a degenerated tet) to one (for an equilateral tet).

On the above example, the worst shape quality is 0.23 and the average is 0.68 (Figure 2).

Note that the boundary triangles being not equilateral, the tets built upon them cannot be perfect either. There is an upper bound for the quality of these boundary tets. Here the limit is 0.56. That means that the generator cannot produce a mesh with a minimum quality better than 0.56 in this case.

The *size quality* is also an important parameter to take into account. The size quality of an edge is a measure based upon its actual length and the target size values set on its vertices. A size quality of 1 indicates that the edge has the right length, i.e. optimal length. A too short edge has a size quality less than 1 (but always positive), and a too long edge has a size quality greater than 1. An edge with a quality of 2 is twice as long as it should be.

The formula used to compute the length quality of an edge AB writes:

$$Q_h^{AB} = L_{AB} \frac{\ln\left(\frac{h_A}{h_B}\right)}{h_A - h_B}$$

with:

L_{AB}	Actual length of edge AB.
h_A	Target size at node A (expected edge length at A).
h_B	Target size at node B (expected edge length at B).

Let's introduce also at this point the h-shock measure of an edge:

$$hs^{AB} = \min\left(\frac{h_A}{h_B}, \frac{h_B}{h_A}\right)^{\frac{1}{Q_h^{AB}}} - 1$$

These two measure are dimensionless and positive.

When $h_A = h_B$ the h-shock is null and the length writes $Q_h^{AB} = \frac{L_{AB}}{h_A}$.

When $Q_h^{AB} = 1$ edge *AB* is considered having optimal length with respect to its target mesh sizes h_A and h_B.

To optimize a mesh we need to improve simultaneously both the shape quality of the elements and the size quality of the edges. On top of these, the h-shock should be kept lower than a maximum threshold to ensure smooth gradations and all the prescribed entities (hard faces, edges and nodes) must be honored. All this makes the job of the optimizer difficult and heuristics must be used.

The mesher computes the histogram of the size qualities (normalized edge lengths) when the flag `my_mesher.settings.compute_Qh_flag`[10] is set to true[11].

On this example, the size qualities are well centered near the value 1 (mean value 1.01) with a small variance (Figure 3).

[10] See reference manual for details on the settings of the meshers.

[11] It can also be computed after the meshing with a call to `cm2::meshtools::edge_qualities`.

```
*******************************************************
*           CM2 TetraMesh(R) Iso (5.6.0.0)            *
*******************************************************
Hard nodes      : 2085/2085
Hard edges      : 6261/6261
Hard faces      : 4174/4174
Nodes           : 2428
Tets            : 8008
Missing faces   : 0
Subdomains      : 1
Volume          : 1.802054E+04
Qmin            : 1.120209E-01 (max-min: 5.561858E-01)
Steiner nodes   : 0
Front time      : 0.07 s.
Refine time     : 0.03 s.
Optim time      : 0.02 s.
Total time      : 0.12 s. (65105.74 th/s.)

************ HISTOGRAM QS *************
Total number of bins     :          11
Total number of counts   :        8008
Number of larger values  :           0
Number of smaller values :           0
V max                    : 9.981352E-01
V mean                   : 6.785422E-01
V min                    : 1.120209E-01

Bin number       -- Bin boundaries --          Hits

   10            0.90           1.00            342
    9            0.80           0.90            881
    8            0.70           0.80           1972
    7            0.60           0.70           2536
    6            0.50           0.60           1947
    5            0.40           0.50            289
    4            0.30           0.40             39
    3            0.20           0.30              1
    2            0.10           0.20              1
    1            0.01           0.10              0
    0            0.00           0.01              0

NODES      : 2428
NEFS       : 8008
TIME       : 0.12 s.
NEFS / s   : 65105.74
```

Figure 2 – Output info for the “mechanical part” example.

```
************ HISTOGRAM QH *************
Total number of bins     :          20
Total number of counts   :       12524
Number of larger values  :           0
Number of smaller values :           0
V max                    : 1.762665E+00
V mean                   : 1.008241E+00
V min                    : 2.886290E-01

Bin number       -- Bin boundaries --          Hits

   19           10.00           +INF              0
   18            5.00          10.00              0
   17            3.33           5.00              0
   16            2.50           3.33              0
   15            2.00           2.50              0
   14            1.67           2.00              5
   13            1.43           1.67            405
   12            1.25           1.43           1430
   11            1.11           1.25           1331
   10            1.00           1.11           2589
    9            0.90           1.00           3237
    8            0.80           0.90           1629
    7            0.70           0.80           1313
    6            0.60           0.70            453
    5            0.50           0.60             75
    4            0.40           0.50             44
    3            0.30           0.40             11
    2            0.20           0.30              2
    1            0.10           0.20              0
    0            0.00           0.10              0
```

Figure 3 – Histogram of the size-qualities of all the edges in the “mechanical part” example.

2. Simple cube

This second example illustrates some of the meshtools auxiliary functions to generate 1-D and 2-D meshes. This will also help introduce some options of **CM2 TetraMesh Iso** in following examples.

The following code meshes a cube from scratch (no boundary mesh read from file, cm2 namespace omitted):

```
#include "stdafx.h"
#include <iostream>

// Simple optional display handler.
static void display_hdl (void*, unsigned, const char* msg) { std::cout << msg; }

int main()
{
   const unsigned       N(6);     // The discretization along each edge.
   const double         L(4.);    // The sides length.
   DoubleMat            pos;
   UIntMat              connectE, connectB, connectB1, connectB2;
   UIntVec              indices;

   // UNLOCK THE DLL.
   tetramesh_iso::registration("Licensed to SMART Inc.", "F53EA108BCWX");

   // BOUNDARY TRIANGLE MESH.
   meshtools2d::extrude_translate(pos, DoubleVec3(0, 0, L), DoubleVec3(L, 0, 0), N, indices);
   meshtools2d::indices_to_connectE2(indices, connectE);
   meshtools2d::extrude_translate_T3(pos, connectE, DoubleVec3(0, L, 0), N, 2, connectB);

   meshtools::copy_mesh(pos, connectB1, connectB);
   meshtools::rotate(pos, DoubleVec3(L/2, L/2, L/2), DoubleVec3(0., M_PI, 0.), connectB1);
   connectB.push_back(connectB1);

   meshtools::copy_mesh(pos, connectB1, connectB);
   meshtools::rotate(pos, DoubleVec3(L/2, L/2, L/2), DoubleVec3(M_PI/2, 0., 0.), connectB1);
   connectB.push_back(connectB1);

   meshtools::copy_mesh(pos, connectB2, connectB1);
   meshtools::rotate(pos, DoubleVec3(L/2, L/2, L/2), DoubleVec3(0., 0., M_PI/2), connectB2);
   connectB.push_back(connectB2);

   meshtools::merge(pos, connectB, /*tol=>*/ 1E-6, /*merge_type=>*/ 0);

   // 3D MESH.
   tetramesh_iso::mesher              the_mesher;
   tetramesh_iso::mesher::data_type   data(pos, connectB);
   the_mesher.run(data);

   // VISUALISATION (OPTIONAL).
   data.print_info(&display_hdl);
   meshtools::medit_output("TH.mesh", data.pos, data.connectM, CM2_TETRA4);

   return 0;
}   // main
```

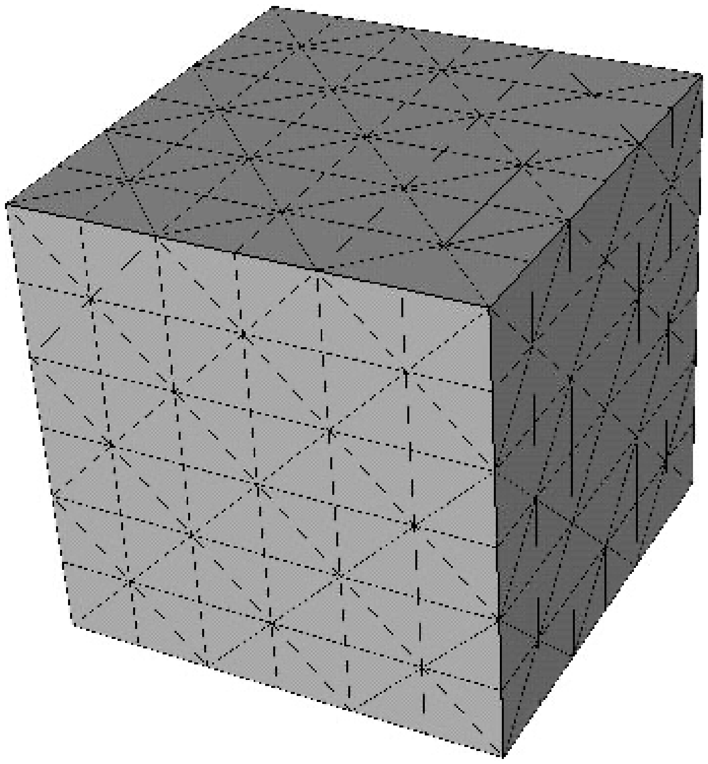


Figure 4 – Simple cube.

Let us again explain this program line by line.

Most of the lines are related to the generation of the boundary mesh. The tetrahedral mesh itself uses only three lines in the program.

Some declarations

Matrix `pos` stores all the points' coordinates: the nodes of the initial (boundary) mesh and the nodes of the final mesh (many of them are in both).

Matrix `connectB` stores the connectivity matrix of the boundary triangle mesh.

Matrices `connectB1`, `connectB2` and `connectE` are temporaries as well as vector `indices`.

Boundary mesh

In this example, we only use routines from the **CM2 MeshTools®** SDK but, as already seen, the user is free to generate this boundary mesh with other tools or even to read it from a file. In any case, the tetrahedral mesher needs this boundary mesh as a couple of matrices: the matrix `pos` of coordinates of the nodes and the matrix `connectB` of connectivity of the boundary triangles.

We choose to generate the mesh on one of the faces and then copy and rotate to get the other five faces of the cube.

The first face is done in three steps. First, N+1 points are generated along the line (0, 0, L) - (L, 0, L):

```
meshtools2d::extrude_translate(pos, DoubleVec3(0., 0., L), DoubleVec3(L, 0., 0.), N, indices);
```

This generates N+1 points equally spaced between (0, 0, L) and (L, 0, L) as new appended columns in the `pos` matrix. The index of each point (i.e. the column in matrix `pos`) is also appended to the vector `indices`. This vector contains upon exit of this function (N = 6):

```
[0 1 2 3 4 5 6]
```

And the matrix `pos` is now a 3x7 matrix. The first column equals to (0, 0, L) and the last one to (L, 0, L).

The vector of indices is used to create the connectivity matrix of the first edge mesh:

```
meshtools2d::indices_to_connectE2(indices, connectE);
```

`connectE` is now a 2x6 matrix:

```
2x6 [0 1 2 3 4 5
       1 2 3 4 5 6]
```

This line mesh is then extruded into a mesh of triangles along vector (0, L, 0):

```
meshtools2d::extrude_translate_T3(pos, connectE, DoubleVec3(0., L, 0.), N, 2, connectB);
```

The 5th parameter (here 2) controls the type of triangle mesh to be generated: 2 is for an alternate “Union Jack” pattern, 0 and 1 for a simply oriented mesh (left or right)[12]. The coordinates of the new points are appended to matrix `pos` which now becomes a 3 x 49 matrix[13] The 72 generated triangles are appended to matrix `connectB` as new columns of indices of nodes.

We now have to copy and rotate this mesh:

```
meshtools::copy_mesh(pos, connectB1, connectB);
meshtools::rotate(pos, DoubleVec3(L/2, L/2, L/2), DoubleVec3(0., M_PI, 0.), connectB1);
connectB.push_back(connectB1);
```

After the copy, the two meshes `connectB` and `connectB1` are coincident (geometrically equal) and matrix `pos` is twice as bigger as before. The new copy is then subjected to a rotation of π around axis {(L/2, L/2, L/2), Y}. It is then concatenated to the first mesh: the columns of `connectB1` are appended to `connectB`.

[12] Several variants of this function are available in the `meshtools2d` library and are described in the API.

[13] The memory is automatically reallocated as the matrix grows. If size is known early, reserving space can avoid memory reallocations and copies: `pos.reserve(3, 500);    // reserve space for 500 points but dimensions are unchanged.`

The two faces together are then copied and rotated twice in order to generate the other four missing faces of the cube:

```
meshtools::copy_mesh(pos, connectB1, connectB);
meshtools::rotate(pos, DoubleVec3(L/2, L/2, L/2), DoubleVec3(M_PI/2, 0., 0.), connectB1);
connectB.push_back(connectB1);

meshtools::copy_mesh(pos, connectB2, connectB1);
meshtools::rotate(pos, DoubleVec3(L/2, L/2, L/2), DoubleVec3(0., 0., M_PI/2), connectB2);
connectB.push_back(connectB2);
```

At this stage, `connectB` has 432 columns, i.e. 432 triangles and matrix `pos` has 294 columns, i.e. 294 points. However, some points are coincident and need to be merged together:

```
meshtools::merge(pos, connectB, /*tol=>*/ 1E-6, /*merge_type=>*/ 0);
```

This function has no effect on matrix `pos` only on `connectB`. When at least two nodes are coincident, the index of the first node encountered replaces all the others subsequent coincident nodes[14].

Tetrahedral mesh

Now we have generated the boundary mesh, we can call the 3-D mesher on it:

```
tetramesh_iso::mesher::data_type   data(pos, connectB);
the_mesher.run(data);
```

The difference with the previous example is that matrices `pos` and `connect` are built first and then passed to the `data` structure whereas in Section 1 we loaded directly `data.pos` and `data.connectB` from the stream. There is no significant difference between the two approaches. We can either modify directly `data.pos` and `data.connectB` or build some separated matrices `pos` and `connect` and then pass them to the data constructor. There is no hard copy done here, only *shallow* copies (data are shared). See manual **CM2 Math1 - overview**.

Upon exit, the matrix `data.pos` is bigger[15] and contains all the new nodes generated inside the cube by the tetrahedral mesher on top of the initial surface points.

[14] Note that this will leave some unused points in matrix `pos`. This space can be recovered using `cm2::meshtools::simplify`.

[15] But the `pos` matrix still has 294 columns.

```
*******************************************************
*           CM2 TetraMesh(R) Iso (5.6.0.0)            *
*******************************************************
Hard nodes      : 218/218
Hard edges      : 648/648
Hard faces      : 432/432
Nodes           : 277
Tets            : 900
Missing faces   : 0
Subdomains      : 1
Volume          : 6.400000E+01
Qmin            : 4.930722E-01 (max-min: 7.821716E-01)
Steiner nodes   : 0
Front time      : 0.01 s.
Refine time     : 0.00 s.
Optim time      : 0.00 s.
Total time      : 0.02 s. (59999.58 th/s.)

************ HISTOGRAM QS ************
Total number of bins     :          11
Total number of counts   :         900
Number of larger values  :           0
Number of smaller values :           0
V max                    : 1.000000E+00
V mean                   : 6.972722E-01
V min                    : 4.930722E-01

Bin number       -- Bin boundaries --          Hits

   10           0.90            1.00            11
    9           0.80            0.90            88
    8           0.70            0.80           373
    7           0.60            0.70           285
    6           0.50            0.60           142
    5           0.40            0.50             1
    4           0.30            0.40             0
    3           0.20            0.30             0
    2           0.10            0.20             0
    1           0.01            0.10             0
    0           0.00            0.01             0

NODES      : 277
NEFS       : 900
TIME       : 0.02 s.
NEFS / s   : 59999.58
```

Figure 5 – Output info for the cube example.

```
************ HISTOGRAM QH ************
Total number of bins     :          20
Total number of counts   :        1392
Number of larger values  :           0
Number of smaller values :           0
V max                    : 1.480346E+00
V mean                   : 9.585737E-01
V min                    : 6.612180E-01

Bin number       -- Bin boundaries --          Hits

   19          10.00            +INF             0
   18           5.00           10.00             0
   17           3.33            5.00             0
   16           2.50            3.33             0
   15           2.00            2.50             0
   14           1.67            2.00             0
   13           1.43            1.67             2
   12           1.25            1.43           126
   11           1.11            1.25           162
   10           1.00            1.11           469
    9           0.90            1.00           109
    8           0.80            0.90            52
    7           0.70            0.80           467
    6           0.60            0.70             5
    5           0.50            0.60             0
    4           0.40            0.50             0
    3           0.30            0.40             0
    2           0.20            0.30             0
    1           0.10            0.20             0
    0           0.00            0.10             0
```

Figure 6 – Histogram of the size-qualities of all the edges in the cube example.

3. Cube with an internal hard line

Starting from the previous example, we add a hard line inside the cube:

```
#include "stdafx.h"

int main()
{
   const unsigned       N(6);     // The discretization along each edge.
   const double         L(4.);    // The sides length.
   DoubleMat            pos;
   UIntMat              connectE, connectB, connectB1, connectB2, connectM2;
   UIntVec              indices;

   // UNLOCK THE DLL.
   tetramesh_iso::registration("Licensed to SMART Inc.", "F53EA108BCWX");

   // BOUNDARY TRIANGLE MESH.
   meshtools2d::extrude_translate(pos, DoubleVec3(0,0,L), DoubleVec3(L, 0., 0.), N, indices);
   meshtools2d::indices_to_connectE2(indices, connectE);
   meshtools2d::extrude_translate_T3(pos, connectE, DoubleVec3(0., L, 0.), N, 2, connectB);

   meshtools::copy_mesh(pos, connectB1, connectB);
   meshtools::rotate(pos, DoubleVec3(L/2, L/2, L/2), DoubleVec3(0., M_PI, 0.), connectB1);
   connectB.push_back(connectB1);

   meshtools::copy_mesh(pos, connectB1, connectB);
   meshtools::rotate(pos, DoubleVec3(L/2, L/2, L/2), DoubleVec3(M_PI/2, 0., 0.), connectB1);
   connectB.push_back(connectB1);

   meshtools::copy_mesh(pos, connectB2, connectB1);
   meshtools::rotate(pos, DoubleVec3(L/2, L/2, L/2), DoubleVec3(0., 0., M_PI/2), connectB2);
   connectB.push_back(connectB2);

   indices.clear(); connectE.clear();
   meshtools2d::extrude_translate(pos, DoubleVec3(0,L/2,L/2), DoubleVec3(L,0,0), 4*N, indices);
   meshtools2d::indices_to_connectE2(indices, connectE);
   meshtools::merge(pos, connectB, connectE, /*tol=>*/ 1E-6);

   // 3D MESH.
   tetramesh_iso::mesher              the_mesher;
   tetramesh_iso::mesher::data_type   data(pos, connectB);
   data.connectE = connectE;
   the_mesher.run(data);

   // MESH VISUALIZATION.
   meshtools::clip(data.pos, data.connectM,
                   DoubleVec3(0., 0., L/2 + 1E-6), DoubleVec3(0., 0., -1.), connectM2);
   meshtools::medit_output("TH.mesh", data.pos, connectM2, CM2_TETRA4);

   return 0;
}   // main
```

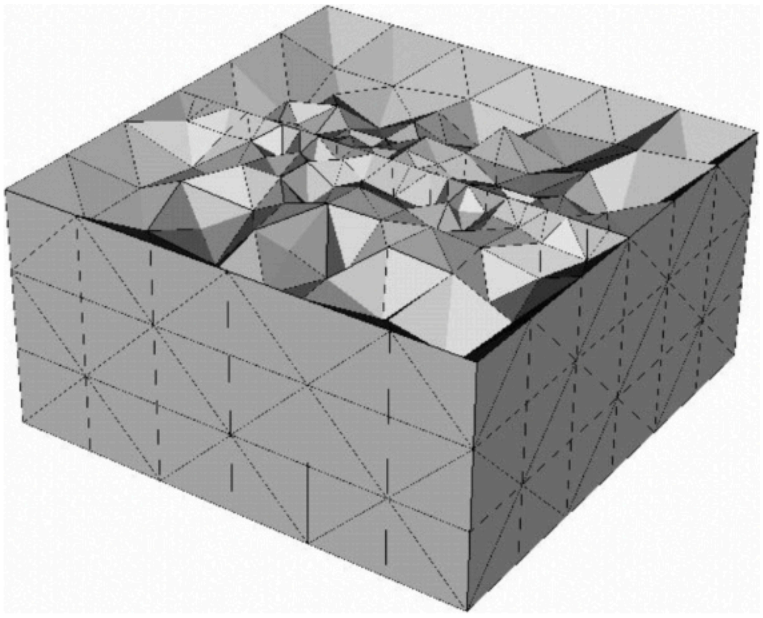


Figure 7 – Half cube with internal line (mesh cut).

The mesh of the internal line is generated like the mesh of the first edge of the cube:

```
meshtools2d::extrude_translate(pos, DoubleVec3(0., L/2, L/2), DoubleVec3(L,0,0), 4*N, indices);
meshtools2d::indices_to_connectE2(indices, connectE);
```

This mesh line starts at the center of one of the faces of the cube and extends to the opposite face. The mesh is four times finer along this line than on the cube edges, in order to make the line more visible and also to get a graded mesh size.

To avoid coincident nodes, a merging is needed between the two nodes at extremities of the line and the two matching nodes of the boundary mesh. This is possible because there are actually two nodes of the boundary mesh with coordinates (0, L/2, L/2) and (L, L/2, L/2) due to the regular structured meshing of the faces. This is done concurrently with the merging of the nodes of the edges of the cube already discussed in the previous section:

```
meshtools::merge(pos, connectB, connectE, /*tol=>*/ 1E-6);
```

The new internal prescribed edges are given to the mesher with:

```
data.connectE = connectE;
```

To visualize the internal line, we ask for a partial copy of the mesh in matrix `connectM2` by removing all tets with at least one node with Z coordinate greater than L/2. This has no effect on the coordinate matrix `data.pos`, or on the connectivity matrix of the full mesh `data.connectM`:

```
meshtools::clip
   (data.pos, data.connectM,
    DoubleVec3(0., 0., L/2 + 1E-6), DoubleVec3(0., 0., -1.), connectM2);

meshtools::medit_output("TH.mesh", data.pos, connectM2, CM2_TETRA4);
```

To constrain an internal surface, almost the same scheme has to be followed: generation of the internal surface mesh, merging with the nodes of the boundary mesh if needed, and insertion of the connectivity of these hard faces into the matrix `data.connectB` for instance with a push-back (see next example).

4. Cube with an internal hard node

There are several ways to get graded sizes in a mesh. The simplest way is to generate faces or edges with different or varying size on the boundary and interior surface or line, like in the previous example. The mesher computes a size value on each hard node[16], interpolates these values inside the domain and generates elements accordingly.

A second way is to specify manually, in the data of the tetrahedral mesher, the target size values on some or all the hard nodes. This is explained in this section.

So far, we have seen only four fields of the `data` structure used by the mesher:

- The `pos` matrix for the coordinates of the points.
- The `connectB` matrix for the connectivity of the hard faces.
- The `connectE` matrix for the connectivity of the hard edges (seen in Example 2).
- The `connectM` matrix for the connectivity of the output 3-D mesh.

In this example, we will add an isolated hard node (embedded) at the center of the cube and specify a target size for the elements near this node. This is done using two new fields: `isolated_nodes` and `metrics`:

```
#include "stdafx.h"

int main()
{
   const unsigned      N(6);
   const double        L(4.);
   DoubleMat           pos(3, 1, L/2.);
   UIntMat             connectB, connectB1, connectB2, connectE, connectM2;
   UIntVec             indices;

   // UNLOCK THE DLL.
   tetramesh_iso::registration("Licensed to SMART Inc.", "F53EA108BCWX");

   // BOUNDARY 2D MESH.
   meshtools2d::extrude_translate(pos, DoubleVec3(0,0,L), DoubleVec3(L,0,0), N,
                                  indices);
   meshtools2d::indices_to_connectE2(indices, connectE);
   meshtools2d::extrude_translate_T3(pos, connectE, DoubleVec3(0,L,0), N, 2,
                                     connectB);

   meshtools::copy_mesh(pos, connectB1, connectB);
   meshtools::rotate(pos, DoubleVec3(L/2,L/2,L/2), DoubleVec3(0,M_PI,0), connectB1);
   connectB.push_back(connectB1);

   meshtools::copy_mesh(pos, connectB1, connectB);
   meshtools::rotate(pos, DoubleVec3(L/2,L/2,L/2), DoubleVec3(M_PI/2,0,0), connectB1);
   connectB.push_back(connectB1);

   meshtools::copy_mesh(pos, connectB2, connectB1);
   meshtools::rotate(pos, DoubleVec3(L/2,L/2,L/2), DoubleVec3(0,0,M_PI/2), connectB2);
   connectB.push_back(connectB2);
   meshtools::merge(pos, connectB, /*tol=>*/ 1E-6, /*merge_type=>*/ 0);

   // 3D MESH.
   tetramesh_iso::mesher              the_mesher;
   tetramesh_iso::mesher::data_type   data(pos, connectB);
   data.isolated_nodes.push_back(0);
   data.metrics.resize(1, 0.05*L/N);
   the_mesher.run(data);

   // MESH VISUALISATION.
   meshtools::clip(data.pos, data.connectM,
                   DoubleVec3(0., 0., L/2+1E-6), DoubleVec3(0., 0., -1.), connectM2);
   meshtools::medit_output("TH.mesh", data.pos, connectM2, CM2_TETRA4);

   return 0;
}   // main
```

Initializing `pos` as a 3 x 1 matrix with the value L/2 simply creates the central node[17]. This node is taken into account by the mesher when its index (here zero) is inserted into the vector `data.isolated nodes`:

```
data.isolated_nodes.push_back(0);
```

The vector `data.metrics` stores the user-specified elements' size. If the size value for a node is zero (or negative or not present), the automatically computed value will be used instead[18]. In this example the vector is resized to 1 to set only a value for the node #0. We ask for a 20 times finer mesh around it:

```
data.metrics.resize(1, 0.05*L/N);
```

For all the other nodes, we let the mesher compute suitable values.

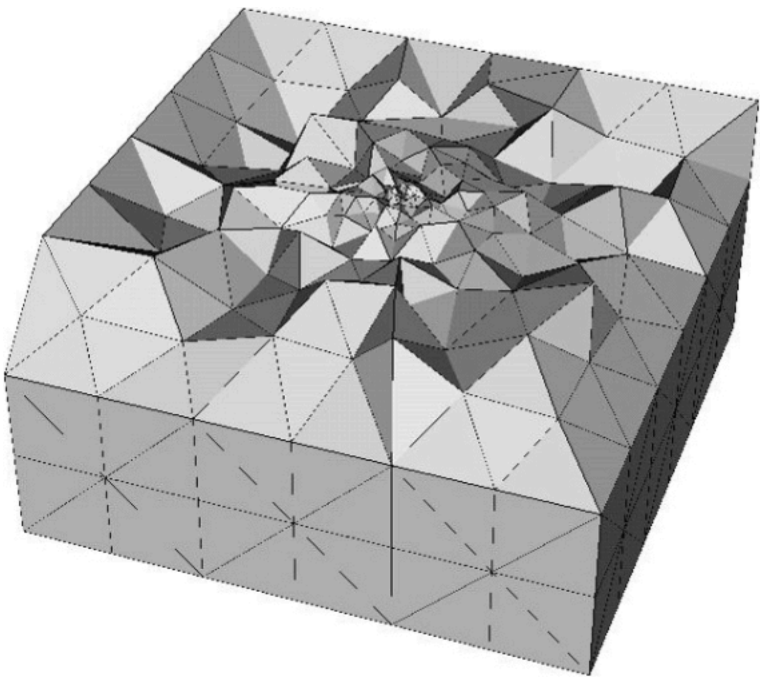


Figure 8 – Half cube with internal hard node (mesh cut).

[16] By averaging the lengths of the adjacent edges to each hard node and the inscribed radius of the triangles.

[17] Another way to do this is from an empty `pos` matrix is: `pos.push_back(DoubleVec3(L/2, L/2, L/2));`

[18] For an isolated node, the computed size is based on the values of the nearest nodes.

5. Internal cavity

A cavity is an internal closed surface with triangles oriented the opposite way from the external boundary. This implies that all triangles of the external boundary are similarly oriented[19]. To illustrate this point, we assume we have the boundary mesh of a sphere ready on a file. The code for this example is similar to that of Example 1.

```
#include "stdafx.h"
#include <fstream>

int main()
{
   DoubleMat           pos;
   UIntMat             connectB;
   std::ifstream       istrm("sphere.dat");

   // UNLOCK THE DLL.
   tetramesh_iso::registration("Licensed to SMART Inc.", "F53EA108BCWX");

   // READ THE BOUNDARY TRIANGLE MESH.
   matio::read(istrm, pos);
   matio::read(istrm, connectB);

   // 3D MESH.
   tetramesh_iso::mesher                the_mesher;
   tetramesh_iso::mesher::data_type     data(pos, connectB);
   the_mesher.run(data);

   // MESH VISUALISATION.
   meshtools::medit_output("TH.mesh", data.pos, data.connectM, CM2_TETRA4);

   return 0;
}    // main
```

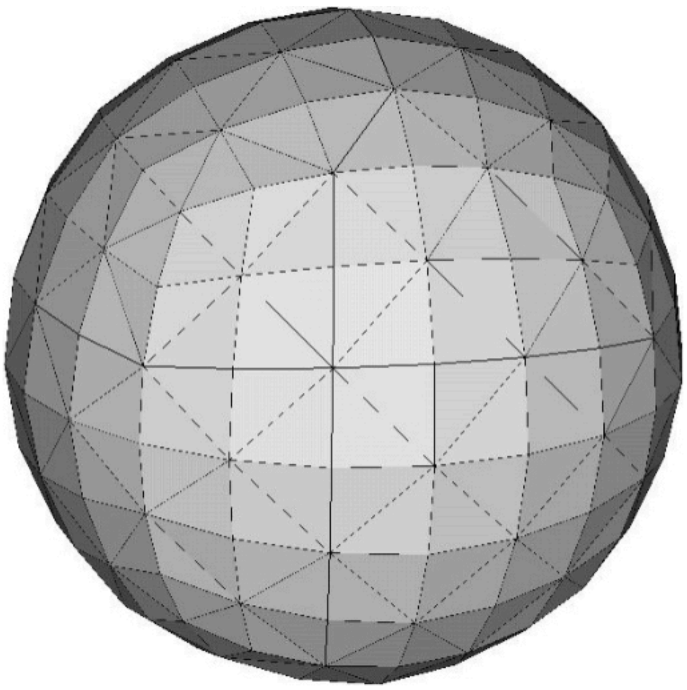


Figure 9 – Full sphere.

The surface of the cavity is generated by shrinking a copy of the external boundary (with ratio ½) and reversing the orientation of the triangles:

[19] Without any cavity, the orientation of the triangles of the external boundary can be arbitrary.

```
#include "stdafx.h"
#include <fstream>

int main()
{
   DoubleMat             pos;
   UIntMat               connectB, connectB2, connectM2;
   std::ifstream         istrm("sphere.dat");

   // UNLOCK THE DLL.
   tetramesh_iso::registration("Licensed to SMART Inc.", "F53EA108BCWX");

   // OUTER BOUNDARY TRIANGLE MESH.
   matio::read(istrm, pos);
   matio::read(istrm, connectB);

   // INNER BOUNDARY.
   meshtools::copy_mesh(pos, connectB2, connectB);
   meshtools::zoom(pos, DoubleVec3(0.), 0.5, connectB2);
   meshtools2d::flip_T3(connectB2);       // Reverse the orientation.
   connectB.push_back(connectB2);

   // 3D MESH.
   tetramesh_iso::mesher               the_mesher;
   tetramesh_iso::mesher::data_type    data(pos, connectB);
   the_mesher.run(data);

   // MESH VISUALISATION.
   meshtools::clip(data.pos, data.connectM, DoubleVec3(0.), DoubleVec3(0., 0., -1.), connectM2);
   meshtools::medit_output("TH.mesh", data.pos, connectM2, CM2_TETRA4);

   return 0;
}   // main
```

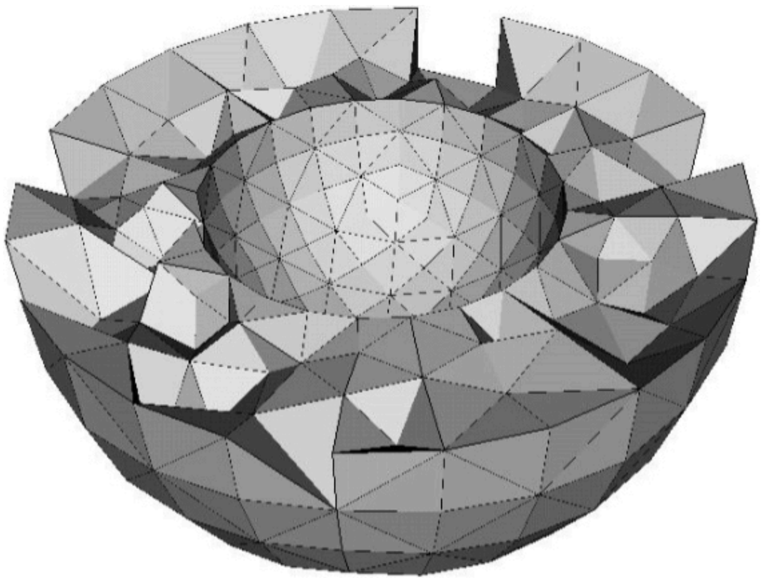


Figure 10 – Hollow sphere (mesh cut).

Note that the matrix `connectB` contains both the external and the internal faces.

Nested shells can be generated simply by changing the orientation of the spheres:

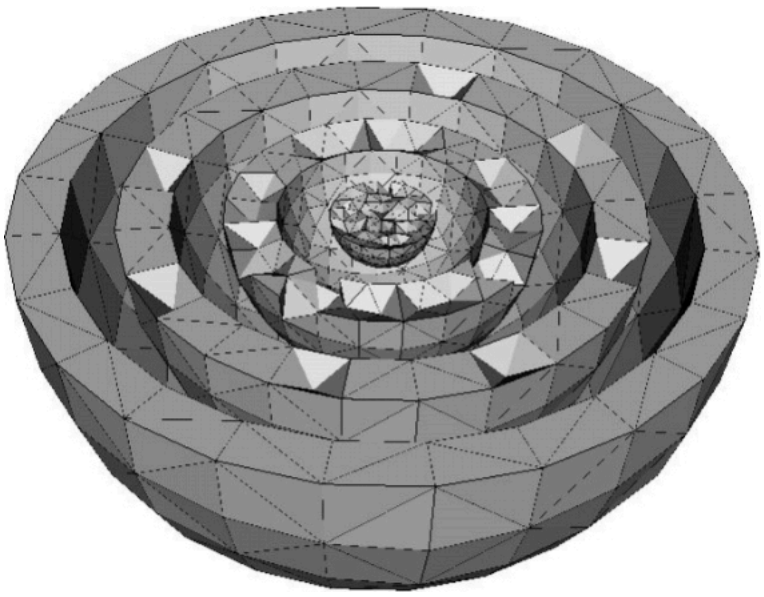


Figure 11 – Nested shells (mesh cut).

6. Quadratic elements & high-order nodes

Let us derive the Example 5 to generate quadratic TH10 elements. To make things simple, we assume here again that the surface mesh of the sphere (here T6 elements) are ready on a file to be read. These quadratic T6 elements can be curved or not.

```
#include "stdafx.h"
#include <fstream>

int main()
{
   DoubleMat            pos;
   UIntMat              connectB, connectT3;
   std::ifstream        istrm("sphere_t6.dat");

   // UNLOCK THE DLL.
   tetramesh_iso::registration("Licensed to SMART Inc.", "F53EA108BCWX");

   // READ THE BOUNDARY TRIANGLE MESH (EXPECTED T6 FACES).
   matio::read(istrm, pos);
   matio::read(istrm, connectB);
   connectT3 = connectB.sub_matrix(0, 0, 3, connectB.cols());  // Linear T3s.

   // 3D MESH.
   tetramesh_iso::mesher              the_mesher;
   tetramesh_iso::mesher::data_type   data(pos, connectT3);
   the_mesher.run(data);
   meshtools3d::convert_into_quadratic(data.pos, data.connectM, connectB, UIntMat());

   // VISUALISATION.
   meshtools::medit_output("TH.mesh", data.pos, data.connectM, CM2_TETRA10);

   return 0;
}   // main
```

The mesh generator accepts only *linear* faces upon entry and give only *linear* solid elements upon exit. Hence, we have to feed the mesher with the linear view of the `connectB` connectivity matrix, called `connectT3` in the above example (view to the first 3 rows from index 0, 0)[20].

After the solid meshing, to transform the TH4 mesh into a TH10 mesh we could simply call[21]:

```
meshtools3d::convert_into_quadratic(data.pos, data.connectM);
```

But in order to reuse the quadratic nodes along the boundaries (and keeping curved faces if any), we use an overload function with additional matrix parameters:

```
meshtools3d::convert_into_high_order(data.pos, data.connectM, connectB, UIntMat());
```

This forces `convert_into_quadratic` to use the high-order nodes of `connectB` wherever faces match (`connectB` is allowed to contain outer boundary faces but also inner embedded boundary faces or any face of the tetrahedrons in `data.connectM`).

[20] The two matrices share the same data. Only dimensions differ (number of rows and leading dimension).

[21] A more general function is available to convert into any type of high-order elements:. `cm2::meshtools3d::convert_into_high_order`. Refer to the HTML reference manual for detailed information.

The connectivity matrix `connectB` has 6 rows. The first three rows are the linear view (first three nodes of the linear faces)[22].

Linear nodes	0	1	2	5	5	1	3	4
	1	3	3	2	0	6	6	3
	2	2	4	4	2	3	7	7
	8	11	13	14	9	18	20	21
	9	8	14	15	16	12	21	22
	10	12	11	16	17	19	18	13

Figure 12 – Example of connectivity matrix for quadratic faces (`connectB`) and view to linear faces (`connectT3`).

The connectivity matrix `data.connectM` has 4 rows after the TetraMesh run, 10 rows after `convert_into_quadratic`. The first four rows are the linear view (first four nodes of the linear tetrahedrons).

[22] The empty matrix parameter stands for specific high-order edges. Here none of them.

7. Multiple meshes

As seen before, matrix `connectB` can contain several internal surfaces. It can also contain several external boundary surfaces. This means that several disconnected domains can be meshed simultaneously. As in the previous example, some care must be taken in the orientation of these surfaces. They must be oriented the same way (for instance all normal outside) and these boundaries must not cross each other.

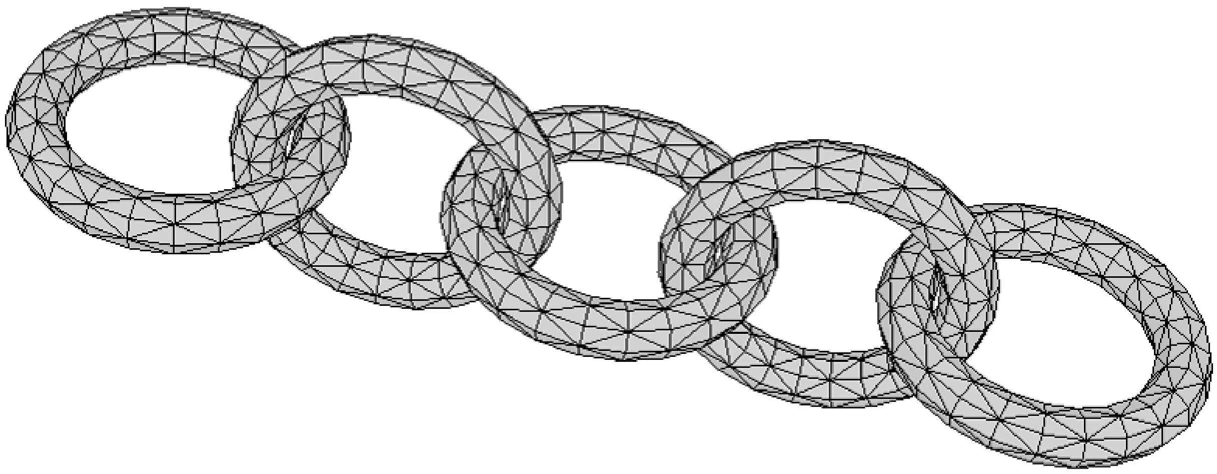


Figure 13 – Multiple disconnected sub-domains.

8. Repairing the boundary mesh

It is a well-known CAD problem that the boundary meshes may sometimes be not perfectly closed (not watertight). In addition, gaps and overlapping elements can be found in 3-D surface meshes.

CM2 TetraMesh Iso/Aniso have a correction algorithm that can repair some of these pathologies[23].

To illustrate this point we take two cylinders that intersect each other. The cylinders are generated by simple extrusion of a circle. They are not closed at their extremities and several triangles intersect each others. Some nodes are also coincident.

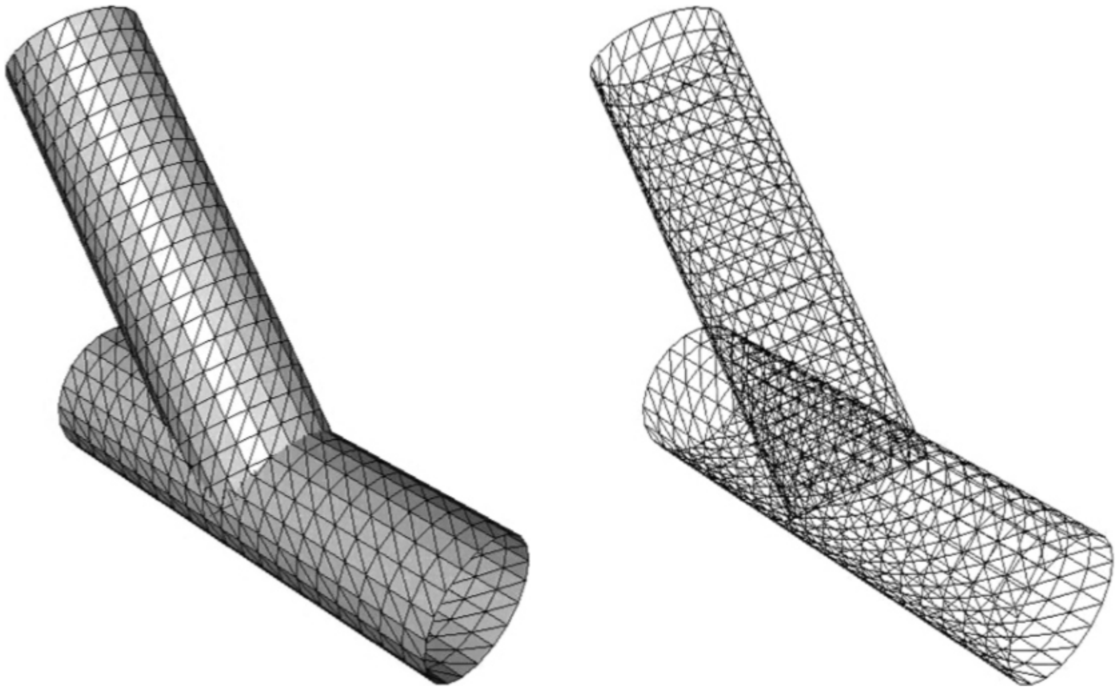


Figure 14 – Intersecting meshes.

In strict-constraint mode (see reference manual), the generators would stop with an error. In non-strict mode, they remove the intersecting and overlapping triangles and fill all the gaps.

[23] This algorithm is effective only in `MESH_MODE` and non-strict constraints mode (see reference manual).

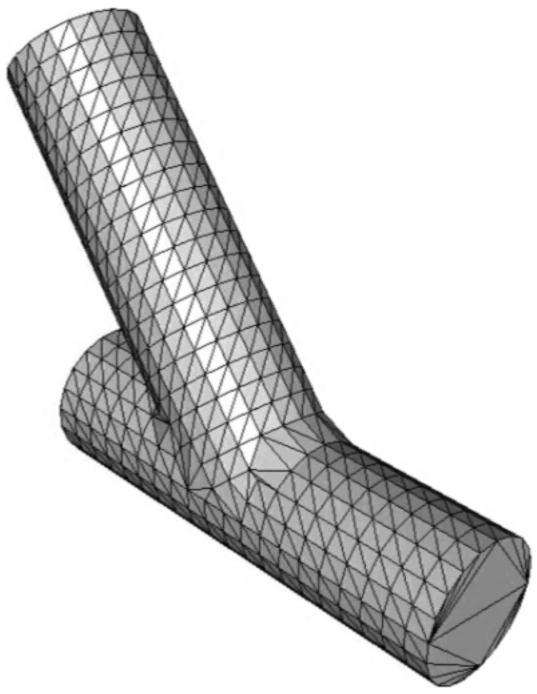


Figure 15 – Repaired mesh.

Note that the repairing algorithm uses only the hard nodes of the initial mesh and usually change the geometry of the surface. A more specific tool, called **CM2 Intersect® T3** can fix intersections much more precisely by adding new nodes at intersection points (but doesn't close open gaps). See **CM2 Intersect T3 - tutorials**.

9. Background mesh

Sometimes it is not convenient to specify the target mesh sizes at some hard nodes. This is the case especially when automatic mesh adaptivity is involved. The *background mesh* option is the solution in this case.

The background mesh is an auxiliary mesh used by the mesher to find the target mesh size at any point inside the domain. It's represented by the connectivity matrix background_mesh in the data of the mesher.

As always, the indices of the nodes refer to columns in the same pos matrix as all other connectivity matrices or vectors (such as connectM or connectB).

The nodes of the background mesh can be all different from the nodes of connectB or some can be common. They must all have a valid associated size value in the metrics array. The size map (also called metrics map) is interpolated inside the background mesh.

In the following example, a simple regular background mesh is used to support a sinusoidal metrics map varying in the three directions. The domain to be meshed is a simple cube regularly discretized on its boundaries.

To simplify the program, we use the auxiliary function cube_boundary to generate the boundary mesh of a cube centered at the origin.

```
#include "stdafx.h"


// Auxiliary function to generate the 6 faces of a cube centered at (0,0,0),
// with edge length equal to "L" and with "N" elements along each edge.
void
cube_boundary (double L, unsigned N, DoubleMat& pos, UIntMat& connectB)
{
   UIntVec            indices;
   UIntMat            connectE, connectB1, connectB2;

   meshtools2d::extrude_translate(pos, DoubleVec3(-L/2, -L/2, +L/2),
                                  DoubleVec3(L, 0., 0.), N, indices);
   meshtools2d::indices_to_connectE2(indices, connectE);
   meshtools2d::extrude_translate_T3(pos, connectE, DoubleVec3(0., L, 0.), N, 2, connectB2);

   meshtools::copy_mesh(pos, connectB1, connectB2);
   meshtools::rotate(pos, DoubleVec3(0.), DoubleVec3(0., M_PI, 0.), connectB1);
   connectB2.push_back(connectB1);

   meshtools::copy_mesh(pos, connectB1, connectB2);
   meshtools::rotate(pos, DoubleVec3(0.), DoubleVec3(M_PI/2, 0., 0.), connectB1);
   connectB2.push_back(connectB1);

   meshtools::copy_mesh(pos, connectB1, connectB1);
   meshtools::rotate(pos, DoubleVec3(0.), DoubleVec3(0., 0., M_PI/2), connectB1);
   connectB2.push_back(connectB1);

   meshtools::merge(pos, connectB2, /*tol=>*/ 1E-6, /*merge_type=>*/ 0);
   connectB.push_back(connectB2);
}


int main()
{
   const double                     L(4.), h0(0.5), h1(0.1);
   DoubleMat                        pos;
   UIntVec                          indices;
   UIntMat                          connectB1, connectB2, BGM, connectM2;
   DoubleVec                        sizes;
   unsigned                         N, N_BGM, n;
   double                           w, h;
   tetramesh_iso::mesher            the_mesher;
   tetramesh_iso::mesher::data_type dataTH;

   // UNLOCK THE DLL.
   tetramesh_iso::registration("Licensed to SMART Inc.", "F53EA108BCWX");

   // THE BOUNDARY OF THE BACKGROUND MESH
   N_BGM = unsigned(std::max(L/h0, L/h1));
```

```
    cube_boundary(L, N_BGM, pos, connectB1);

    // THE 3D BACKGROUND MESH.
    dataTH.pos = pos;
    dataTH.connectB = connectB1;
    the_mesher.run(dataTH);
    dataTH.extract(pos, BGM);

    // THE METRICS ON THE 3D BACKGROUND MESH.
    meshtools::unique_indices(indices, BGM);
    sizes.resize (pos.cols(), 0.);
    for (size_t i = 0; i < indices.size(); ++i)
    {
       n = indices[i];
       w = vecscal::max_norm(pos.col(n));
       h = ::cos(8.*M_PI* w/L) * (h0-h1)/2. + (h0+h1)/2.;
       sizes[n] = h;
    }

    // THE BOUNDARY OF THE FINAL MESH.
    cube_boundary(L, N, pos, connectB2);

    // THE 3D MESH.
    dataTH.pos = pos;
    dataTH.connectB = connectB2;
    dataTH.background_mesh = BGM;
    dataTH.metrics = sizes;
    the_mesher.run(dataTH);

    // MESH VISUALISATION.
    meshtools::clip(dataTH.pos, dataTH.connectM,
                    DoubleVec3(0., 0., -1E-6), DoubleVec3(0., 0., -1.), connectM2);
    meshtools::medit_output("TH.mesh", data.pos, connectM2, CM2_TETRA4);

    return 0;
 }   // main
```

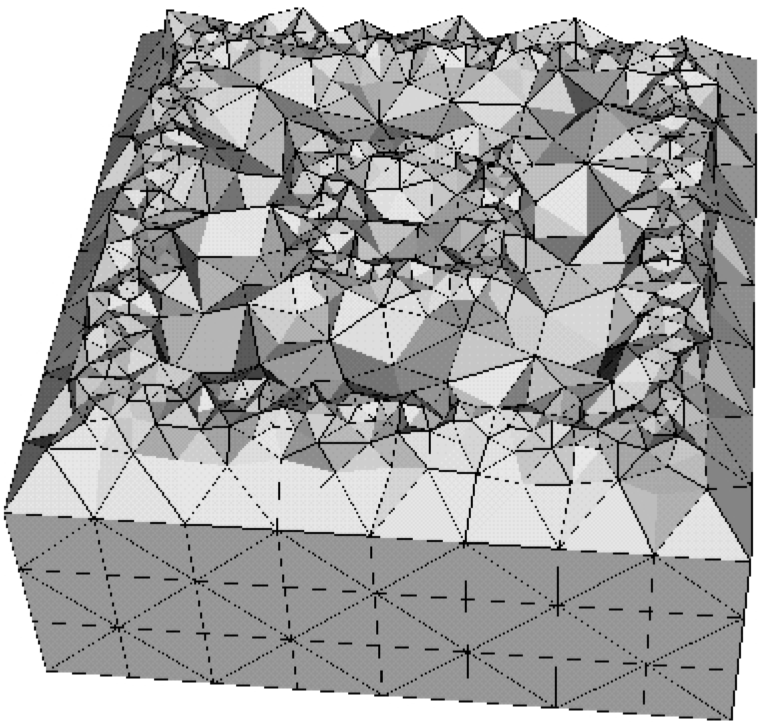


Figure 16 – Use of a background mesh to support a metric map on the domain (mesh cut).

The background mesh does not need to fit exactly the domain to be meshed. It can cover only a part of it or be partially outside of the domain. In the zones not covered by the background mesh, the default interpolation of the sizes from the hard nodes is used instead.

Here is an example where the domain and the background mesh are both spheres but the latter is half the radius of the former. We have set a uniform value for the metric map on the background mesh to get a finer mesh in this part. Hence, the metric drops abruptly at the limit of the background mesh, but the mesh is still conformal.

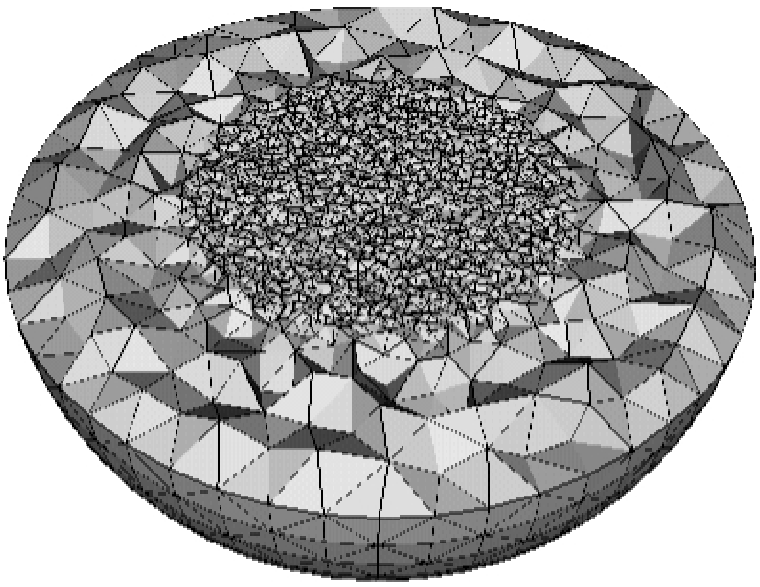


Figure 17 – Background mesh covering only a fraction of the domain (mesh cut).

The background mesh is a very useful feature to control precisely the size of the elements all over the domain. It can however be time consuming. The finer the background mesh, the longer the meshing process. The background mesh should be coarse wherever the metric is slowly varying and fine only in the regions where the metric is sharply varying and should be approximated with accuracy.

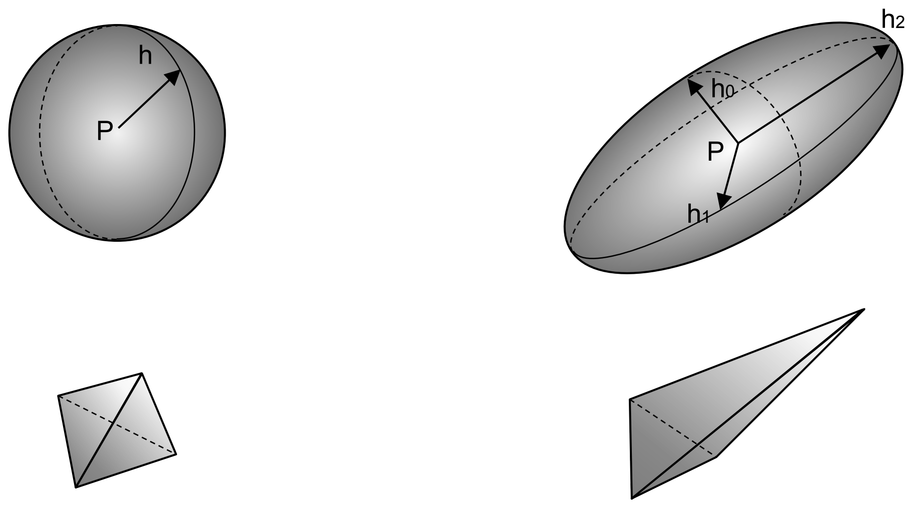
We can also consider the case where the boundary surface mesh of the domain must also be governed by background mesh. Two steps with two different background meshes are required[24]. First, the surface mesh is generated with the help of a surface background mesh. An anisotropic mesher supporting the background mesh option such as **CM2 TriaMesh® Aniso** can be used for this purpose (refer to the manual **CM2 TriaMesh - CM2 QuadMesh Iso/Aniso - tutorials**). After the surface mesh, we can proceed as the previous example to generate a 3-D background mesh supporting the metric map and run the tetrahedral mesher to get the final adapted 3-D mesh.

☞ A background mesh can be used also in `REGULARIZE_MODE` on a pre-existing mesh to adapt/optimize it to a changed metrics map (saving a new meshing from scratch).

[24] A third kind of background mesh can be needed to generate the line meshes.

10. Anisotropic meshes (CM2 TetraMesh Aniso)

CM2 TetraMesh Iso is an *isotropic* unstructured mesh generator: it tends to produce *equilateral* tetrahedrons. It's sometimes useful however to have elements *stretched* in some special directions. To deal with complex domains, we still need an unstructured mesher. Here comes the anisotropic unstructured mesh generator **CM2 TetraMesh Aniso**.
CM2 TetraMesh Aniso is almost identical to the isotropic version except for the `data.metrics` array. This array is now a matrix (instead of a vector). In the isotropic case, we need only a scalar at each node to define the size. Now, the size is defined by a 3 x 3 symmetric matrix at each node (6 doubles), stored column-wise in the `metrics` array.



Isotropic tet
(size 1 in the isotropic metric h)

Anisotropic tet
(size 1 in the anisotropic metric M)

Figure 18 – A single scalar defines an isotropic metric (left).
A 3-D anisotropic metric needs three vectors (right).

3-D anisotropic metrics are defined as below:

$$M_j = \begin{bmatrix} a & b & d \\ b & c & e \\ d & e & f \end{bmatrix} \qquad \text{with:} \quad \begin{array}{l} a > 0 \\ ac\text{-}b^2 > 0 \\ Det(M_j) > 0 \\ \text{i.e. the three given values are} > 0 \end{array}$$

$$data3D.metrics = \begin{bmatrix} . & . & . & a & . & . & . \\ . & . & . & b & . & . & . \\ . & . & . & c & . & . & . \\ . & . & . & d & . & . & . \\ . & . & . & e & . & . & . \\ . & . & . & f & . & . & . \end{bmatrix}$$

column #j

Figure 19 – Definition and storage of the 3-D anisotropic metrics.

Let (v0, v1, v2) be the three orthonormal vectors along the axes of the ellipsoid:

$$\begin{aligned}
&\|\mathbf{v}_0\| = \|\mathbf{v}_1\| = \|\mathbf{v}_2\| = 1 \\
&\langle \mathbf{v}_0, \mathbf{v}_1 \rangle = 0 \\
&\langle \mathbf{v}_0, \mathbf{v}_2 \rangle = 0 \\
&\langle \mathbf{v}_1, \mathbf{v}_2 \rangle = 0 \\
&\langle \mathbf{v}_0 \times \mathbf{v}_1, \mathbf{v}_2 \rangle = 1
\end{aligned}$$

Then, the metric M_j writes:

$$M_j = \mathbf{B} \begin{bmatrix} \frac{1}{h_0^2} & 0 & 0 \\ 0 & \frac{1}{h_1^2} & 0 \\ 0 & 0 & \frac{1}{h_2^2} \end{bmatrix} {}^T\mathbf{B} \qquad \text{with:} \quad \begin{aligned} &\mathbf{B} = [\mathbf{v}_0 \quad \mathbf{v}_1 \quad \mathbf{v}_2] \\ &\textit{stored column-wise} \end{aligned}$$

The h_0, h_1 and h_2 values are the target sizes of the tets in the three principal directions near point P_j. They are the inverse of the square root of the eigen values of metric M_j.

The 3-D metric equivalent of an isotropic size of *h* writes:

$$M_j = \begin{bmatrix} \frac{1}{h^2} & 0 & 0 \\ 0 & \frac{1}{h^2} & 0 \\ 0 & 0 & \frac{1}{h^2} \end{bmatrix}$$

A null matrix would lead to infinite sizes in the three directions (infinite sphere).

When the user doesn't specify any metric, the mesher uses the default one which is equivalent to the isotropic case we have seen before. For each hard node, the default metric is based on the length of the adjacent edges. This leads to the same default behavior as their related isotropic counterparts. Take the previous examples and replace:

```
tetramesh_iso::mesher     the_mesher;
```

with

```
tetramesh_aniso::mesher   the_mesher;
```

and you get the same output meshes[25].

[25] The anisotropic mesher is much slower than its isotropic counterpart though (about 6 times slower).

The following example shows a long "beam" as a parallelepiped of size 10x10x100 discretized with 10 elements along each direction. To get stretched tets in the direction of the beam, we need also to specify an anisotropic metric. Here, the metric is uniform on all the beam and equals simply:

$$M_j = \begin{bmatrix} \frac{1}{h_0^2} & 0 & 0 \\ 0 & \frac{1}{h_0^2} & 0 \\ 0 & 0 & \frac{1}{h_1^2} \end{bmatrix} \qquad \text{with:} \qquad \begin{aligned} h_0 &= \frac{L_0}{N_0} \\ h_1 &= \frac{L_1}{N_1} \end{aligned}$$

Here, the directions of anisotropy are identical to the global axes X, Y, Z.

```
#include "stdafx.h"

int main()
{
{
   const double               L0(10.), L1(100.);
   const unsigned             N0(10), N1(10);
   DoubleMat                  pos;
   UIntVec                    indices;
   UIntMat                    connectE, connectB1, connectB2, connectB;
   UIntMat                    connectM, connectM2;
   DoubleMat                  metrics;
   double                     D0, D1;

   // UNLOCK THE DLL.
   tetramesh_aniso::registration("Licensed to SMART Inc.","F53EA108BCWX");

   // BOUNDARY 2D MESH.
   meshtools2d::extrude_translate(pos, DoubleVec3(0.),
                                  DoubleVec3(0., L0, 0.), N0, indices);
   meshtools2d::indices_to_connectE2(indices, connectE);
   meshtools2d::extrude_translate_T3(pos, connectE, DoubleVec3(L0, 0., 0.),
                                     N0, 2, connectB);

   meshtools::copy_mesh(pos, connectB1, connectB);
   meshtools2d::flip_T3(connectB1);
   meshtools::translate(pos, DoubleVec3(0., 0., L1), connectB1);
   connectB.push_back(connectB1);

   connectB1.clear();
   meshtools2d::extrude_translate_T3(pos, connectE, DoubleVec3(0., 0., L1),
                                     N1, 2, connectB1);
   meshtools::copy_mesh(pos, connectB2, connectB1);
   meshtools::rotate(pos, DoubleVec3(L0/2, L0/2, L1/2),
                     DoubleVec3(0,0,M_PI/2), connectB2);
   connectB1.push_back(connectB2);
   meshtools::copy_mesh(pos, connectB2, connectB1);
   meshtools::rotate(pos, DoubleVec3(L0/2, L0/2, L1/2),
                     DoubleVec3(0., 0., M_PI), connectB2);
   connectB1.push_back(connectB2);
   connectB.push_back(connectB1);

   meshtools::merge(pos, connectB, /*tol=>*/ 1E-6, /*merge_type=>*/ 0);

   // METRICS.
   metrics.resize(6, pos.cols(), 0.);

   D0 = 1. / ((L0 / N0) * (L0 / N0));
   D1 = 1. / ((L1 / N1) * (L1 / N1));

   for (size_t n = 0; n < pos.cols(); ++n)
   {
      metrics(0,n) = D0;    // Mxx
      metrics(1,n) = 0.0;   // Mxy
      metrics(2,n) = D0;    // Myy = Mxx
      metrics(3,n) = 0.0;   // Mxz
      metrics(4,n) = 0.0;   // Myz
      metrics(5,n) = D1;    // Mzz
   }

   // 3D MESH.
   tetramesh_aniso::mesher               the_mesher;
   tetramesh_aniso::mesher::data_type    data(pos, connectB);
   the_mesher.settings.compute_Qh_flag = true;
   data.metrics = metrics;
   the_mesher.run(data);
   data.extract(pos, connectM);
   data.print_info(&display_hdl);

   // MESH VISUALISATION.
   meshtools::clip(pos, connectM, DoubleVec3(L0/2 +1E-6, L0/2 +1E-6, 0.),
                   DoubleVec3(-1., 0., 0.), connectM2); // clip out X > 0
   meshtools::medit_output("TH.mesh", pos, connectM2, CM2_TETRA4);

   return 0;
}   // main
```

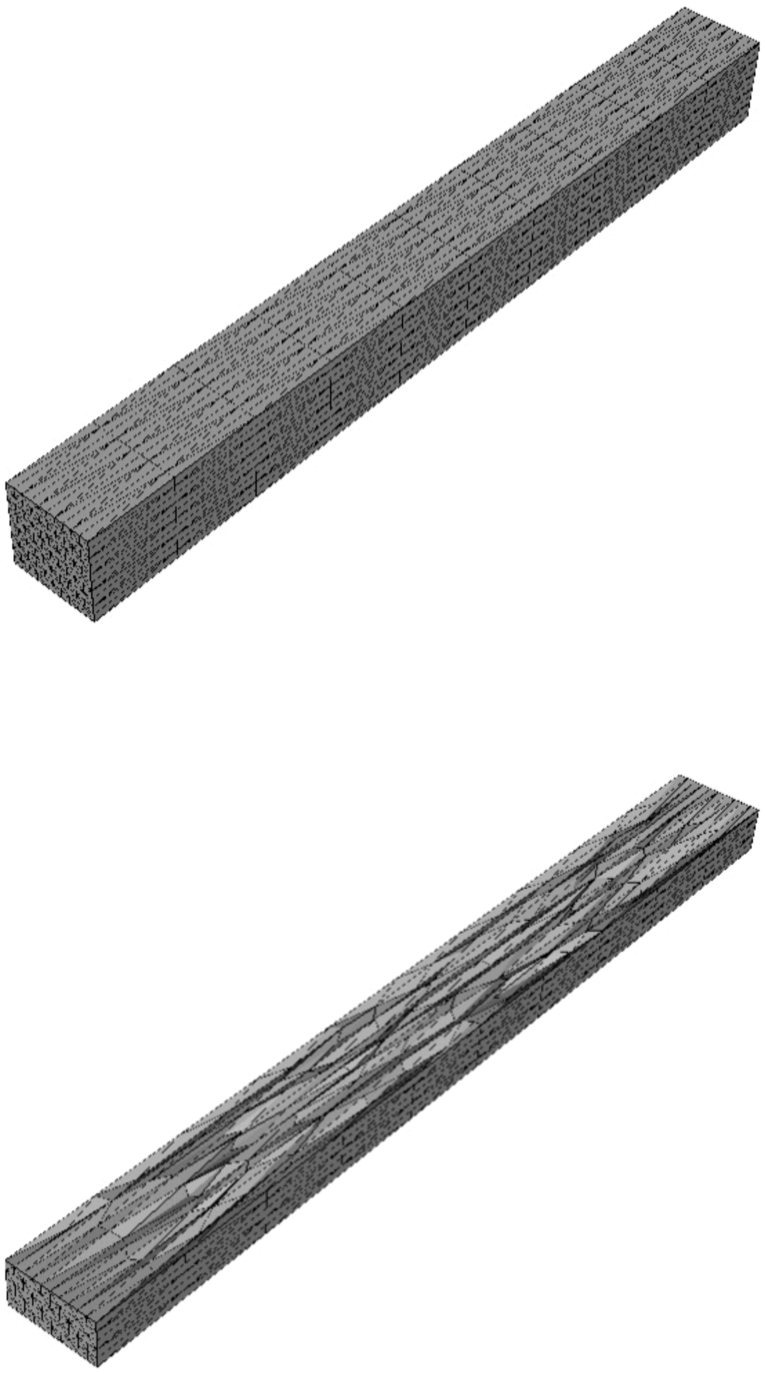


Figure 20 – An anisotropic beam. Complete model (up) and mesh cut (bottom).

Here is the output given by `data.print_info`. Note that the qualities (shape and edge) are computed using the local anisotropic metrics.

```
*******************************************************
*          CM2 TetraMesh Aniso(R) (5.6.0.0)           *
*******************************************************
Hard nodes      : 602/602
Hard edges      : 1800/1800
Hard faces      : 1200/1200
Nodes           : 1729
Tets            : 8360
Missing faces   : 0
Subdomains      : 1
Volume          : 1.000000E+04
Qmin            : 4.381141E-01 (max-min: 7.821716E-01)
Steiner nodes   : 0
Front time      : 0.02 s.
Refine time     : 0.12 s.
Optim time      : 0.06 s.
Total time      : 0.20 s. (42222.23 th/s.)

************ HISTOGRAM QS *************
Total number of bins     :           11
Total number of counts   :         8360
Number of larger values  :            0
Number of smaller values :            0
V max                    : 9.796000E-01
V mean                   : 7.451805E-01
V min                    : 4.381141E-01

Bin number       -- Bin boundaries --          Hits

   10            0.90            1.00           302
    9            0.80            0.90          2168
    8            0.70            0.80          3138
    7            0.60            0.70          2417
    6            0.50            0.60           334
    5            0.40            0.50             1
    4            0.30            0.40             0
    3            0.20            0.30             0
    2            0.10            0.20             0
    1            0.01            0.10             0
    0            0.00            0.01             0

NODES      : 1729
NEFS       : 8360
TIME       : 0.20 s.
NEFS / s   : 42222.23
```

Figure 21 – Output info. Shape qualities are computed in the anisotropic metrics.

```
************ HISTOGRAM QH *************
Total number of bins     :           20
Total number of counts   :        10688
Number of larger values  :            0
Number of smaller values :            0
V max                    : 1.589670E+00
V mean                   : 1.073384E+00
V min                    : 5.940392E-01

Bin number       -- Bin boundaries --          Hits

   19           10.00            +INF             0
   18            5.00           10.00             0
   17            3.33            5.00             0
   16            2.50            3.33             0
   15            2.00            2.50             0
   14            1.67            2.00             0
   13            1.43            1.67            83
   12            1.25            1.43          1689
   11            1.11            1.25          1979
   10            1.00            1.11          2737
    9            0.90            1.00          2741
    8            0.80            0.90          1034
    7            0.70            0.80           396
    6            0.60            0.70            28
    5            0.50            0.60             1
    4            0.40            0.50             0
    3            0.30            0.40             0
    2            0.20            0.30             0
    1            0.10            0.20             0
    0            0.00            0.10             0
```

Figure 22 – Edge quality histogram in the anisotropic metrics.

In the next example the metric is not uniform. A sphere (radius R) is uniformly meshed on its boundary. We used the following metric map:

$$\mathbf{M}(r) = \mathbf{B}(r) \begin{bmatrix} \frac{1}{h_0^2} & 0 & 0 \\ 0 & \frac{1}{h(r)^2} & 0 \\ 0 & 0 & \frac{1}{h(r)^2} \end{bmatrix} \mathbf{B}(r)^T$$

with:

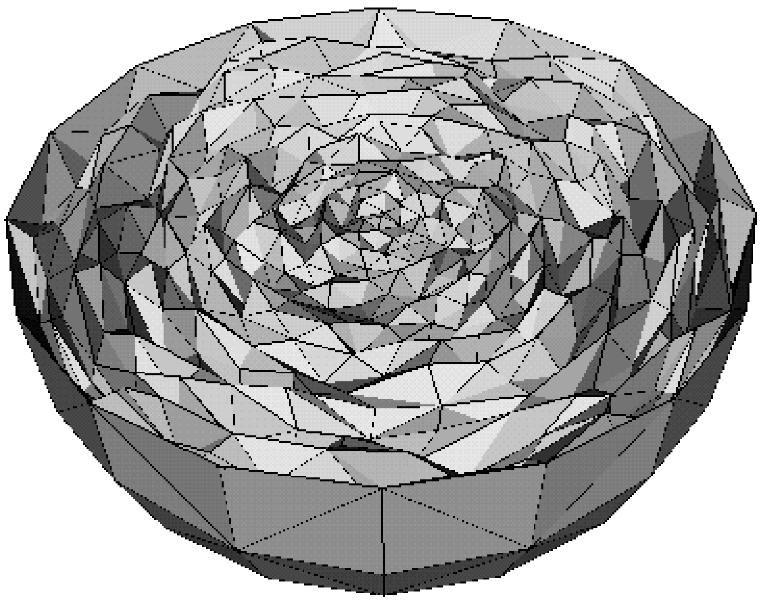
$$\mathbf{B}(r) = \left[\mathbf{u_r}(r) \quad \mathbf{v_\theta}(r) \quad \mathbf{w}_\varphi(r)\right]$$

the local spheric basis.

$$h(r) = h_1 \frac{r}{R} + h_0 \left(1 - \frac{r}{R}\right)$$

Radially we want the constant size h_0 but tangentially the size varies between h_0 at center and h_1 on the boundary of the sphere.

The mesher input metric map is only given at the boundary nodes of the external sphere and at the center node. The mesher will do the interpolation in-between.



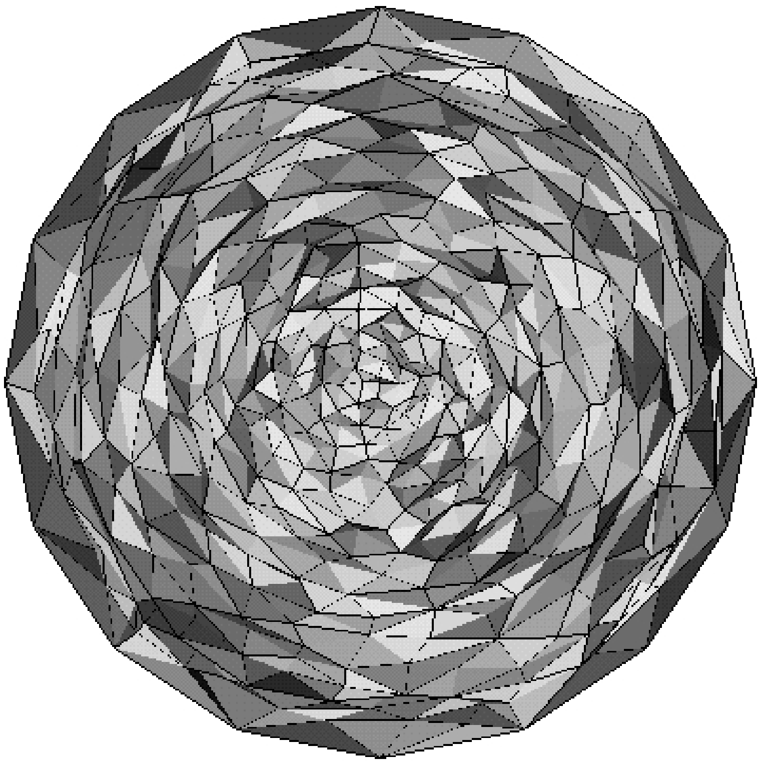


Figure 23 – An anisotropic sphere (mesh cut).

Background meshes can also be used with the 3-D anisotropic mesher in the same way as the isotropic counter-part (see Example 9). Recall that the metric matrix at each node of the background mesh must be valid (three strictly positive eigen-values).

COMPUTING
OBJECTS

CM2 TetraMesh® Iso

Version 5.6

mesh gallery

Revision February 2025.

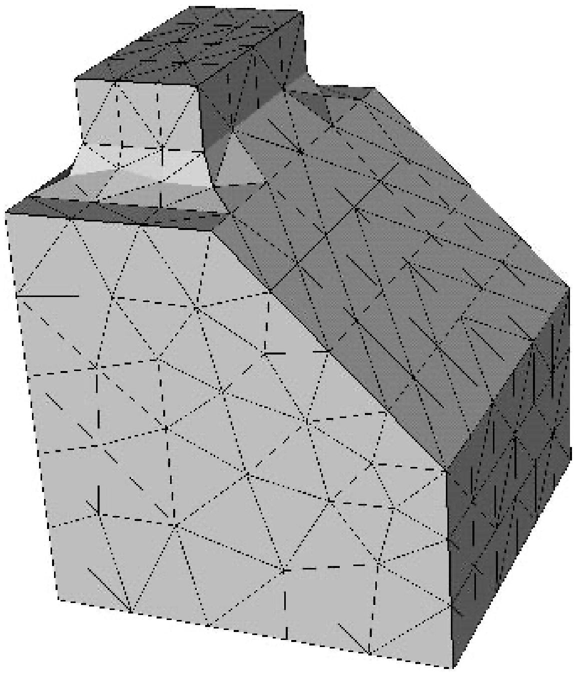


Figure 1 – Block-stop.

```
Nodes            : 270
Tets             : 883
Missing faces    : 0
Subdomains       : 1
Volume           : 9.277810E-01
Qmin             : 3.409671E-01 (max-min: 5.641645E-01)
Steiner nodes    : 0
Front time       : 0.00 s.
Refine time      : 0.01 s.
Optim time       : 0.00 s.
Total time       : 0.01 s. (63071.70 th/s.)
```

The output information given here are only indicative. All runs were done with x64 CM2 libs (Visual Studio 2022 MD build) on Windows 8.1 x64 with Intel Xeon E3-1270 V2 3.5 GHz (4 cores with hyper-threading, turbo boost disabled).

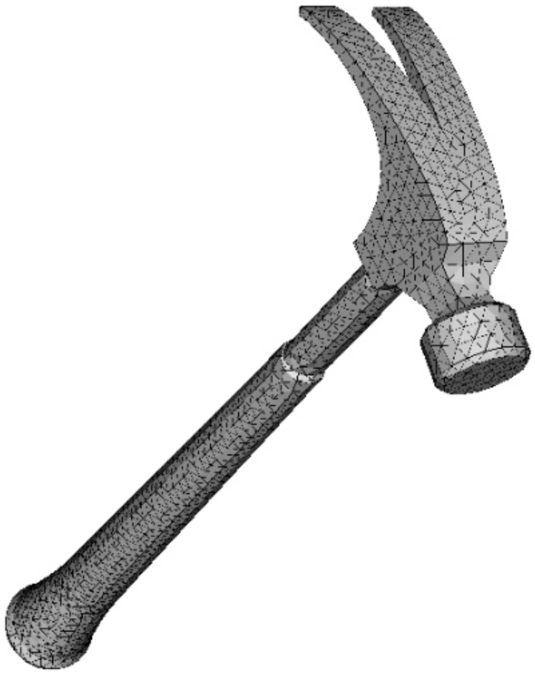


Figure 2 – Hammer.

```
odes           : 3027
Tets            : 12031
Missing faces   : 0
Subdomains      : 1
Volume          : 2.019083E+11
Qmin            : 3.017365E-02 (max-min: 3.031752E-02)
Steiner nodes   : 0
Front time      : 0.07 s.
Refine time     : 0.08 s.
Optim time      : 0.01 s.
Total time      : 0.16 s. (75193.71 th/s.)
```

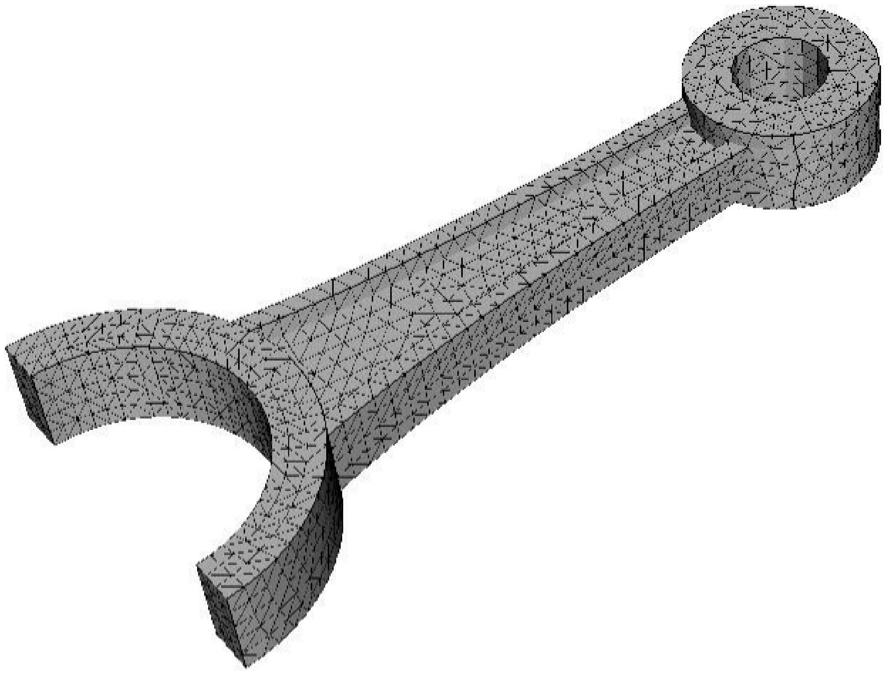


Figure 3 – Driving Rod.

```
Nodes           : 1844
Tets            : 5937
Missing faces   : 0
Subdomains      : 1
Volume          : 9.848692E-01
Qmin            : 2.677825E-01 (max-min: 6.300594E-01)
Steiner nodes   : 0
Front time      : 0.04 s.
Refine time     : 0.03 s.
Optim time      : 0.02 s.
Total time      : 0.09 s. (62494.88 th/s.)
```

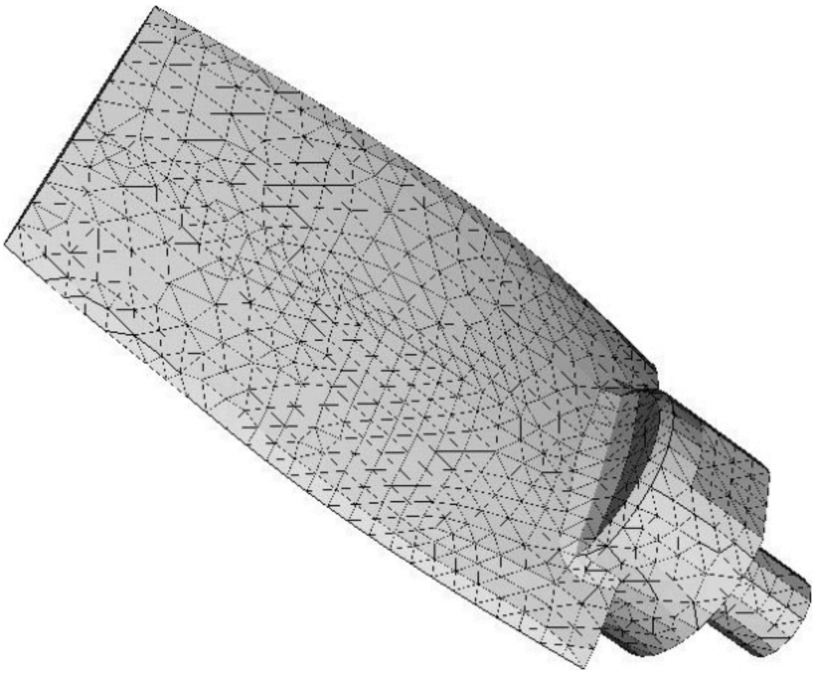


Figure 4 – Blade.

```
Nodes           : 1631
Tets            : 5542
Missing faces   : 0
Subdomains      : 1
Volume          : 1.149084E-01
Qmin            : 1.701167E-01 (max-min: 2.183494E-01)
Steiner nodes   : 0
Front time      : 0.04 s.
Refine time     : 0.03 s.
Optim time      : 0.01 s.
Total time      : 0.08 s. (70151.90 th/s.)
```

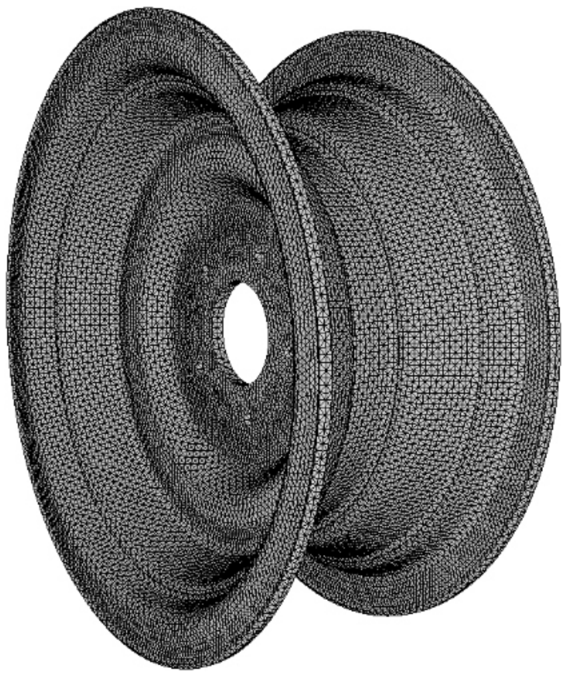


Figure 5 – Rim.

```
Nodes           : 63984
Tets            : 238849
Missing faces   : 0
Subdomains      : 1
Volume          : 6.174414E+06
Qmin            : 4.427687E-02 (max-min: 2.414112E-01)
Steiner nodes   : 2
Front time      : 2.55 s.
Refine time     : 1.14 s.
Optim time      : 0.78 s.
Total time      : 4.52 s. (52819.33 th/s.)
```

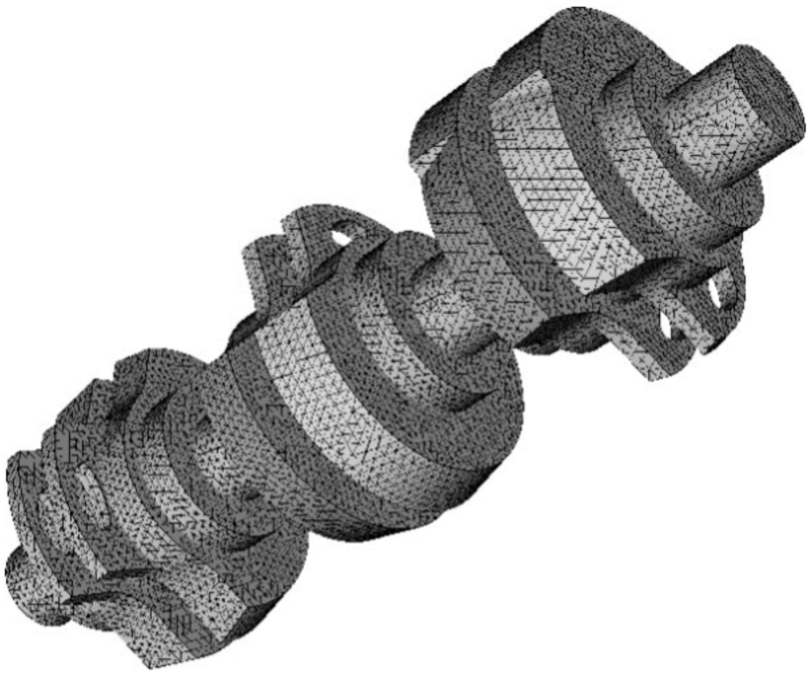


Figure 6 – Crank.

```
Nodes            : 50260
Tets             : 253949
Missing faces    : 0
Subdomains       : 1
Volume           : 1.170195E+05
Qmin             : 3.187160E-01 (max-min: 4.462804E-01)
Steiner nodes    : 0
Front time       : 0.65 s.
Refine time      : 2.04 s.
Optim time       : 1.34 s.
Total time       : 4.05 s. (62734.43 th/s.)
```

COMPUTING OBJECTS