COMPUTING OBJECTS

CM2 SurfRemesh® T3/Q4

Version 5.6

tutorials

Revision February 2025.

https://www.computing-objects.com

Forewords

This manual describes the 3-D surface remeshers of the **CM2 MeshTools®** SDK: **CM2 SurfRemesh® T3** and **CM2 SurfRemesh® Q4**.

CM2 SurfRemesh T3 and **CM2 SurfRemesh Q4** address the problem of regenerating a mesh on a 3-D surface defined by a discrete representation (initial surface mesh). This initial discrete representation may have elements with very high aspect ratio such as in STL meshes and even some topological errors (degenerated elements, holes, overlapping elements).

These remeshers can generate a mesh finer than the initial mesh (refinement) or coarser (decimation). They can also be used to regularize a 3-D surface mesh (node-smoothing only, local node inserting, local node removal...).

CM2 SurfRemesh Q4 can generate mixed quad-dominant meshes (the default) or all-quad meshes. The former mode is recommended since it usually gives better meshes.

If the surface to be remeshed is closed, the solid mesh generators **CM2 TetraMesh® Iso, CM2 TetraMesh® Aniso** and even **CM2 HexaMesh® Iso** can be used after **CM2 SurfRemesh T3** to fill the domain with solid elements.
After **CM2 SurfRemesh Q4**, only **CM2 HexaMesh® Iso** can be used to fill the domain with solid elements.
The reader can read more about **CM2 TetraMesh Iso/Aniso** in the manual **CM2 TetraMesh Iso/Aniso - tutorials** and about **CM2 HexaMesh® Iso** in the manual **CM2 HexaMesh Iso- tutorials.**

Like many other meshers of the library, **CM2 SurfRemesh T3/Q4** 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 SurfRemesh T3/Q4 - reference manual**.

The source code of the **CM2 MeshTools®** SDK (full library) has been registered with the APP under Inter Deposit number IDDN.FR.001.260002.00.R.P.1998.000.20700 (22/06/1998) and IDDN.
FR.001.250030.00.S.P.1999.000.20700 (16/06/1999) is regularly deposited since then.

The source code specific to **CM2 SurfRemesh® T3**, together with this manual, has been registered with the APP under Inter Deposit number IDDN.FR.001.450010.000.S.P.2007.000.20600 (9/11/2007) and is regularly deposited since then.

The source code specific to **CM2 SurfRemesh® Q4**, together with this manual, has been registered with the APP under Inter Deposit number IDDN.FR.001.440018.000.R.P.2008.000.20700 (31/10/2008) and is regularly deposited since then.

Table of contents

This manual shows examples of surface remeshing illustrating along the way some of the major options of the remeshers.

Each example starts with including the file `stdafx.h` (can be a pre-compiled header) giving access to the required 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::surfremesh_t3`. 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 "surfremesh_t3.h"  // CM2 SurfRemesh T3.
#include "surfremesh_q4.h"  // CM2 SurfRemesh Q4.

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

Required libraries[1]:

- `cm2math1`
- `cm2misc`
- `cm2meshtools`
- `cm2meshtools1d`
- `cm2meshtools2d`
- `cm2surfremesh_t3`
- `cm2surfremesh_q4`

[1] The lib names end with `_($platform)_($ver)`. For instance `cm2surfremesh_t3_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 – default mode remeshing

This first example is a regular cylinder with circular base. The diameter and the height of this cylinder are 100. The input mesh is a STL-mesh provided by a CAD modeler (Figure 1).

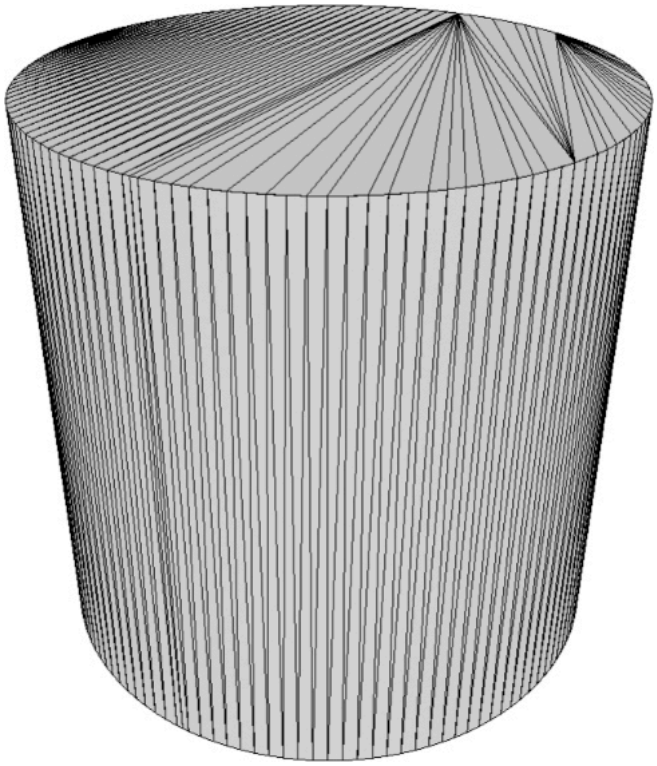


Figure 1 – Initial mesh of the cylinder (H = 100, D = 100).

Many triangles have high aspect ratio making this initial mesh not suited for FEA computations. The program to generate a new better mesh on this tessellated surface is shown below. Figure 2 shows the resulting mesh.

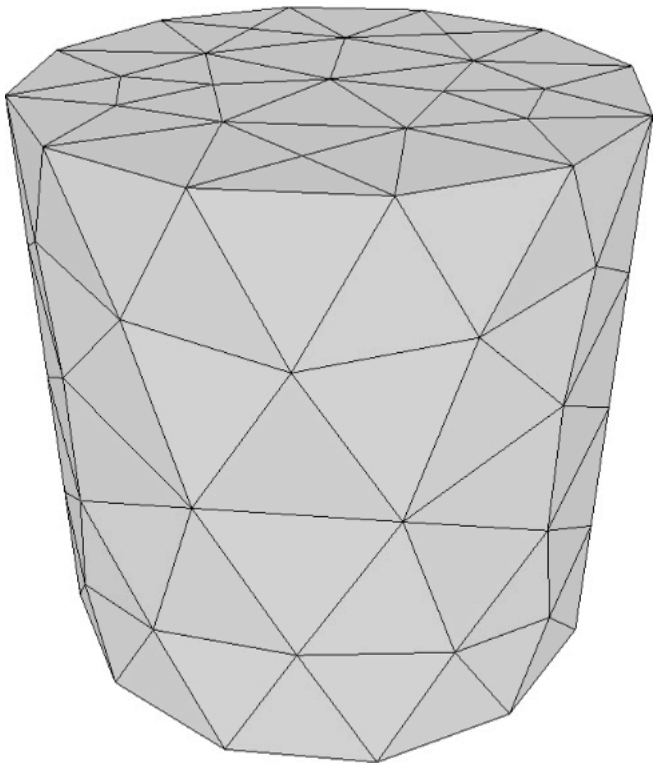


Figure 2 – The cylinder remeshed with default settings.

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

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


int main()
{
   surfremesh_t3::mesher               the_remesher;
   surfremesh_t3::mesher::data_type    data;

   // UNLOCK THE DLL.
   surfremesh_t3::registration("Licensed to SMART Inc.", "0BEE453363E8");

   // READ THE INITIAL MESH (STL FORMAT).
   meshtools::STL_input("cylinder.stl", data.pos, data.connectM);

   // REMESH THE SURFACE.
   the_remesher.run(data);            // Remesh the surface with default settings.

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

   // VISUALIZATION (OPTIONAL).
   meshtools::medit_output("out.mesh", data.pos, data.connectM, CM2_FACET3);

   return 0;
}   // main
```

Let us explain this program line by line.

Some declarations

`the_remesher` is an instance of `surfremesh_t3::mesher`.
`data` is a key structure for the remesher. It stores, among others, the matrix `pos` which contains all the points' coordinates and the matrix `connectM` which is the connectivity matrix of the mesh.

`data.pos` stores all the nodes involved in the meshing process: the nodes of the initial mesh and the nodes of the final mesh (some of them may be in both). It is stored as a `DoubleMat` matrix (variable-sized matrix of doubles)[2].

`data.connectM` stores the connectivity matrix of the meshes (first of the initial mesh then of the final mesh after remeshing) as `UIntMat` matrices (natural integers). `connectM(i, j)` is the ith local node of the jth initial element. This integer is the column number in matrix `pos` where the coordinates of this node can be found[3].

Authorization of the library

The library needs to be unlocked through a call to `surfremesh_t3::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[4]. Note that both strings are case sensitive and the registration call must be made each time the library is loaded into memory and before the first run of the remesher.

```
surfremesh_t3::registration("Licensed to SMART Inc.", "F5BEA10ABCWX");
```

The initial mesh

In this example, the initial mesh is read from a STL file (ASCII). The `meshtools::STL_input` function reads the points' coordinates, the nodes indices of the triangles and merge together any coincident nodes.

```
meshtools::STL_input("cylinder.stl", data.pos, data.connectM);
```

The new mesh

```
the_remesher.run(data);
```

Upon exit, the matrix `data.pos` is bigger and contains the new nodes generated on the surface by the remesher. These new points are appended to the original matrix. The initial points of the input mesh are unchanged. The connectivity of the final mesh is stored in the matrix `data.connectM`, each column storing the indices of the nodes for an element.

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

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

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

Output information

Printed information about the generated mesh and a MEDIT[5] output file are obtained with:

```
data.print_info(&display_hdl);
meshtools::medit_output("out.mesh", data.pos, data.connectM, CM2_FACET3);
```

Figure 3 shows the output of `data.print_info(&display_hdl)`:

The new mesh has 98 nodes and 192 triangles and its quality is much better. In the initial mesh, the average quality was 0.042. In the final mesh, it is 0.88.

The `max_error_distance` is a measure of the Hausdorff distance between the two meshes. The nodes of the new mesh are located exactly onto the initial surface. It is not the case however for the centroid of the new elements or for the initial nodes. If the surface is not flat, the new mesh may indeed be some distance away from it. The `max_error_distance` is a measure of this maximum distance (so-called *Hausdorff distance*).

The times spent in the several steps of the meshing process are given in seconds[6].

The first step is for cleaning the initial mesh (node merging and edges swapping).

In the second step, the mesh is partitioned into groups of connected triangles (so-called *patches*) that can be remeshed individually. The boundaries of the patches are located where the angle between two adjacent triangles is greater than a prescribed value[7] (`patch_angle_tolerance` – see Example 4). The boundaries of these patches are called the *skeleton lines*. The singular nodes among the skeleton lines (nodes connected to one or three or more skeleton lines) are called *skeleton nodes*.

The skeleton lines and the patches are remeshed during the third step. The skeleton nodes remain present in the new mesh.

Finally, geometrical and topological optimizations are performed to improve the quality of the elements.

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

[6] 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).

[7] For the sake of clarity, we can consider that the angle between adjacent triangles is the only parameter used by the algorithm (it is indeed not the only one, but the most important).

```
*******************************************************
*           CM2 SurfRemesh(R) T3 (5.6.0.0)            *
*******************************************************
Initial Mesh
  Nodes          : 256
  Triangles      : 508
  Area           : 4.711434E+04
  Qavg           : 4.163685E-02
  Qmin           : 2.128816E-02
New Mesh
  Nodes          : 98
  Triangles      : 192
  Area           : 4.613917E+04
  Qavg           : 8.838215E-01
  Qmin           : 7.037609E-01
Max error dist. : 1.758797E+00
Cleaning time   : 0.00 s.
Partition time  : 0.02 s.
Remesh time     : 0.01 s.
Optim time      : 0.00 s.
Total time      : 0.03 s. (6997.00 t/s.)

************ HISTOGRAM QS ************
Total number of bins     :           11
Total number of counts   :          192
Number of larger values  :            0
Number of smaller values :            0
V max                    : 9.987371E-01
V mean                   : 8.838215E-01
V min                    : 7.037609E-01

Bin number        -- Bin boundaries --         Hits

   10            0.90             1.00          112
    9            0.80             0.90           32
    8            0.70             0.80           48
    7            0.60             0.70            0
    6            0.50             0.60            0
    5            0.40             0.50            0
    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
```

Figure 3 – Output info for the cylinder example.

Note that the initial connectivity matrix is overwritten by the new one in matrix `data.connectM` and some nodes in `data.pos` are no longer referenced[8]. To keep the initial mesh, the following code should be used instead:

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

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


int main()
{
   surfremesh_t3::mesher               the_remesher;
   surfremesh_t3::mesher::data_type    data;
   UIntMat                             connectMi;

   // UNLOCK THE DLL.
   surfremesh_t3::registration("Licensed to SMART Inc.", "0BEE453363E8");

   // READ THE INITIAL MESH (STL FORMAT).
   meshtools::STL_input("cylinder.stl", data.pos, data.connectM);
   connectMi.copy(data.connectM);     // Save initial mesh. Hard copy.

   // REMESH THE SURFACE.
   the_remesher.run(data);            // Remesh the surface.

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

   // VISUALIZATION (OPTIONAL).
   meshtools::medit_output("in.mesh", data.pos, connectMi, CM2_FACET3);
   meshtools::medit_output("out.mesh", data.pos, data.connectM, CM2_FACET3);

   return 0;
}   // main
```

The initial mesh is in `connectMi` and the new mesh in `data.connectM`. Indices in both matrices refer to columns in `data.pos` (containing nodes of the old and the new mesh).

Another way to do this could be:

[8] You can get ride of the nodes unused anymore with `cm2::meshtools::simplify`. Please refer to the HTML reference manual of CM2 MeshTools.

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

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


int main()
{
   surfremesh_t3::mesher               the_remesher;
   surfremesh_t3::mesher::data_type    data;
   DoubleMat                           pos;
   UIntMat                             connectMi, connectMr;

   // UNLOCK THE DLL.
   surfremesh_t3::registration("Licensed to SMART Inc.", "0BEE453363E8");

   // READ THE INITIAL MESH (STL FORMAT).
   meshtools::STL_input("cylinder.stl", pos, connectMi);

   // REMESH THE SURFACE.
   data.pos = pos;                     // Shallow copy.
   data.connectM.copy(connectMi);      // Hard copy.
   the_remesher.run(data);             // Remesh the surface.
   pos = data.pos;                     // Shallow copy.
   connectMr = data.connectM;          // Shallow copy.

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

   // VISUALIZATION (OPTIONAL).
   meshtools::medit_output("in.mesh", pos, connectMi, CM2_FACET3);
   meshtools::medit_output("out.mesh", pos, connectMr, CM2_FACET3);

   return 0;
}   // main
```

In the following example (*part #1*), the skeleton lines found by the algorithm are the natural ridges of the solid (and some lines inside the circular holes).

Here also, we do not specify any size value for the new mesh. We let the remesher compute default values based on the distance between each skeleton node.

Concerning the program, the main difference with the previous one concerns the reading of the initial mesh. The coordinates and the connectivity matrices are no longer in a STL format file, but are read with `matio::transpose_read` from a "matrix-formatted" file.

Here the expected format for the matrices is:

$$
\begin{array}{l}
n \ X \ m \ [\\
d_{0,0} \quad d_{0,1} \quad d_{0,2} \quad \ldots \ d_{0,m-1} \\
d_{1,0} \quad d_{1,1} \quad d_{1,2} \quad \ldots \ d_{1,m-1} \\
\ldots \\
d_{n-1,0} \quad d_{n-1,1} \quad d_{n-1,2} \ldots \ d_{n-1,m-1} \ ]
\end{array}
$$

The format for each component of the matrix is free.

For instance for the coordinate matrix (4 nodes, 3 coordinates per node):

```
4 X 3 [
  0 0.5 2.0
  0 1 1
  1E-3 -7 1.0
  3.4 -2.1 -1.0 ]
```

```
#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()
{
   surfremesh_t3::mesher               the_remesher;
   surfremesh_t3::mesher::data_type    data;

   // UNLOCK THE DLL.
   surfremesh_t3::registration("Licensed to SMART Inc.", "0BEE453363E8");

   // READ THE INITIAL MESH.
   std::ifstream       istrm(“part1.dat”);
   matio::transpose_read(istrm, data.pos);
   matio::transpose_read(istrm, data.connectM);

   // REMESH THE SURFACE.
   the_remesher.run(data);

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

   // VISUALIZATION (OPTIONAL).
   meshtools::medit_output("out.mesh", data.pos, data.connectM, CM2_FACET3);

   return 0;
}   // main
```

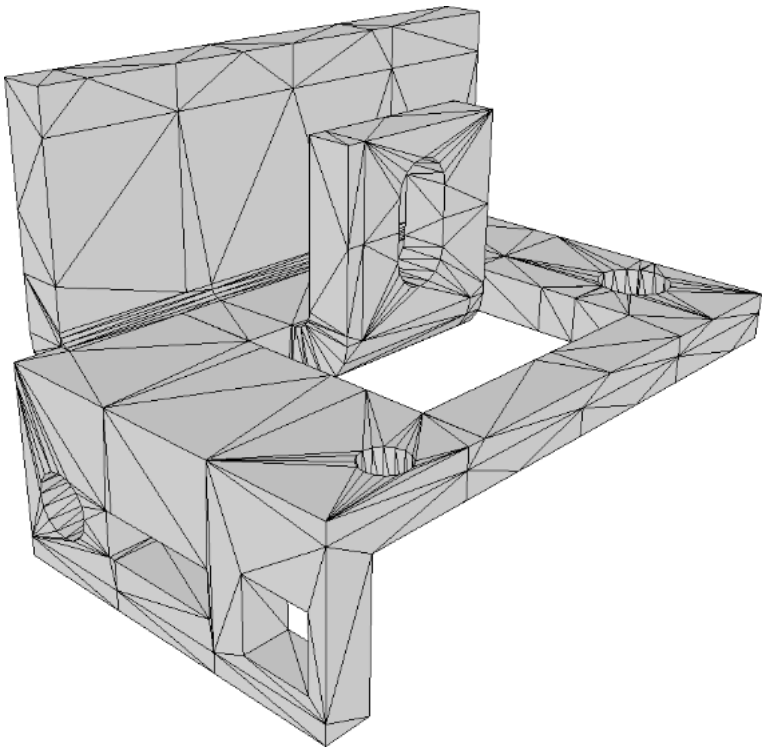


Figure 4 – Part #1 - Initial mesh (bounding box X: 0.10, Y: 0.10, Z: 0.15).

```
New Mesh
  Nodes          : 3051
  Triangles      : 6126
  Area           : 4.427879E-02
  Qavg           : 8.308896E-01
  Qmin           : 1.598984E-01

Max error dist. : 1.947421E-04
```

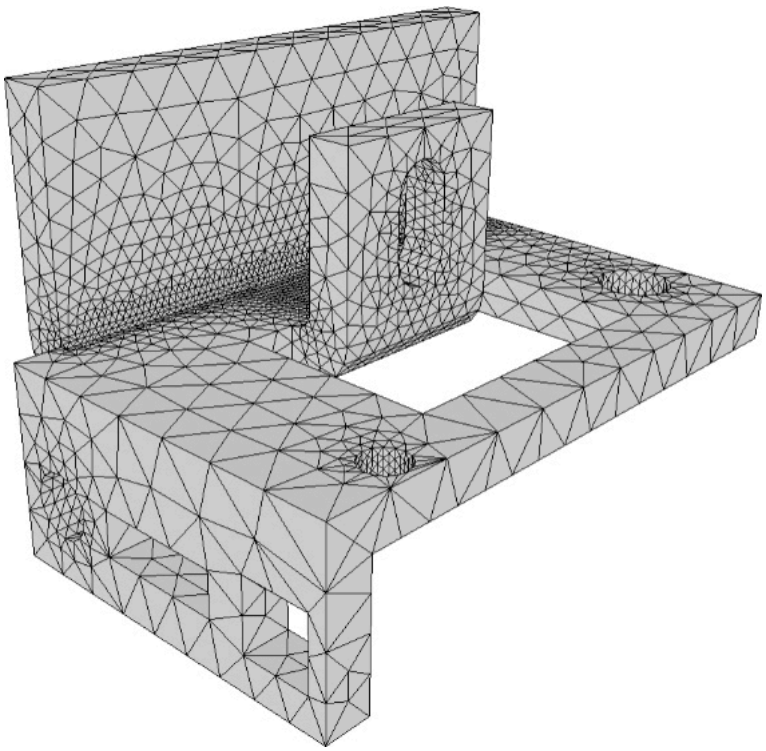


Figure 5 – Part #1 remeshed with default settings.

```
New Mesh
  Nodes          : 3051
  Triangles      : 6126
  Area           : 4.427879E-02
  Qavg           : 8.308896E-01
  Qmin           : 1.598984E-01

Max error dist. : 1.947421E-04
```

In this example where most of the patches are flat, there is a very small geometric (Hausdorff) error between the two meshes.

Note also the variations of the elements' size in the new mesh. It is a typical result with the default settings of the surface remesher. The next example addresses this point.

2. Limiting the range of the elements' size

Usually, the default elements' size does not give satisfactory meshes. They are either too fine or too coarse. For this matter, the user can limit the range of the elements' size with two settings: `min_h` and `max_h`. In the default settings, `min_h` = 0 and `max_h` = DBL_MAX.

Let us use again the cylinder example. We now set the two bounds to the same value: 10.

This will give a uniform mesh in which all elements' edges have a length of 10 on average.

```
#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()
{
   surfremesh_t3::mesher               the_remesher;
   surfremesh_t3::mesher::data_type    data;

   // UNLOCK THE DLL.
   surfremesh_t3::registration("Licensed to SMART Inc.", "0BEE453363E8");

   // READ THE INITIAL MESH.
   std::ifstream       istrm(“part1.dat”);
   matio::transpose_read(istrm, data.pos);
   matio::transpose_read(istrm, data.connectM);

   // REMESH THE SURFACE.
   the_remesher.settings.min_h = 10.;
   the_remesher.settings.max_h = 10.;
   the_remesher.run(data);

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

   // VISUALIZATION (OPTIONAL).
   meshtools::medit_output("out.mesh", data.pos, data.connectM, CM2_FACET3);

   return 0;
}   // main
```

☞ A third parameter (`target_h`) can also be used. This value is the size of the elements inside the patches (away from the skeleton lines). If left to its default (zero), the mesh size inside patches is driven by the edge sizes of the skeleton lines (between `min_h` and `max_h`)
We recommend usually to set `target_h` = `max_h`. This can give meshes inside the patches somewhat coarser than with the default null value (if `min_h` < `max_h`).

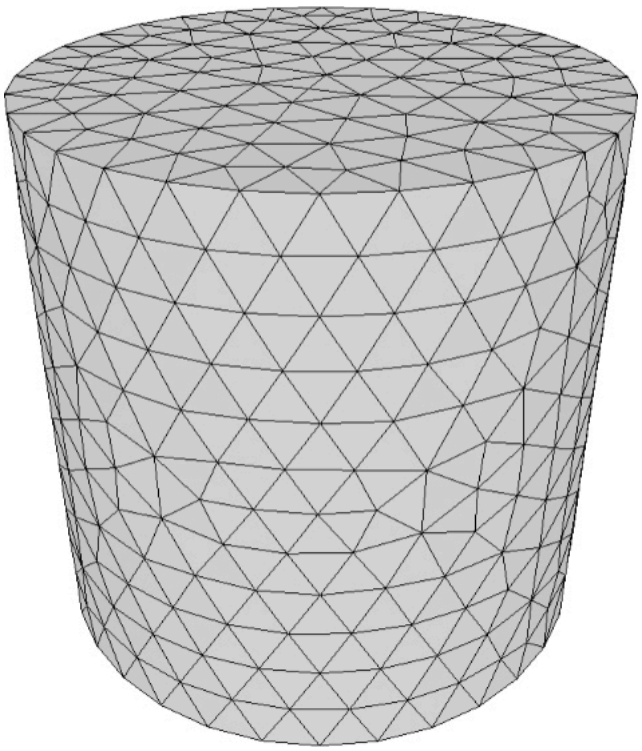


Figure 6 – Cylinder remeshed with `min_h` = 10 and `max_h` = 10.

```
Initial Mesh
  Nodes          : 256
  Triangles      : 508
  Area           : 4.711434E+04
  Qavg           : 4.163685E-02
  Qmin           : 2.128816E-02
New Mesh
  Nodes          : 554
  Triangles      : 1104
  Area           : 4.697452E+04
  Qavg           : 9.381782E-01
  Qmin           : 7.172157E-01
Max error dist.  : 2.986012E-01
```

The values `min_h` = `max_h` = 5 give the mesh shown in Figure 7.

Note that the Hausdorff distance between the initial STL mesh and the new mesh decreases as the latter gets finer.

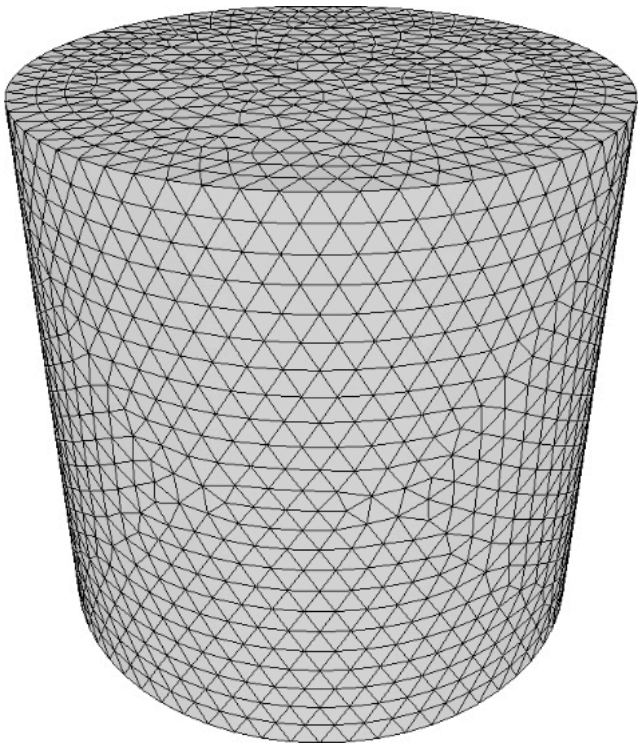


Figure 7 – Cylinder remeshed with `min_h` = 5 and `max_h` = 5.

```
Initial Mesh
  Nodes          : 256
  Triangles      : 508
  Area           : 4.711434E+04
  Qavg           : 4.163685E-02
  Qmin           : 2.128816E-02
New Mesh
  Nodes          : 2172
  Triangles      : 4340
  Area           : 4.708151E+04
  Qavg           : 9.547583E-01
  Qmin           : 7.996002E-01
Max error dist. : 6.471375E-02
```

Below are two resulting meshes for the *Part #1* example with different values for `min_h` and `max_h`:

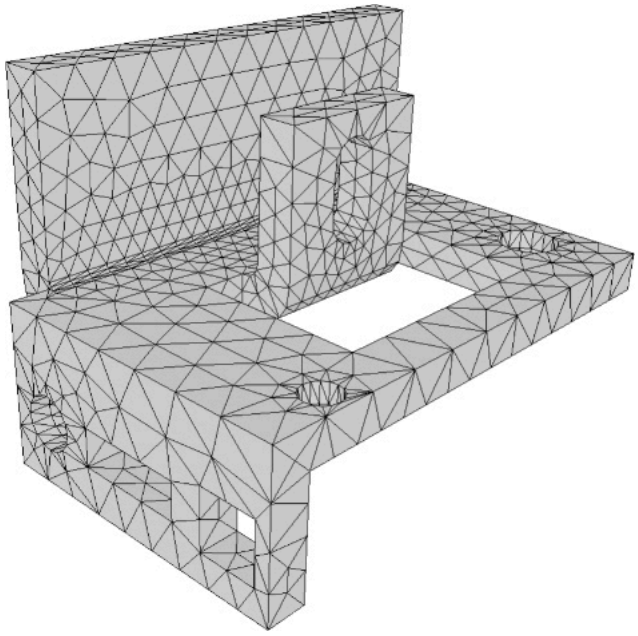


Figure 8 – Part #1 remeshed with `min_h` = 0.005 and `max_h` = DBL_MAX.

```
Initial Mesh
  Nodes          : 379
  Triangles      : 782
  Area           : 4.428174E-02
  Qavg           : 3.987199E-01
  Qmin           : 9.139058E-04
New Mesh
  Nodes          : 1290
  Triangles      : 2604
  Area           : 4.427485E-02
  Qavg           : 7.823978E-01
  Qmin           : 1.598984E-01
Max error dist.  : 4.184950E-04
```

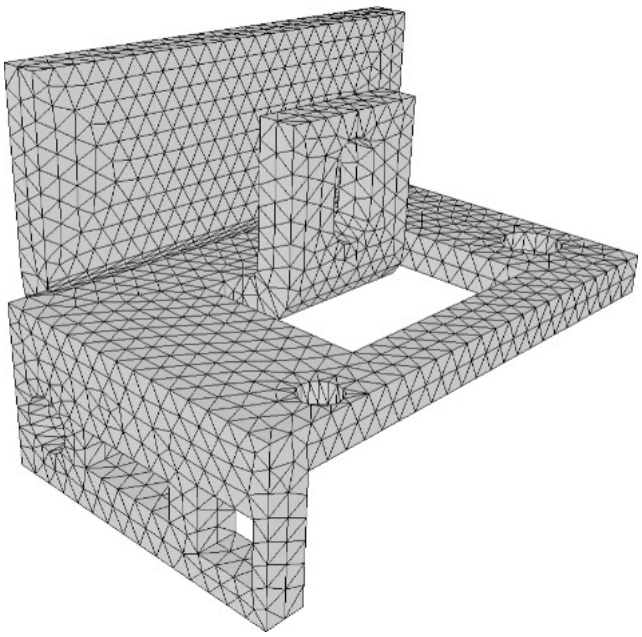


Figure 9 – Part #1 remeshed with `min_h` = 0.005 and `max_h` = 0.005.

```
Initial Mesh
  Nodes          : 379
  Triangles      : 782
  Area           : 4.428174E-02
  Qavg           : 3.987199E-01
  Qmin           : 9.139058E-04
New Mesh
  Nodes          : 2136
  Triangles      : 4296
  Area           : 4.428039E-02
  Qavg           : 8.711093E-01
  Qmin           : 2.734789E-01
Max error dist. : 2.760341E-04
```

Note that in this last example where `min_h` = `max_h` most of the domain is meshed uniformly. But due to the constraints on patches some areas such as the high-curvature fillets are remeshed finer.

3. User's specified punctual sizes

Until now, we have seen two fields in the `data_type` structure used to exchange data with the mesher:

- The `pos` matrix for the points' coordinates.
- The `connectM` matrix for the connectivity of the 3-D meshes (same matrix used for the initial mesh and for the final mesh).

We have seen also the effects of two global settings: `min_h` and `max_h`. In a more specific way, the user can set elements' size values at some nodes of the initial mesh. This is done using the `metrics` field of the `data_type` structure.

To illustrate this feature let us use again the *Part #1* example with `min_h` = 0.05 and `max_h` =0.05. We now specify a size value of 0.001 at nodes #68 and #79. At these nodes[9], these values supersede the settings bounds `min_h` and `max_h` and the new mesh follows locally these sizes.

```
#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()
{
   surfremesh_t3::mesher               the_remesher;
   surfremesh_t3::mesher::data_type    data;

   // UNLOCK THE DLL.
   surfremesh_t3::registration("Licensed to SMART Inc.", "0BEE453363E8");

   // READ THE INITIAL MESH.
   std::ifstream       istrm(“part1.dat”);
   matio::transpose_read(istrm, data.pos);
   matio::transpose_read(istrm, data.connectM);

   // REMESH THE SURFACE.
   the_remesher.settings.min_h = 0.005;
   the_remesher.settings.max_h = 0.005;
   the_remesher.settings.target_h = 0.005;
   data.metrics.resize(pos.cols(), 0.);
   data.metrics[68] = 0.001;
   data.metrics[79] = 0.001;
   the_remesher.run(data);

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

   // VISUALIZATION (OPTIONAL).
   meshtools::medit_output("out.mesh", data.pos, data.connectM, CM2_FACET3);

   return 0;
}   // main
```

The result is a new mesh with uniform size of 0.005 everywhere on the surface except near these four nodes where the elements' size drops to 0.001.

[9] The node IDs can also be found with the help of `cm2::meshtools::node_detector`. This class can find the closest node to a point or all nodes inside a box.

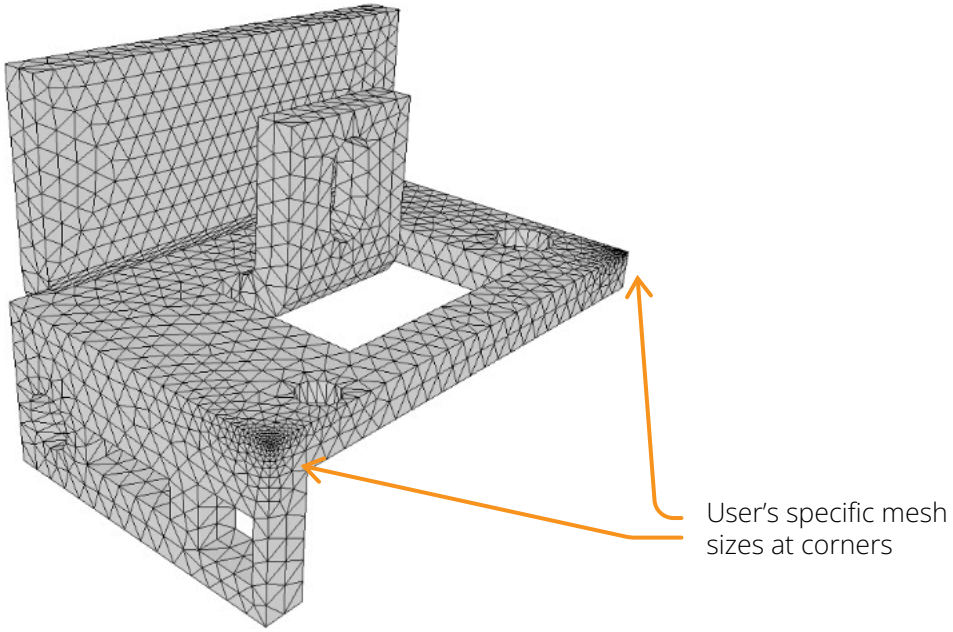


Figure 10 – Part #1 remeshed with h = 0.001 at four nodes (`min_h` = 0.005, `max_h` = 0.005) .

Specific mesh sizes can be set on any node of the initial mesh. In the following example, we set a sinus variation in `data.metrics` for all initial nodes (we used mesh at Figure 9 as initial mesh):

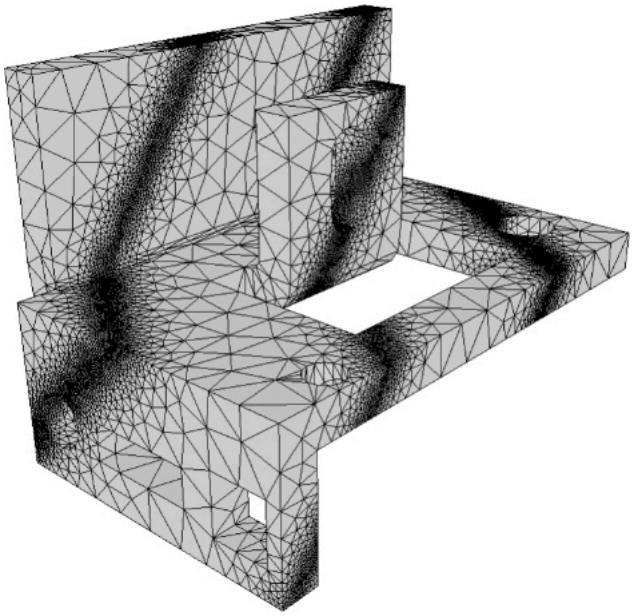


Figure 11 – Part #1 remeshed with user metrics on all initial nodes.

Hence, surface meshes can be coarsened in some areas and refined in some others.

4. Patches

The algorithm of the remeshers has three major steps. First the initial mesh is partitioned into patches/sub-domains, second the patch boundaries (called "skeleton lines" or "skeleton edges") are remeshed and finally the patches are remeshed one by one. This third step is divided into three sub steps: each patch of the initial mesh, called an *initial patch*, is unfolded to the $Z = 0$ plane, remeshed into a new patch and then mapped back to the initial surface making a *final patch*. Similarly we define *initial skeleton lines* and *final skeleton lines*.

The partition algorithm is based mostly on the angle between adjacent triangles. Starting with some element seeds, the patches are grown by neighboring traversal until the angle between two adjacent triangles is greater than a fixed parameter (`patch_angle_tolerance` in the remeshers' settings)[10] or when a user-specified edge is encountered or when the patch can no longer be unfolded within the `strain_tolerance` limit (see reference manual).

When `patch_angle_tolerance` is low the patches tend to be small, with somehow degraded elements quality but small Hausdorff distance (small chordal error). With a larger `patch_angle_tolerance` value, the patches tend to be bigger with better elements quality but larger Hausdorff distance (i.e. higher chordal error). The following example illustrates this point.

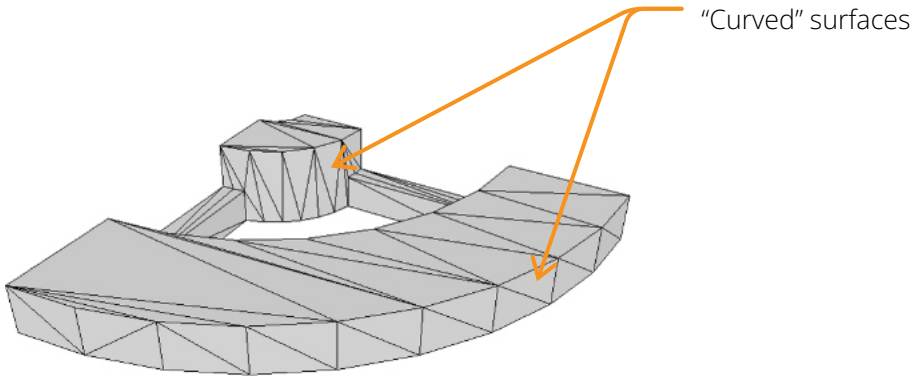


Figure 12 – Part #2 - Initial mesh.

```
Initial Mesh
  Nodes          : 86
  Triangles      : 172
  Area           : 5.352784E+04
  Qavg           : 4.778100E-01
  Qmin           : 1.452030E-02
```

[10] Other special conditions can also stop the patch growth that are beyond the scope of this manual.

We first set the value of patch_angle_tolerance to 5 degrees:

```
#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()
{
   surfremesh_t3::mesher               the_remesher;
   surfremesh_t3::mesher::data_type    data;

   // UNLOCK THE DLL.
   surfremesh_t3::registration("Licensed to SMART Inc.", "0BEE453363E8");

   // READ THE INITIAL MESH.
   std::ifstream       istrm(“part1.dat”);
   matio::transpose_read(istrm, data.pos);
   matio::transpose_read(istrm, data.connectM);

   // REMESH THE SURFACE.
   the_remesher.settings.min_h = 10.;
   the_remesher.settings.max_h = 10.;
   the_remesher.settings.target_h = 10.;
   the_remesher.settings.patch_angle_tolerance = 5.;
   the_remesher.run(data);

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

   // VISUALIZATION (OPTIONAL).
   meshtools::medit_output("out.mesh", data.pos, data.connectM, CM2_FACET3);

   return 0;
}   // main
```

The new mesh is close to the initial mesh. Many angles of the initial mesh remain in the new mesh (Figure 13).

With patch_angle_tolerance = 20, the patches are bigger. This gives a better and smoother new mesh at the cost of a higher geometric error between the two meshes (Figure 16).

With patch_angle_tolerance = 0, the initial mesh is partitioned into pure flat patches (can be limited to only one element). The chordal error between the initial mesh and the new mesh is minimum[11].

The default value patch_angle_tolerance = 20 degrees is a good compromise in most cases. A value of 45 degrees should be considered as a maximum.

[11] Note that the error distance may still be non null due to node merges and edge swaps during the final optimization step driven by final_optimization_flag and optim_tolerance. To reduce the geometric error to zero set patch_angle_tolerance = optim_tolerance = 0.

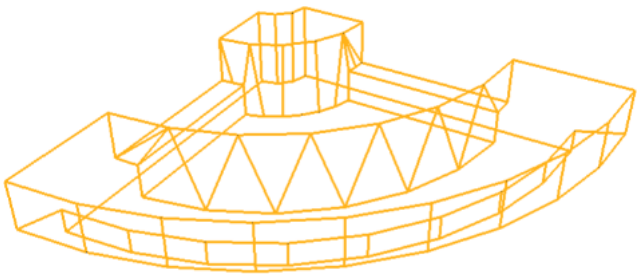


Figure 13 – Part #2 - Skeleton lines in the initial mesh (`patch_angle_tolerance` = 5).

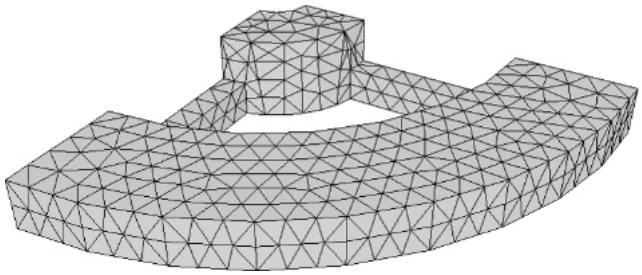


Figure 14 – Part #2 remeshed with `min_h` = 10, `max_h` = 10 and `patch_angle_tolerance` = 5.

```
New Mesh
  Nodes          : 659
  Triangles      : 1318
  Area           : 5.352579E+04
  Qavg           : 8.621966E-01
  Qmin           : 2.777585E-01
Max error dist.  : 1.912226E-01
```

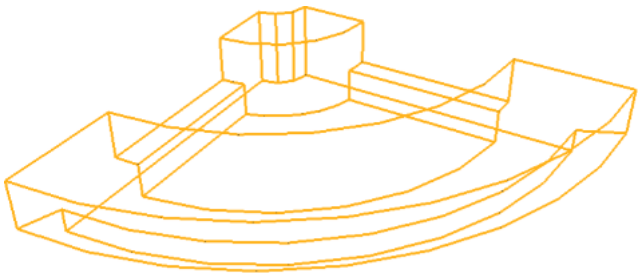


Figure 15 – Part #2 - Skeleton lines in the initial mesh (patch_angle_tolerance = 20).

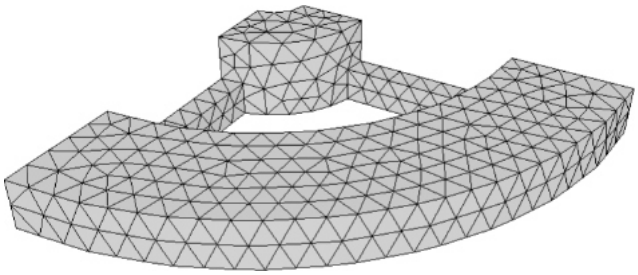


Figure 16 – Part #2 remeshed with min_h = 10, max_h = 10 and patch_angle_tolerance = 20.

```
New Mesh
  Nodes          : 653
  Triangles      : 1306
  Area           : 5.356828E+04
  Qavg           : 8.813238E-01
  Qmin           : 6.071915E-01
Max error dist.  : 3.821892E-01
```

☞ Usually, the higher the angle tolerance, the better the mesh, the smaller the number of elements but the higher the Hausdorff distance between initial and remeshed surfaces.

5. Remeshing with quadrangles (CM2 SurfRemesh Q4)

CM2 SurfRemesh Q4 can be used in place of **CM2 SurfRemesh T3** to remesh with quadrangles instead of triangles. From the previous example, simply changing the namespace changes the type of the remesher:

```
#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()
{
   surfremesh_q4::mesher               the_remesher;
   surfremesh_q4::mesher::data_type    data;

   // UNLOCK THE DLL.
   surfremesh_q4::registration("Licensed to SMART Inc.", "0BEE453363E8");

   // READ THE INITIAL MESH.
   std::ifstream       istrm(“part1.dat”);
   matio::transpose_read(istrm, data.pos);
   matio::transpose_read(istrm, data.connectM);

   // REMESH THE SURFACE.
   the_remesher.settings.min_h = 10.;
   the_remesher.settings.max_h = 10.;
   the_remesher.settings.target_h = 10.;
   the_remesher.run(data);

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

   // VISUALIZATION (OPTIONAL).
   meshtools::medit_output("out.mesh", data.pos, data.connectM, CM2_FACE_MIX);

   return 0;
}   // main
```

The default settings of this remesher is the *quad-dominant* mode: it generates mixed quadrangle-triangle meshes. This is why the output file is performed here with the type `CM2_FACE_MIX`.

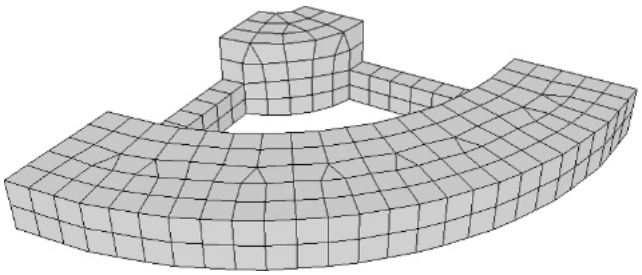


Figure 17 – Part #2 remeshed in quad-dominant mode
(`min_h` = 10, `max_h` = 10, `patch_angle_tolerance` = 20).

```
New Mesh
  Nodes          : 550
  Elements       : 564
    Quadrangles  : 536 (95.04 %, 97.92 %)
    Triangles    : 28 (4.96 %, 2.08 %)
  Area           : 5.356221E+04
  Qavg           : 9.109910E-01
  Qmin           : 5.229764E-01
Max error dist.  : 3.824302E-01
```

An all-quad mesh can be generated by changing a flag in the remesher's settings:

```
int main()
{
   surfremesh_q4::mesher               the_remesher;
   surfremesh_q4::mesher::data_type    data;

   // UNLOCK THE DLL.
   surfremesh_q4::registration("Licensed to SMART Inc.", "0BEE453363E8");

   // READ THE INITIAL MESH.
   std::ifstream       istrm(“part1.dat”);
   matio::transpose_read(istrm, data.pos);
   matio::transpose_read(istrm, data.connectM);

   // REMESH THE SURFACE.
   the_remesher.settings.min_h = 10.;
   the_remesher.settings.max_h = 10.;
   the_remesher.settings.target_h = 10.;
   the_remesher.settings.all_quad_flag = true;
   the_remesher.run(data);

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

   // VISUALIZATION (OPTIONAL).
   meshtools::medit_output("out.mesh", data.pos, data.connectM, CM2_FACEQ4);

   return 0;
}   // main
```

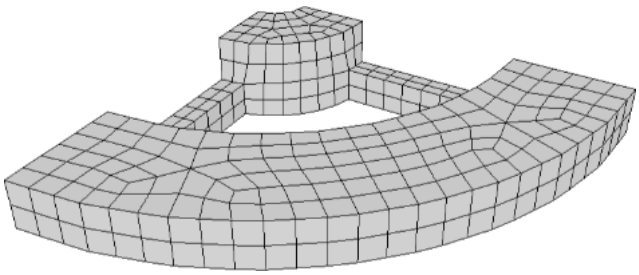


Figure 18 – Part #2 remeshed in all-quad mode
(`min_h` = 10, `max_h` = 10, `patch_angle_tolerance` = 20).

```
New Mesh
  Nodes          : 706
  Elements       : 706
    Quadrangles  : 706 (100.00 %, 100.00 %)
    Triangles    : 0 (0.00 %, 0.00 %)
  Area           : 5.357723E+04
  Qavg           : 8.113556E-01
  Qmin           : 4.757265E-01
Max error dist.  : 3.326376E-01
```

☞ If triangles are tolerated, the quad-dominant mode (the default settings) should be preferred against the all-quad mode. It gives usually better meshes with fewer elements.
The all-quad mode should be restricted to very simple surfaces such as this one.

6. Keeping the skeleton lines

Let us now consider the remeshing of a sphere (Figure 19):

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

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


int main()
{
   const double              R(10.);
   const unsigned            N(64);
   UIntVec                   indices;
   UIntMat                   connectE;
   DoubleMat                 pos;
   UIntMat                   connectMi, connectMr;

   // UNLOCK THE DLL.
   surfremesh_t3::registration("Licensed to SMART Inc.", "0BEE453363E8");

   // THE INITIAL MESH.
   pos.push_back(DoubleVec3(0., 0., R));
   meshtools1d::extrude_rotate(pos, 0, DoubleVec3(0.), DoubleVec3(0., M_PI, 0.), N/2, indices);
   meshtools1d::indices_to_connectE2(indices, connectE);
   meshtools2d::extrude_rotate_T3(pos, connectE, DoubleVec3(0.),
                                  DoubleVec3(0., 0., 2.*M_PI), N, 1, connectMi);

   // REMESH THE SURFACE.
   surfremesh_t3::mesher                  the_remesher;
   surfremesh_t3::mesher::data_type       data;
   data.pos = pos;
   data.connectM.copy(connectMi);       // Hard copy of connectMi
   the_remesher.settings.min_h = 1.;
   the_remesher.settings.max_h = 1.;
   the_remesher.settings.target_h = 1.;
   the_remesher.run(data);
   pos = data.pos;
   connectMr = data.connectM;

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

   // VISUALIZATION (OPTIONAL).
   meshtools::medit_output("in.mesh", pos, connectMi, CM2_FACET3);
   meshtools::medit_output("out.mesh", pos, connectMr, CM2_FACET3);

   return 0;
}   // main
```

To generate the initial mesh we start with meshing a "Greenwich" meridian[12]:

```
pos.push_back(DoubleVec3(0., 0., R));
meshtools1d::extrude_rotate(pos, 0, DoubleVec3(0.), DoubleVec3(0., M_PI, 0.), N/2, indices);
meshtools1d::indices_to_connectE2(indices, connectE);
```

[12] For explanation on `cm2::meshtools1d::extrude_rotate`, `cm2::meshtools1d::indices_to_connectE2` and `cm2::meshtools2d::extrude_rotate_T3`, please refer to the HTML reference manual of the **CM2 MeshTools®** SDK.

This arc is then extruded into a triangle surface with a 2 π rotation:

```
meshtools2d::extrude_rotate_T3(pos, connectE, DoubleVec3(0.),
                               DoubleVec3(0., 0., 2.*M_PI), N, 1, connectMi);
```

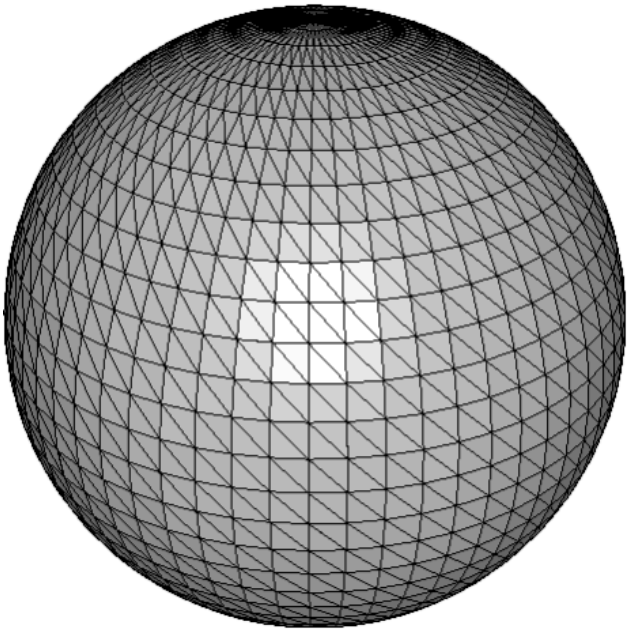


Figure 19 – Sphere. Initial mesh.

Note that this way of generating the mesh leads to multiple coincident nodes (along the initial meridian) and degenerated triangles at the poles. Fortunately, the remeshers are able to fix these issues and the new meshes are perfectly closed (watertight) with no degenerated element (due to the initial cleaning step).

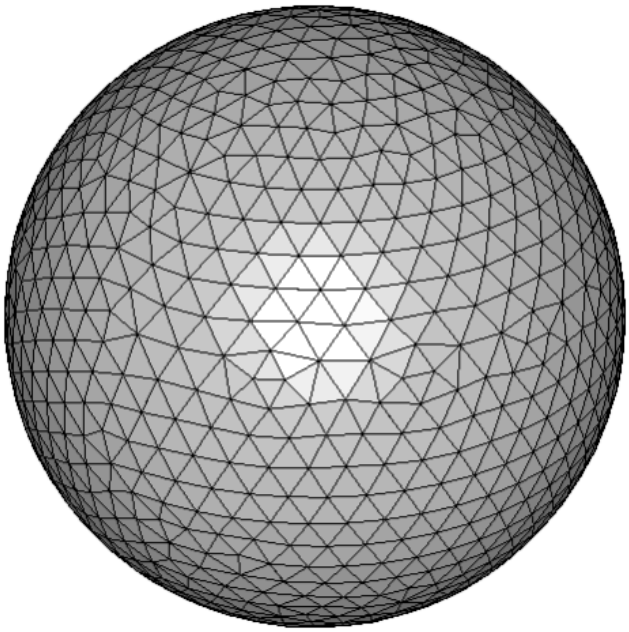


Figure 20 – Sphere remeshed with triangles.

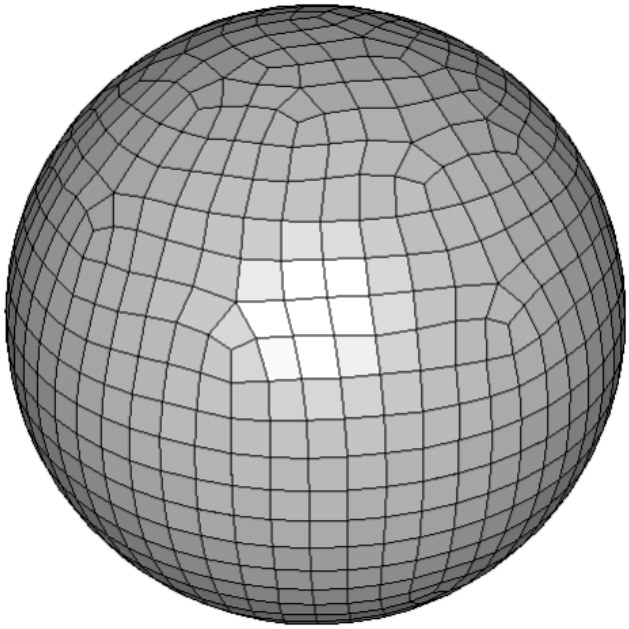


Figure 21 – Sphere remeshed with quadrangles only (all-quad mode).

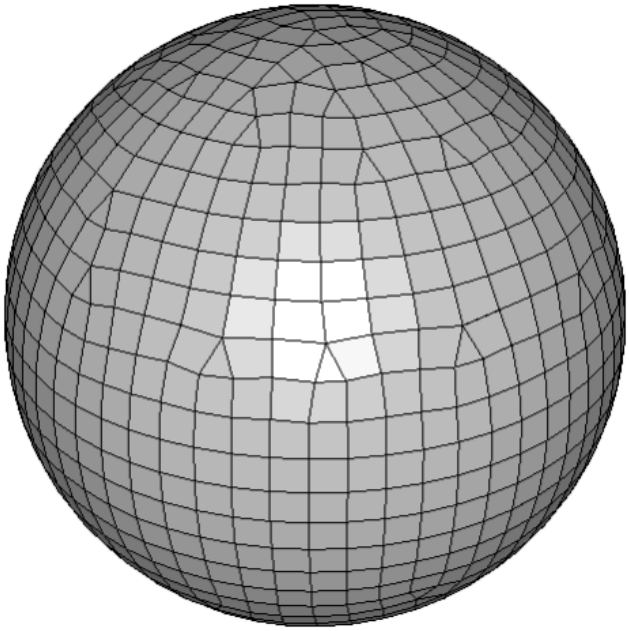


Figure 22 – Sphere remeshed with quads and triangles (mixed mode).

It appears that the remeshers divide the initial sphere into two patches (Figure 23) and that the initial G meridian and the equator lines are lost in the remeshing process. Should we be interested in keeping them, we have to put the edges of these lines into the edge connectivity matrix `skeleton_edges_in` as shown in the following example[13].

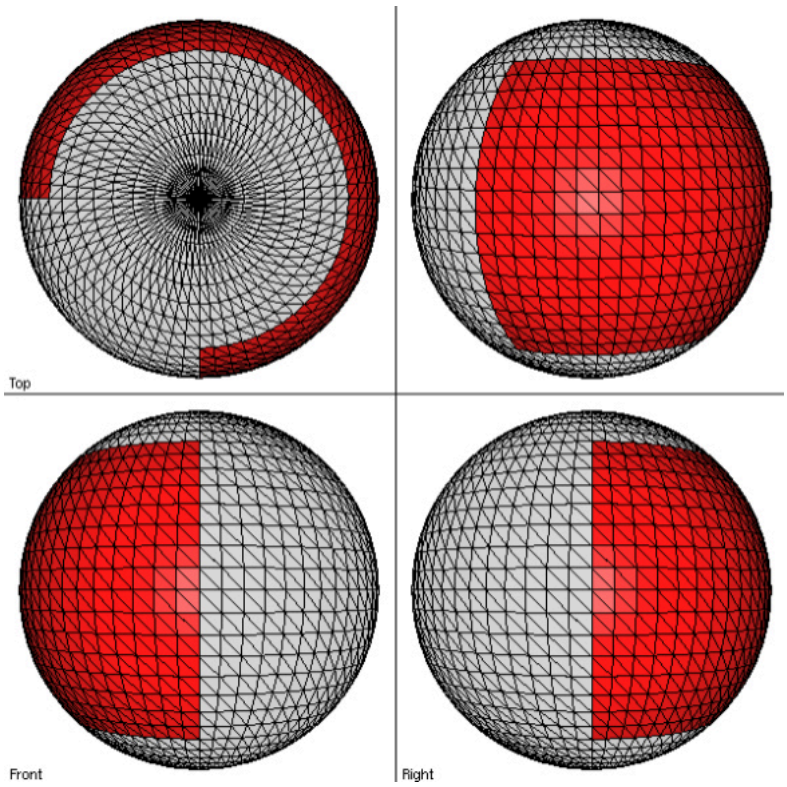


Figure 23 – Patches on the initial sphere (two patches).

[13] The meridian edges are readily available. However the equator edges need to be searched for (the code to do that is not shown here).

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

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

int main()
{
   const double              R(10.);
   const unsigned            N(64);
   UIntVec                   indices;
   UIntMat                   connectE;
   DoubleMat                 pos;
   UIntMat                   connectMi;

   // UNLOCK THE DLL.
   surfremesh_t3::registration("Licensed to SMART Inc.", "0BEE453363E8");

   // THE INITIAL MESH.
   pos.push_back(DoubleVec3(0., 0., R));
   meshtools1d::extrude_rotate(pos, 0, DoubleVec3(0.), DoubleVec3(0., M_PI, 0.), N/2, indices);
   meshtools1d::indices_to_connectE2(indices, connectE);
   meshtools2d::extrude_rotate_T3(pos, connectE, DoubleVec3(0.),
                                  DoubleVec3(0., 0., 2.*M_PI), N, 1, connectMi);

   // REMESHING THE SURFACE.
   surfremesh_t3::mesher                  the_remesher;
   surfremesh_t3::mesher::data_type       data;
   data.pos = pos;
   data.connectM.copy(connectMi);        // Hard copy of connectMi
   data.skeleton_edges_in = connectE;    // Keep the meridian line.
   the_remesher.settings.min_h = 1.;
   the_remesher.settings.max_h = the_remesher.settings.target_h = 1.;
   the_remesher.run(data);
   pos = data.pos;
      // SOME OUTPUT INFO (OPTIONAL).
   data.print_info(&display_hdl);

   // VISUALIZATION (OPTIONAL).
   meshtools::medit_output("out.mesh", pos, data.connectM, CM2_FACET3);

   return 0;
}   // main
```

Now the remesher divides the initial sphere into four patches in order to keep the constrained line.

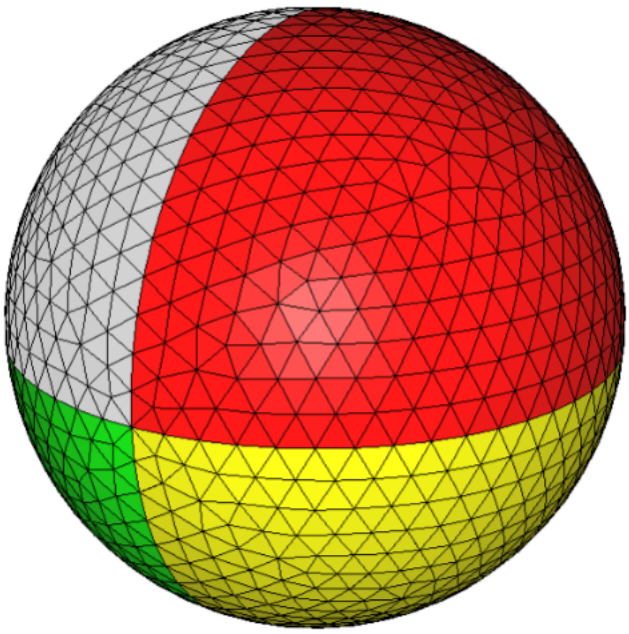


Figure 24 – Patches on the remeshed sphere when G meridian is kept (four patches).

Here the G meridian line (and by chance the equator line also) is kept in the final mesh. However it is remeshed: the nodes and edges along it are different now. The geometry of the line is preserved but not its mesh.

To keep the initial edges, we shall use the `hard_edges` field instead of `skeleton_edges_in`:

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

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


int main()
{
   const double              R(10.);
   const unsigned            N(64);
   UIntVec                   indices;
   UIntMat                   connectE;
   DoubleMat                 pos;
   UIntMat                   connectMi, connectMr;

   // UNLOCK THE DLL.
   surfremesh_t3::registration("Licensed to SMART Inc.", "0BEE453363E8");

   // THE INITIAL MESH.
   pos.push_back(DoubleVec3(0., 0., R));
   meshtools1d::extrude_rotate(pos, 0, DoubleVec3(0.),
                               DoubleVec3(0., M_PI, 0.), N/2, indices);
   meshtools1d::indices_to_connectE2(indices, connectE);
   meshtools2d::extrude_rotate_T3(pos, connectE, DoubleVec3(0.),
                                  DoubleVec3(0., 0., 2.*M_PI), N, 1,
                                  connectMi);

   // REMESHING THE SURFACE.
   surfremesh_t3::mesher                  the_remesher;
   surfremesh_t3::mesher::data_type       data;
   data.pos = pos;
   data.connectM.copy(connectMi);     // Hard copy of connectMi
   data.hard_edges = connectE;        // Keep the initial edges along the meridian.
   the_remesher.settings.min_h = 1.;
   the_remesher.settings.max_h = 1.;
   the_remesher.run(data);
   pos = data.pos;
   connectMr = data.connectM;

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

   // VISUALIZATION (OPTIONAL).
   meshtools::medit_output("out.mesh", pos, connectMr, CM2_FACET3);

   return 0;
}   // main
```

COMPUTING OBJECTS

CM2 SurfRemesh® T3/Q4

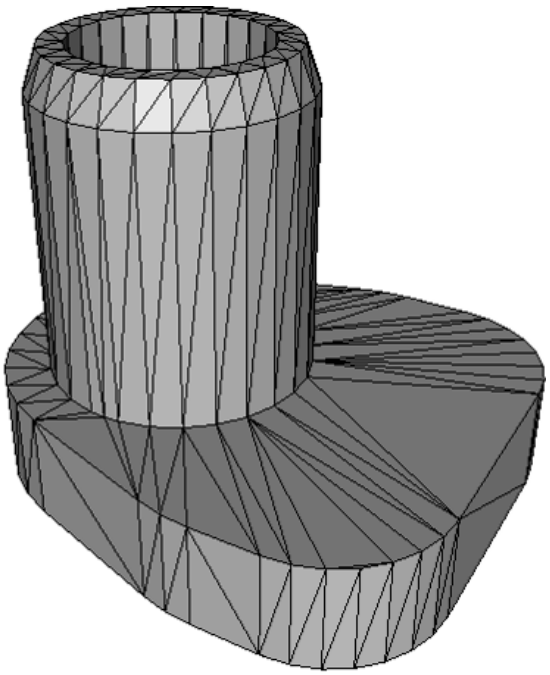
Version 5.6

Mesh gallery

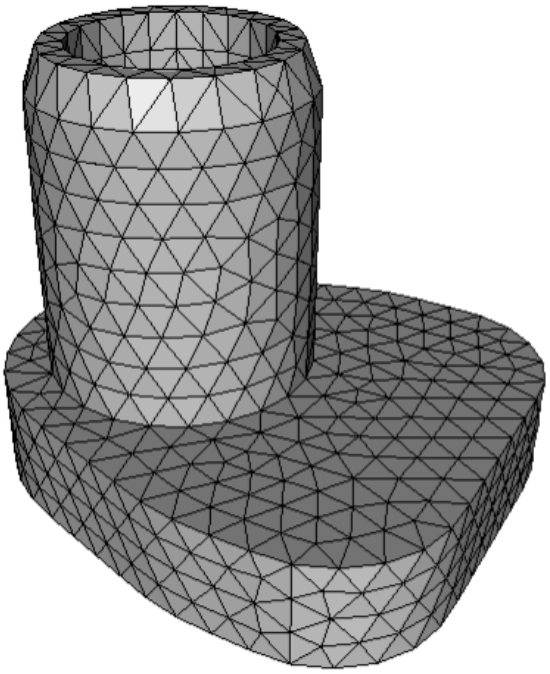
Revision February 2025.

https://www.computing-objects.com

Example 1

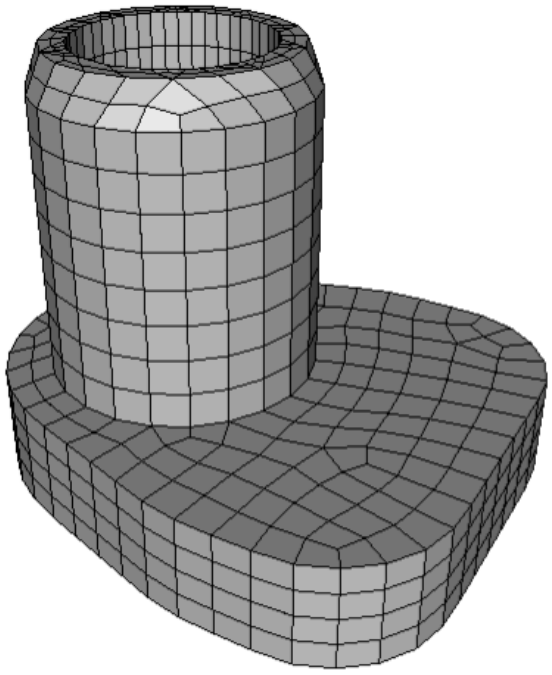


```
Initial mesh

Nodes            : 202
Triangles        : 400
Area             : 1.772E+05
Qavg             : 4.058E-01
Qmin             : 6.857E-02
```

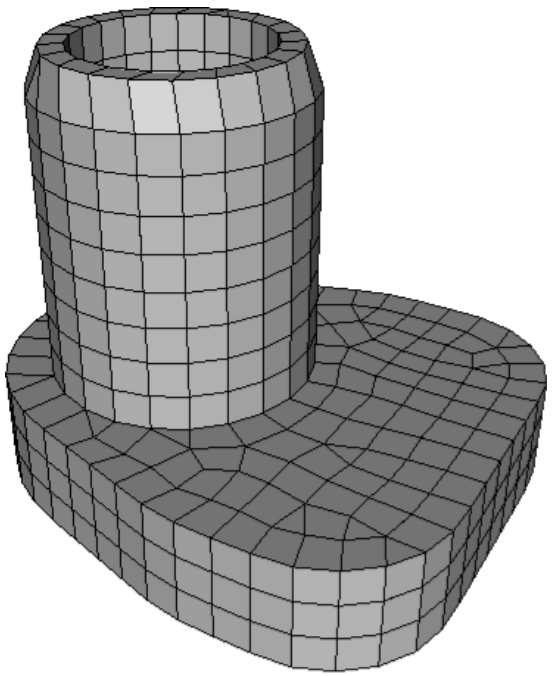


```
New mesh CM2 SurfRemesh T3

New Mesh
  Nodes          : 939
  Triangles      : 1874
  Area           : 1.765980E+05
  Qavg           : 9.110418E-01
  Qmin           : 5.289499E-01
Max error dist.  : 1.888201E+00
Cleaning time    : 0.00 s.
Partition time   : 0.02 s.
Remesh time      : 0.01 s.
Optim time       : 0.00 s.
Total time       : 0.03 s. (60451.34 t/s.)
```

The output information given here are only indicative. All runs were done with x64 CM2 libs (Visual Studio 2010 MD build) on Windows 8.1 x64 with Intel Xeon E3-1270 V2 3.5 Ghz (8 threads, turbo boost disabled).



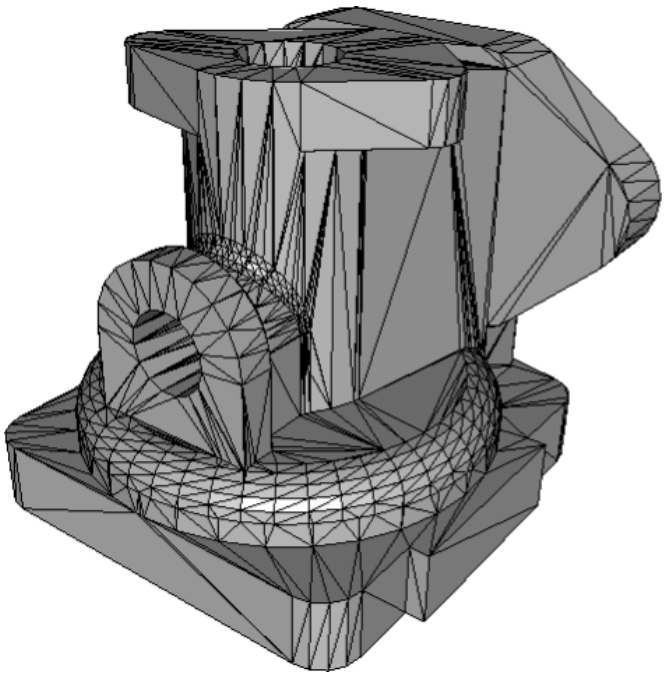
```
New mesh CM2 SurfRemesh Q4 (all-quad)

  Nodes          : 886
  Elements       : 884
    Quadrangles  : 884 (100.00 %, 100.00 %)
    Triangles    : 0 (0.00 %, 0.00 %)
  Area           : 1.764462E+05
  Qavg           : 8.459630E-01
  Qmin           : 2.016499E-01
Max error dist.  : 1.857904E+00
Cleaning time    : 0.00 s.
Partition time   : 0.01 s.
Remesh time      : 0.03 s.
Optim time       : 0.00 s.
Total time       : 0.05 s. (19217.39 e/s.)
```



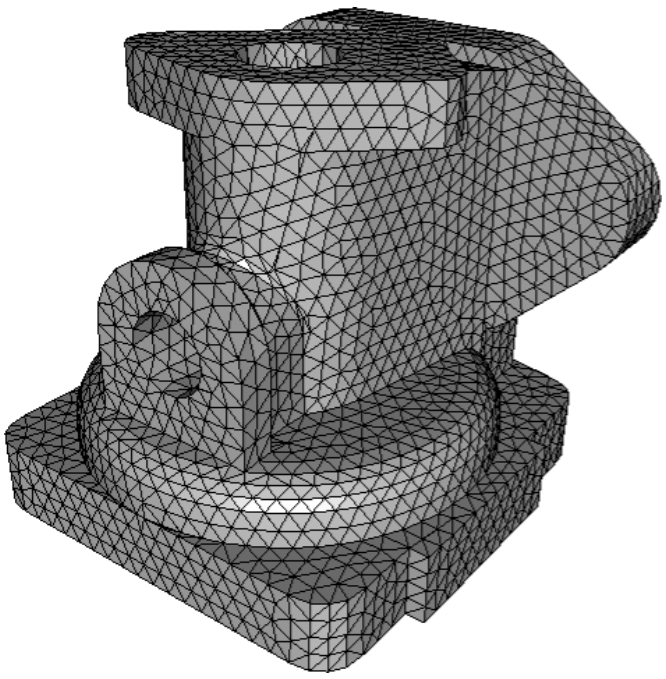
```
New mesh CM2 SurfRemesh Q4 (quad-dominan)

  Nodes          : 811
  Elements       : 821
    Quadrangles  : 797 (97.08 %, 98.81 %)
    Triangles    : 24 (2.92 %, 1.19 %)
  Area           : 1.765048E+05
  Qavg           : 9.035228E-01
  Qmin           : 3.706508E-01
Max error dist.  : 2.168070E+00
Cleaning time    : 0.00 s.
Partition time   : 0.02 s.
Remesh time      : 0.02 s.
Optim time       : 0.00 s.
Total time       : 0.04 s. (22189.08 e/s.)
```

The percent figures indicate the ratio between the number of quads and triangles and the ratio between the areas of quads and triangles.

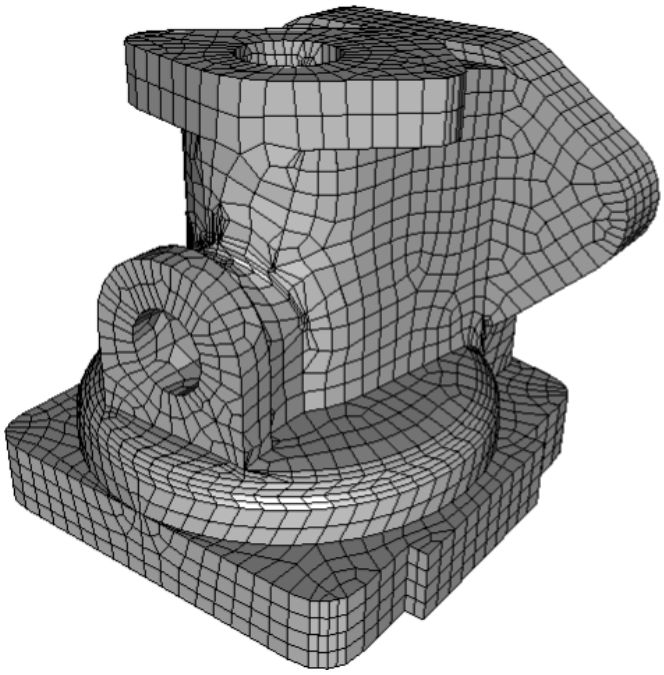
Example 2



```
Initial mesh

Nodes          : 1152
Triangles      : 2316
Area           : 8.778E+04
Qavg           : 3.764E-01
Qmin           : 3.233E-04
```

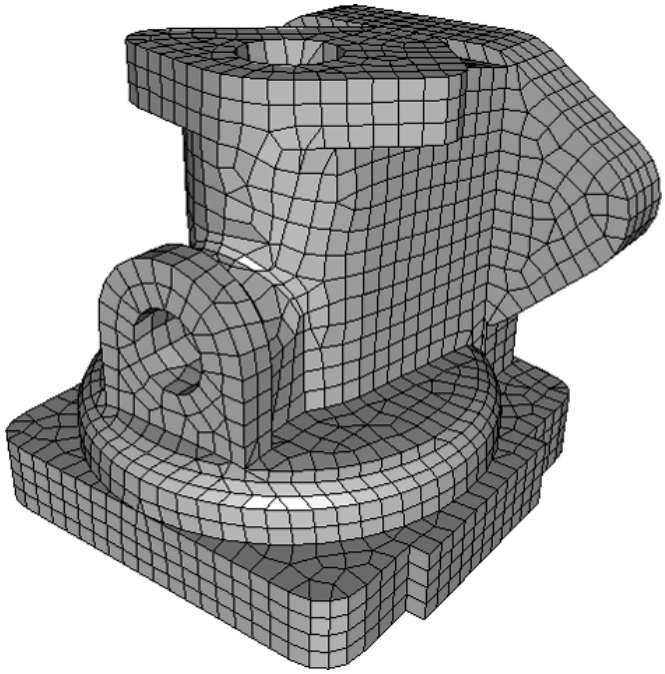


```
New mesh CM2 SurfRemesh T3

  Nodes          : 4181
  Triangles      : 8374
  Area           : 8.759187E+04
  Qavg           : 8.737813E-01
  Qmin           : 3.312088E-02
Max error dist.  : 5.498549E-01
Cleaning time    : 0.01 s.
Partition time   : 0.12 s.
Remesh time      : 0.04 s.
Optim time       : 0.03 s.
Total time       : 0.19 s. (43388.58 t/s.)
```



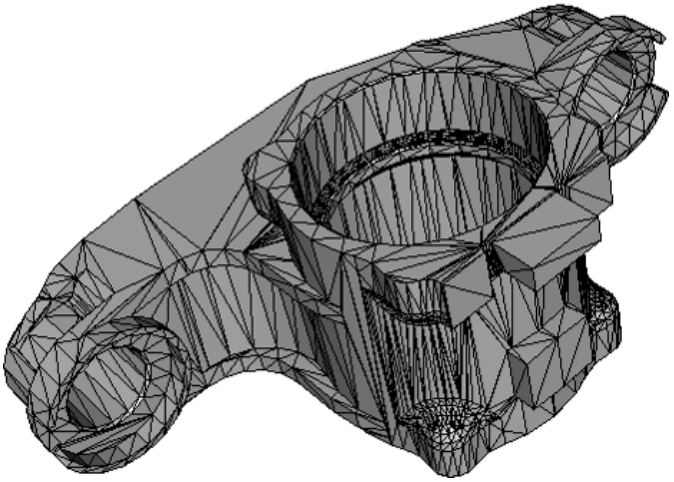
```
New mesh CM2 SurfRemesh Q4 (all-quad)

  Nodes          : 4891
  Elements       : 4897
    Quadrangles  : 4897 (100.00 %, 100.00 %)
    Triangles    : 0 (0.00 %, 0.00 %)
  Area           : 8.761525E+04
  Qavg           : 6.821883E-01
  Qmin           : 3.784221E-03
Max error dist.  : 1.267006E+00
Cleaning time    : 0.01 s.
Partition time   : 0.11 s.
Remesh time      : 0.16 s.
Optim time       : 0.02 s.
Total time       : 0.30 s. (16269.11 e/s.)
```

Some quads have a very bad shape quality: rather use the quad-dominant mode.

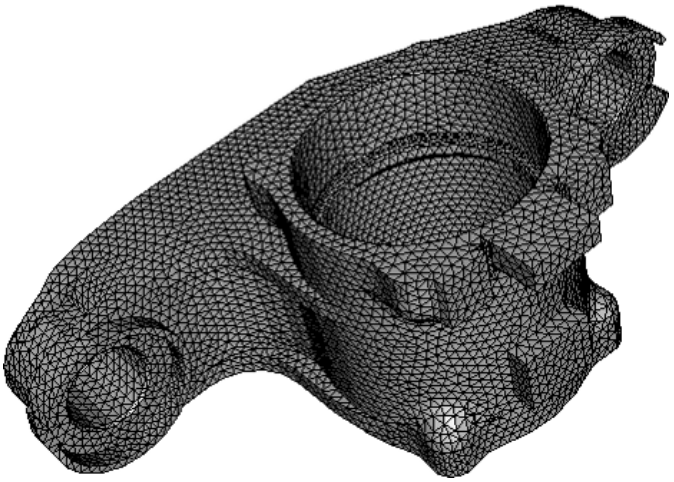


```
New mesh CM2 SurfRemesh Q4 (quad-dominant)

  Nodes          : 3907
  Elements       : 4009
    Quadrangles  : 3817 (95.21 %, 98.23 %)
    Triangles    : 192 (4.79 %, 1.77 %)
  Area           : 8.760907E+04
  Qavg           : 8.149696E-01
  Qmin           : 1.252741E-02
Max error dist.  : 6.558714E-01
Cleaning time    : 0.01 s.
Partition time   : 0.11 s.
Remesh time      : 0.06 s.
Optim time       : 0.02 s.
Total time       : 0.21 s. (19461.16 e/s.)
```

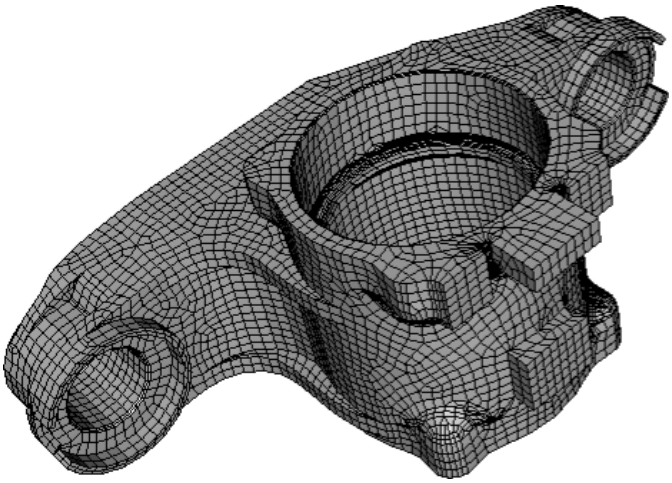
Example 3



```
Initial mesh

Nodes          : 2674
Triangles      : 5364
Area           : 8.922E+05
Qavg           : 3.667E-01
Qmin           : 2.070E-04
```

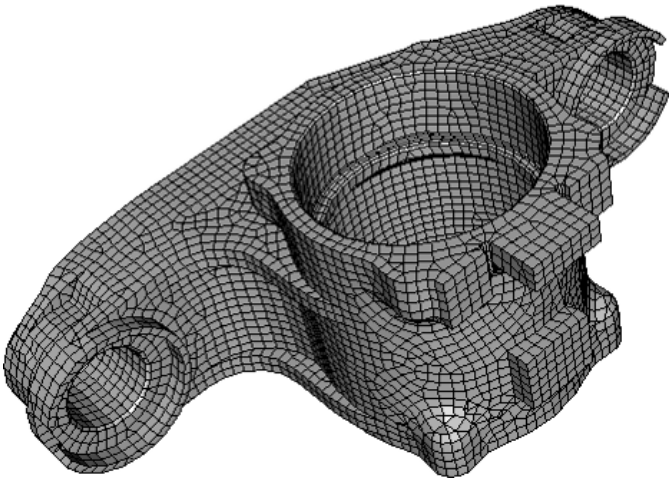


```
New mesh CM2 SurfRemesh T3

  Nodes          : 11153
  Triangles      : 22322
  Area           : 8.914079E+05
  Qavg           : 8.573457E-01
  Qmin           : 3.250150E-02
Max error dist.  : 9.770317E-01
Cleaning time    : 0.01 s.
Partition time   : 0.26 s.
Remesh time      : 0.21 s.
Optim time       : 0.06 s.
Total time       : 0.56 s. (40219.82 t/s.)
```



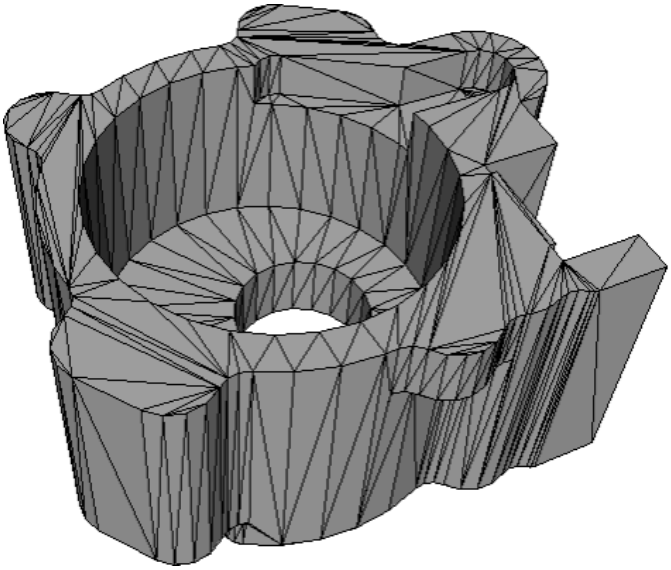
```
New mesh CM2 SurfRemesh Q4 (all-quad)

  Nodes          : 12970
  Elements       : 12978
    Quadrangles  : 12978 (100.00 %, 100.00 %)
    Triangles    : 0 (0.00 %, 0.00 %)
  Area           : 8.916372E+05
  Qavg           : 6.891853E-01
  Qmin           : 2.739883E-03
Max error dist.  : 1.237764E+00
Cleaning time    : 0.01 s.
Partition time   : 0.25 s.
Remesh time      : 0.56 s.
Optim time       : 0.06 s.
Total time       : 0.91 s. (14245.88 e/s.)
```

Some quads have a very bad shape quality: rather use the quad-dominant mode.

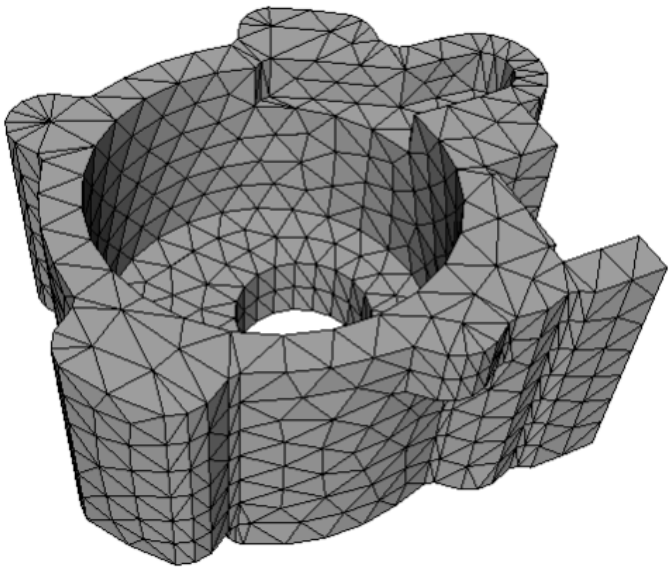


```
New mesh CM2 SurfRemesh Q4 (quad-dominant)

  Nodes          : 10089
  Elements       : 10277
    Quadrangles  : 9917 (96.50 %, 98.90 %)
    Triangles    : 360 (3.50 %, 1.10 %)
  Area           : 8.914259E+05
  Qavg           : 8.330071E-01
  Qmin           : 3.436851E-02
Max error dist.  : 1.038236E+00
Cleaning time    : 0.01 s.
Partition time   : 0.25 s.
Remesh time      : 0.14 s.
Optim time       : 0.05 s.
Total time       : 0.47 s. (21819.54 e/s.)
```

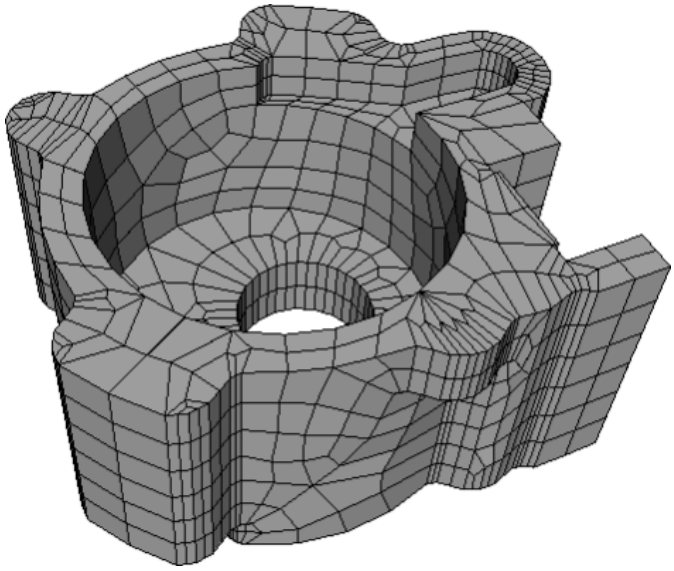
Example 4



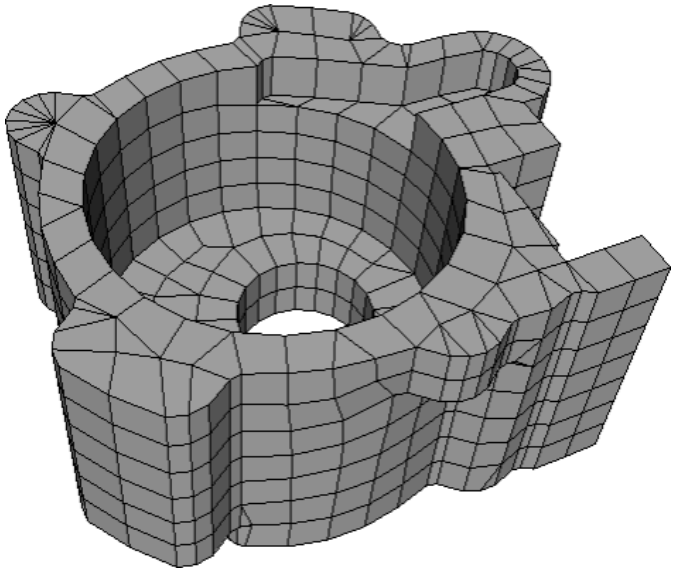
```
Initial mesh

Nodes          : 406
Triangles      : 812
Area           : 6.876E+04
Qavg           : 3.052E-01
Qmin           : 2.591E-03
```



```
New mesh CM2 SurfRemesh T3

  Nodes          : 972
  Triangles      : 1944
  Area           : 6.860757E+04
  Qavg           : 7.690156E-01
  Qmin           : 1.499089E-01
Max error dist.  : 1.335935E+00
Cleaning time    : 0.00 s.
Partition time   : 0.02 s.
Remesh time      : 0.02 s.
Optim time       : 0.00 s.
Total time       : 0.04 s. (48600.05 t/s.)
```



```
New mesh CM2 SurfRemesh Q4 (all-quad)

  Nodes          : 1511
  Elements       : 1511
    Quadrangles  : 1511 (100.00 %, 100.00 %)
    Triangles    : 0 (0.00 %, 0.00 %)
  Area           : 6.860654E+04
  Qavg           : 5.668175E-01
  Qmin           : 3.951873E-02
Max error dist.  : 1.500103E+00
Cleaning time    : 0.00 s.
Partition time   : 0.02 s.
Remesh time      : 0.08 s.
Optim time       : 0.00 s.
Total time       : 0.10 s. (15109.98 e/s.)
```



```
New mesh CM2 SurfRemesh Q4 (quad-dominant)

  Nodes          : 926
  Elements       : 944
    Quadrangles  : 908 (96.19 %, 99.06 %)
    Triangles    : 36 (3.81 %, 0.94 %)
  Area           : 6.858510E+04
  Qavg           : 7.651056E-01
  Qmin           : 1.464406E-01
Max error dist.  : 1.683082E+00
Cleaning time    : 0.00 s.
Partition time   : 0.03 s.
Remesh time      : 0.04 s.
Optim time       : 0.00 s.
Total time       : 0.07 s. (13882.34 e/s.)
```

COMPUTING
OBJECTS