COMPUTING OBJECTS

CM2 SurfRemesh® T3/Q4

Version 5.6

reference manual

Revision February 2025.

https://www.computing-objects.com

Forewords

This manual is a reference manual for the structures `data_type` and `settings_type` of the surface mesh generators **CM2 SurfRemesh® T3/Q4** of the **CM2 MeshTools®** SDK.

It is based on the HTML reference manual (Doxygen© generated).

For examples, please refer to the manual **CM2 SurfRemesh T3/Q4 - tutorials**.

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

Table of contents

1. Data of the remeshers

The whole data set for a run of the remeshers is gathered into a single structure of type `data_type`:

```
void surfremesh_t3::mesher::run (surfremesh_t3::mesher::data_type& data);
void surfremesh_q4::mesher::run (surfremesh_q4::mesher::data_type& data);
```

The `data_type` structure contains the following fields:

Coordinates of the points

Matrix `pos` of dimensions 3 x `total_nods` (IN/OUT).

It stores the coordinates of all points. The coordinates are stored column-wise. The column index is the index of the node (zero-based, i.e. from 0 to N-1). The *X*-coordinates are in the first row, the *Y*-coordinates in the second row and the *Z*-coordinates in the third row.

Upon exit, the coordinates of the newly generated nodes are appended to the back of the matrix as new columns. The initial columns are left unchanged[1].

Elements

Matrix `connectM` (IN/OUT) of the connectivity of the elements (column oriented).

For the triangle remesher, the dimensions of this matrix are 3 x `nefs_in` (input) and 3 x `nefs` (output).

For the quadrangle remesher, the dimensions are 3 x `nefs_in` upon entry (only triangle meshes upon entry[2]) and 4 x `nefs` upon exit. In this case, the leading part of the matrix (from columns 0 to `nefs_Q4` - 1) is the connectivity of the quadrangle elements. The tailing part (from columns `nefs_Q4` to `nefs` - 1) is the connectivity of the triangle elements, the fourth node ID in this part being `CM2_NONE` (i.e. unsigned(-1)).

The ordering of the elements in this matrix is irrelevant.

The initial connectivity is lost and overwritten by the new mesh connectivity, except when the remesher fails.

Metric map

Vector `metrics` (IN).

The user can specify a target mesh size at each initial node. If the value for a node is zero or negative or not present a default value will be used instead[3]. The new mesh is generated to fit best locally the metric values.

Only the values on the nodes of the initial mesh are considered.

[1] To keep only the nodes of the final mesh, use function `cm2::meshtools::simplify` after the remeshing:
`cm2::meshtools::simplify(data.pos, data.connectM);`

[2] To remesh an all-quad or a quad-dominant mesh, use `cm2::meshtools2d::split_Q4_into_T3` to transform into an all-triangle mesh prior to remeshing.

[3] The default mesh size value at a hard node is the minimum distance to any other hard node, caped with `min_h` and `max_h`.

Neighbors

Matrix `neighbors` (OUT) of the neighbor elements of dimensions 3 x `nefs` (T3) or 4 x `nefs` (Q4).

This matrix gives, for each element in the final mesh, the indices of the three or four neighboring elements (`CM2_NONE` if none or several).

`neighbors(i, j)` is the neighbor of the j^{th} element sharing its i^{th} edge. See Figure 1 for the local numbering of the faces.

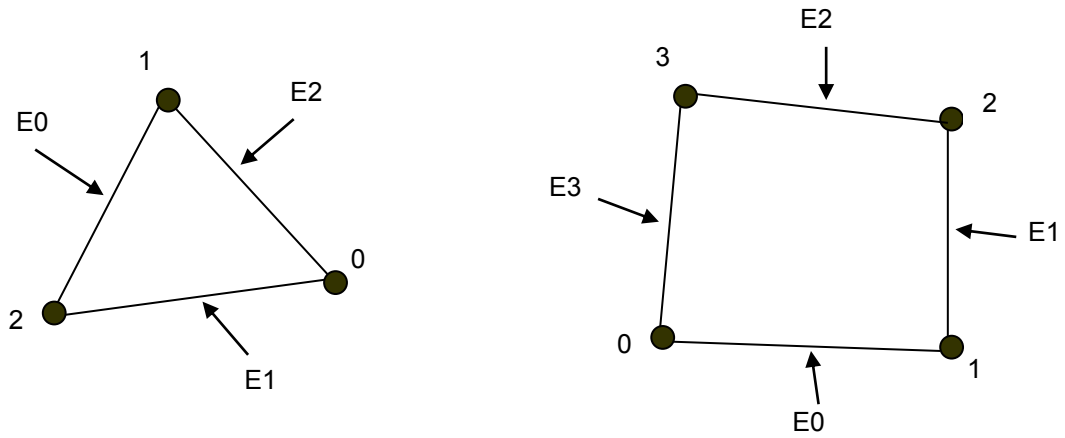

Figure 1 – Nodes and edges local numbering in triangles and quads.

Ancestors

`ancestors` is a vector of size `total_nods`, i.e. the number of columns in the `pos` matrix upon exit (OUT).

This vector gives, for each node, the index of one of the elements in which it belongs (`CM2_NONE` if the node is not referenced in the final connectivity matrix `connectM`).

Together with the `neighbors` matrix, this can make easy the design of search and traversal algorithms (such as looking for all elements connected to a node).

Shape qualities

Vector `shape_qualities` is a vector of size `nefs` (OUT).

This vector gives the shape quality of each element.

The formula for the shape quality of a triangle writes:

$$Q_s = 4\sqrt{3}\,\frac{S}{L_{\max}\,P}$$

with:

- S Area of the triangle.
- L_{max} Length of the longest edge of the triangle.
- P Perimeter of the triangle.

The formula for the shape quality of a 2-D quadrangle writes:

$$Q_s = 8\sqrt{2}\,\frac{S_{\min}}{L_{\max}\,P}$$

with:

- S_{min} Minimum area of the 4 triangles.
- L_{max} Max length of the four sides and the two diagonals.
- P Perimeter of the quadrangle.

The formula for the shape quality of a 3-D quadrangle writes:

$$Q_s^{3D} = Q_s^{2D}\ Q_w$$

with:

Q_w Warp quality of the quadrangle

$$Q_w = 1 - \frac{\text{acos}\left(\min\left(\langle N_0, N_2\rangle, \langle N_1, N_3\rangle\right)\right)}{\pi}$$

N_i Normal to the quad at node i

Histograms

Histograms `histo_Qs_in`, `histo_Qs` and `histo_Qh` (OUT).

`histo_Qs_in` is the histogram of the shape qualities in the initial mesh. `histo_Qs` is the histogram of the vector `shape_qualities` in the final mesh.

`histo_Qh` is the histogram of the normalized edge lengths in the final mesh (computed only when the option flag `compute_Qh_flag` is up, see Section 3).

Each histogram stores the minimum, the maximum and the average value as data members.

Hard faces

Vector `hard_faces` (IN).

Put in this vector the IDs of the initial triangles (column IDs in input matrix `connectM`) that must be kept in the final mesh. Their edges and nodes will be also considered as hard edges and hard nodes.

Skeleton edges of the initial mesh

Matrix `skeleton_edges_in` of dimension 2 x NE_{ski} (IN/OUT).

These skeleton edges are the boundaries of the patches in the initial mesh. They are the *stitches* in the initial mesh found by the partitioning algorithm or specified by user. The skeleton lines are remeshed in the final mesh.

Only the edges in the initial mesh are considered.

Skeleton edges of the final mesh

Matrix `skeleton_edges` of dimension 2 x NE_{skf} (OUT).

These skeleton edges are the *remeshed* skeleton lines. They are the boundaries of the patches in the final mesh.

Edges to exclude from the skeleton edge set

Matrix `exclude_skeleton_edges` of dimension 2 x NE_{xsk} (IN).

User can put in this matrix the edges that should not be considered as skeleton edges, if possible. However, only the edges that violate the `patch_angle_tolerance` criterion can be excluded this way. The algorithm will keep any other skeleton edge if it violates any other criterion (`strain_tolerance` or if it is impossible to remesh without splitting along this edge).

Only the edges in the initial mesh are considered.

Hard edges

Matrix `hard_edges` of dimension 2 x NE_{he} (IN).

Hard edges are skeleton edges that should not be remeshed. Their vertices are considered as hard nodes.

Only the edges in the initial mesh are considered.

Hard nodes

Vector `hard_nodes` (IN/OUT).

Upon entry, the user can put in this vector some node IDs to constrain them to remain in the final mesh. Upon exit, this vector contains in addition all the skeleton nodes, e.g. the topologically singular nodes (nodes connected to one, three or more skeleton lines) and the nodes of the skeleton lines for which the angle between coincident skeleton lines is greater than `patch_angle_tolerance`.

Only the nodes in the initial mesh are considered.

Nodes to exclude from the hard node set

Vector `exclude_hard_nodes` (IN).

This vector contains the nodes that should not be selected as skeleton nodes, if possible.Only the nodes that violate the `patch_angle_tolerance` criterion of the skeleton lines can be excluded. The algorithm will keep any other skeleton node, as well as the user-defined hard nodes.

Only the nodes in the initial mesh are considered.

Field to interpolate on the new mesh

Matrix `interpolate_field` (IN/OUT).

This matrix contains nodal data to be interpolated to the new mesh (such as FEA data, textures...)
Data for node #i is at column #i (can be a single scalar or a vector). As new nodes are appended to the coordinate matrix `pos`, new values are linearly interpolated and appended to this matrix[4].

Initial elements' color

Vector `colors_in` (IN/OUT).

The patch ID (color) of the initial elements.

Automatic coloring when `colors_in` is empty upon entry:

- The partitioning algorithm use its own criterions (`settings_type::patch_angle_tolerance`, `settings_type::strain_tolerance`, `settings_type::max_chordal_error`, `skeleton_edges_in`, `exclude_skeleton_edges`) to partition the initial mesh and set colors to the patches.
- User has little control over the patches (only through the above criterions) and none over the color values.

Patches are colored consecutively starting from 0.
Elements eliminated during the initial clean-up and final optimization steps have no color ID (`CM2_NONE`).

[4] To eliminate the old nodes together with their data, use the following code snippet after the remeshing:

```
UIntMat connectM;
connectM.copy(data.connectM);                                    // Copy of initial connectivity.
cm2::meshtools::simplify(data.pos, connectM);                    // Eliminate unused nodes.
cm2::meshtools::simplify(data.interpolate_field, data.connectM); // Eliminate unused data.
```

User coloring when `colors_in` is not empty upon entry:

- The boundaries between different colors are considered as skeleton edges and honored as such.
- User can provide colors for only some elements in the initial mesh (*partial user coloring*).
The boundaries between colored elements and non-colored elements are also considered as skeleton edges.
- The partitioning algorithm can add other skeleton edges (angulous edges or in order to respect other criterions such as `settings_type::strain_tolerance`, `settings_type::max_chordal_error`...)
- Additional color values (when partial user coloring) start with the next highest value given by user in `colors_in` and increase consecutively.

Final elements' color

Vector `colors` (OUT).

The patch ID (color) of the final elements.

Elements with the same color in `colors` are the remeshed elements (children) from the initial elements (parents) with that color in `colors_in`.

Initial skeleton edges' color

Vector `skeleton_colors_in` (IN/OUT).

The skeleton line ID (color) of the initial skeleton edges

Automatic coloring when `skeleton_colors_in` is empty upon entry:

- The color of the skeleton lines is defined by the partitioning algorithm (one color value by line with continuous numbering starting with 0).
- User has little control over the lines and none over the color values.
Skeleton lines are colored consecutively starting from 0.
Edges eliminated during the initial clean-up and final optimization steps have no color ID (`CM2_NONE`).

User coloring when `skeleton_colors_in` is not empty upon entry:

- The extremities of colored lines are considered as hard nodes and honored as such.
- The partitioning algorithm can add other skeleton nodes (angulous nodes).

Additional line color values (when *partial user coloring*) start with the next highest value given by user in `skeleton_colors_in` and increase consecutively.

Final skeleton edges' color

Vector `skeleton_colors` (OUT).

The skeleton line ID (color) of the final skeleton edges.

Skeleton edges with the same color in `skeleton_colors` are the remeshed edges (children) from the initial edges (parents) with that color in `skeleton_colors_in`.

Maximum error distance

Scalar `max_error_distance` (OUT).

This is an approximate maximum distance between the initial mesh and the final mesh (so-called *Hausdorff* distance). This is a measure of the geometric error between the two meshes.

Errors and warnings

Two enums, `error_code` and `warning_code`, give upon exit the error and warning codes if any (OUT).

A correct run gives the `CM2_NO_ERROR` and `CM2_NO_WARNING` codes (zero values). See Section 2 for more detailed explanations of the error and warning codes.

String `msg1` holds explanation messages about the error/warning if any. In case of error, the meshing process is aborted and the output mesh (`connectM`) is left unchanged (equal to the input mesh). With a mere warning, the process goes to its end though the final mesh may have a poor quality.

String `msg2` is for debugging purpose and shouldn't normally be used on production code.

Complementary information

In this section are gathered the remaining fields of the data structure (OUT):

- `nefs_in`: the number of elements in the initial mesh (also equals to the number of columns in `connectM` upon entry). `nefs_in = nefs_Q4_in` + `nefs_T3_in`.
- `nefs_T3_in`: the number of triangles in the initial mesh.
- `nefs_Q4_in`: the number of quadrangles in the initial mesh (null with **CM2 SurfRemesh T3**).
- `nefs`: the number of elements in the final mesh (also equals to the number of columns in `connectM` upon exit). `nefs = nefs_Q4` + `nefs_T3`.
- `nefs_T3`: the number of triangles in the final mesh (null with **CM2 SurfRemesh Q4** in all-quad mode).
- `nefs_Q4`: the number of quadrangles in the final mesh (null with **CM2 SurfRemesh T3**).
- `nods_in`: the number of nodes in the initial mesh (less or equal to the number of columns in `pos` upon entry).
- `nods`: the number of nodes in the final mesh (less or equal to the number of columns in `pos` upon exit).
- `total_nods`: the total number of points (i.e. columns) in matrix `pos` upon exit.
- `area_in`: the area of the initial mesh. `area_in = area_Q4_in` + `area_T3_in`
- `area_T3_in`: the area of the triangles in the initial mesh.
- `nefs_Q4_in`: the area of the quadrangles in the initial mesh (null with **CM2 SurfRemesh T3**).
- `area`: the area of the final mesh. `area = area_Q4` + `area_T3`
- `area_T3`: the area of the triangles in the final mesh (null with **CM2 SurfRemesh Q4** in all-quad mode).
- `area_Q4`: the area of the quadrangles in the final mesh (null with **CM2 SurfRemesh T3**).
- `Qmin_in`: the worst shape quality of the elements in the initial mesh.
- `Qmin`: the worst shape quality of the elements in the final mesh.
- `error_code`, `warning_code`: the error and warning codes (see Section 2).
- `msg1`: the error message.
- `cleaning_time`, `partition_time`, `remesh_time`, `optim_time`: the times spent in the successive steps of the remesher.
- `total_time`: the total time spent in the remesher.
- `speed`: the global speed of the remesher (number of generated elements per second).

```
struct data_type
{
   DoubleMat               pos;
   UIntMat                 connectM;
   DoubleVec               metrics;

   UIntMat                 neighbors;
   UIntVec                 ancestors;
   DoubleVec               shape_qualities;
   misc::histogram         histo_Qs_in;
   misc::histogram         histo_Qs;
   misc::histogram         histo_Qh;

   UIntVec                 hard_faces;
   UIntMat                 skeleton_edges_in;
   UIntMat                 skeleton_edges;
   UIntMat                 exclude_skeleton_edges;
   UIntMat                 hard_edges;
   UIntVec                 hard_nodes;
   UIntVec                 exclude_hard_nodes;

   DoubleMat               interpolate_field;

   UIntVec                 colors_in;
   UIntVec                 colors;
   UIntVec                 skeleton_colors_in;
   UIntVec                 skeleton_colors;

   double                  max_error_distance;

   unsigned                nefs_in;
   unsigned                nods_in;
   double                  area_in;
   double                  Qmin_in;

   unsigned                nefs;
   unsigned                nefs_Q4;
   unsigned                nefs_T3;
   unsigned                nods;
   double                  area;
   double                  area_Q4;
   double                  area_T3;
   double                  Qmin;

   unsigned                total_nods;

   double                  cleaning_time;
   double                  partition_time;
   double                  remesh_time;
   double                  optim_time;
   double                  total_time;
   double                  speed;

   error_type              error_code;
   warning_type            warning_code;
   char                    msg1[256];
};
```

Table 1 – The `data_type` structures.

Function members

The `data_type` are equipped with constructors, copy operator and other utility members:

```
// Constructors.
data_type();
data_type (const data_type& cpy);
data_type (const DoubleMat& P, const UIntMat& connectM);

// Copy operator.
const data_type& operator= (const data_type& data);

// Members.
void shallow_copy (const data_type& data);
bool valid() const;
int check() const;
void clear();
void clear_data_out();
void extract (DoubleMat& P, UIntMat& connectM) const;
void extract (DoubleMat& P, UIntMat& connectQ4, UIntMat& connectT3) const;
void IDs_sorted_by_color_in (UIntVec& color_order, UIntVec& xColor) const;
void IDs_sorted_by_color (UIntVec& color_order, UIntVec& xColor) const;
void skeleton_IDs_sorted_by_color_in (UIntVec& color_order, UIntVec& xColor) const;
void skeleton_IDs_sorted_by_color (UIntVec& color_order, UIntVec& xColor) const;
void save (const char* filename, unsigned precision = 16) const;
int load (const char* filename);
void print_info (display_handler_type hdl) const;
bool error_raised() const;
bool warning_raised() const;
```

Table 2 – The `data_type` function members.

The `IDs_sorted_by_color_in` member gets the order of the initial elements according to their color.
The `IDs_sorted_by_color` member gets the order of the final elements according to their color.
The `skeleton_IDs_sorted_by_color_in` member gets the order of the initial skeleton edges according to their color.
The `skeleton_IDs_sorted_by_color` member gets the order of the final skeleton edges according to their color.

Refer to the HTML reference manual for more details.

2. Error and warning codes

Error codes

The error code is a public field of the class `data_type::error_code` as an enum of type `error_type`.

```
enum error_type
{
   CM2_NO_ERROR,                    // 0
   CM2_LICENSE_ERROR,               // -100
   CM2_SETTINGS_ERROR,              // -101
   CM2_DATA_ERROR,                  // -102
   CM2_NODES_LIMIT_ERROR,           // -103
   CM2_DEGENERATED_ELEMENT,         // -107
   CM2_REMESHING_ERROR,             // -108
   CM2_SYSTEM_MEMORY_ERROR,         // -199
   CM2_INTERNAL_ERROR               // -200
};
```

Table 3 – Error codes enum.

`CM2_NO_ERROR`	OK, no problem.
`CM2_LICENSE_ERROR`	The registration must occur before the instantiation of any meshers. Check also your license. You may have to renew it. Please, contact <license@computing-objects.com>.
`CM2_SETTINGS_ERROR`	At least one setting is not valid (see Section 3). Check the positivity/range of scalar values such as `shape_quality_weight, max_gradation`...
`CM2_DATA_ERROR`	The input data are not valid. Check the sizes of matrices, of vectors, node indices in the connectivity matrices, look for insane values...
`CM2_NODES_LIMIT_ERROR`	The limit on the number of nodes is too low.
`CM2_DEGENERATED_ELEMENT`	At least one of the final elements is invalid (null area). Can't generate a valid mesh. Check the input data.
`CM2_REMESHING_ERROR`	Can't generate a better mesh. Reduce parameter `min_h`, `max_h`, `patch_angle_tolerance`, `strain_tolerance`.
`CM2_SYSTEM_MEMORY_ERROR`	Insufficient memory available. Mesh is too big to be generated (over several tens millions elements).
`CM2_INTERNAL_ERROR`	Unexpected error. Save the data by calling `data_type::save` and send the file zipped to <support@computing-objects.com>.

Table 4 – Error codes.

In case of error the output mesh in field `data.pos` and `data.connectM` is identical to the input mesh.

For error codes `CM2_DEGENERATED_ELEMENT`, `CM2_REMESHING_ERROR` and `CM2_INTERNAL_ERROR`, save the data by calling `data_type::save`, zip the file and send to <support@computing-objects.com>.

Example

```
if (data.error_raised())
{
  // Error, do something according to data.error_code (forward message to a window...)
}
```

Warning codes

The warning code is located in the structure `data_type`.

```
enum warning_type
{
   CM2_NO_WARNING,                    //   0
   CM2_INTERRUPTION,                  // -10
   CM2_NODES_LIMIT_WARNING,           // -11
   CM2_SHAPE_QUALITY_WARNING,         // -12
   CM2_NON_CLOSED_MANIFOLD_WARNING    // -13
};
```

Table 5 – Warning codes enum.

`CM2_NO_WARNING`	OK, no problem.
`CM2_NODES_LIMIT_WARNING`	The node limit has been reached and the mesh may be far from optimal.
`CM2_INTERRUPTION`	The user has aborted the run (through the interrupt handler). The final mesh may be empty or valid but of poor quality.
`CM2_SHAPE_QUALITY_WARNING`	The final mesh is valid but at least one of the elements is very bad (shape quality < 0.01). Reduce `min_h`, `max_h`. Increase `patch_angle_tolerance`, `strain_tolerance`, `optim_tolerance`...
`CM2_NON_CLOSED_MANIFOLD_WARNING`	The 3-D surface mesh could not be made watertight. This warning can be raised only in the closed-manifold mode (`closed_manifold_flag = true`).

Table 6 – Warning codes.

Example

```
if (data.warning_raised())
{
  // Warning, do something according to data.warning_code (remesh again,
  // increase optimisation level...)
}
```

3. Settings of the remeshers

CM2 SurfRemesh T3/Q4 have several settings that can change drastically their outputs. They are gathered into a structure of type `settings_type` as a public field of the remeshers:

```
cm2::surfremesh_t3::mesher::settings_type   settings;
cm2::surfremesh_q4::mesher::settings_type   settings;
```

Minimum size of the elements

`min_h`. Default = 0. (should be reset by user).

Lower bound for the metric map used to generate the new mesh. This value controls the size of the smallest elements along the skeleton lines[5]. The user's specified values in `metrics` are not limited by this value.

Maximum size of the elements

`max_h`. Default = `DBL_MAX` (should be reset by user).

Upper bound for the metric map used to generate the new mesh. This value controls the size of the largest elements along the skeleton lines. The user's specified values in `metrics` are not limited by this value.

Target mesh size

`target_h`. Default = 0 (should be reset by user).

Whereas `min_h` and `max_h` control the mesh size along the skeleton lines, `target_h` controls the mesh size inside the patches.

With the default value (`target_h` = 0) the size map is interpolated inside the patches from the bounding skeleton lines. With small geometric features (fillets, ridges, holes...) the skeleton lines may be remeshed quite fine (as low as `min_h`) which in turn, through the default interpolation inside the patches, can lead to over-fine mesh and excessive number of elements. Setting `target_h` (for instance to same value as `max_h`) mitigates this effect. The elements size tends (according to the `max_gradation_2d` parameter) toward this value (if > 0) as we go away from the skeleton lines.

When `min_h` = `max_h` you can leave `target_h` = 0 (the default). However, when `min_h` << `max_h`, it's a good practice to set `target_h` to same value as `max_h` to keep the total number of elements as low as possible.

Max chordal error

`max_chordal_error`. Default = -0.02 (i.e. 2% of local radius).

Maximum chordal error allowed. The mesh size is reduced locally to limit the chordal error between the mesh and the surface:

- If negative, this value is relative to the local radii (for instance -0.01 => max chordal error = 1% of local radii).
- If positive, this value is absolute (for instance 0.1 => max chordal error = 0.1).

Max gradation 1d

`max_gradation_1d`. Positive value. Default = 0.5

[5] However some elements may be smaller to satisfy geometric constraints.

This parameter controls the gradation of the element sizes along the skeleton lines towards the `max_h` mesh size. A value close to 0 leads to a more progressive variation (smoother) of the mesh sizes along the skeleton lines.

When `max_gradation_1d` = 0, the sizes are linearly interpolated between the sizes at the skeleton nodes (which depend on `min_h`, `max_h` and `max_chordal_error`).

Max gradation 2d

`max_gradation_2d`. Positive value. Default = 0.5

This parameter controls the gradation of the element sizes inside the patches towards the `target_h` mesh size. A value close to 0 leads to a more progressive variation (smoother) of the mesh sizes inside patches.

When `max_gradation_2d` = 0, `target_h` is discarded and the sizes are linearly interpolated between sizes along the skeleton lines.

Patch angle tolerance

`patch_angle_tolerance`. Default = 20.

Parameter used to delimit the patches (groups of connected triangles in the initial mesh).

An angle between two adjacent triangles greater than this value will draw a limit between two patches (we call such limit a *skeleton edge*). A small value tends to give numerous small patches and the final mesh will be close to the initial mesh but may be of poor quality. A big value tends to give fewer bigger patches. The remesher can do a better job on big patches but the final mesh may be more distant from the initial mesh (increased Hausdorff distance).

A value of 45° should be considered as the maximum for this parameter.

Initial fixing tolerance

`fix_tolerance`. Default = -0.0025 (i.e. 0.25% of `min_h`).

Controls the merging of nodes and the swapping of edges during the initial clean-up step.

This parameter is the maximum allowable distance (absolute or relative to `min_h`) that a node merging or an edge swapping can induce on the initial surface.

If `fix_tolerance` > 0, `fix_tolerance` is taken as an absolute tolerance.
If `fix_tolerance` < 0, this is a relative tolerance. The distance of merging is then computed as the product of `-fix_tolerance` and `min_h` or, if `min_h` is null, as the product of `-fix_tolerance` and a mean of the lengths of the edges in the initial mesh.

This parameter should be used only to treat pathologically close nodes that are present in the initial mesh.

A value of 1% (i.e. -0.01) should be considered as the maximum for this parameter.

Final optimization tolerance

`optim_tolerance`. Default = -0.05 (i.e. 5% of min_h).

Controls the merging of nodes and the swapping of edges during the final optimization step.

This parameter is the maximum allowable distance (absolute or relative to `min_h`) that a node merging or an edge swapping can induce on the final surface.

If `optim_tolerance` > 0, `optim_tolerance` is taken as an absolute tolerance.

If `optim_tolerance` < 0, this is a relative tolerance. The distance of merging is then computed as the product of -`optim_tolerance` and `min_h`, or if `min_h` is null, as the product of -`optim_tolerance` and a mean of the lengths of the edges in the initial mesh.

A value of 20% (i.e. -0.20) should be considered as the maximum for this parameter.

Strain tolerance

`strain_tolerance`. Default = 0.30

This is the maximum allowable strain for the elements in a patch when unfolded.

Set `strain_tolerance` = 0 if no strain is allowed. Only perfectly unfoldable patches will be selected in this case (i.e. flat or simple-curved patches).
A small value tends to reduce the size of the patches. A large value tends to give fewer bigger patches.

A value of 0.60 should be considered as the maximum for this parameter.

Initial clean-up

`initial_cleanup_flag`. Default = `true`.

Flag to allow a clean-up of the initial mesh (node merging, gaps filling and topological fixing).

Closing initial notches

`notch_closing_flag`. Default = `true`.

Flag to allow the closing of notches during the clean-up phase. Triggers a specific algorithm within the initial clean-up step to close open nodes (merged to the closest open nodes).

Disabled when `initial_cleanup_flag` = `false`.

Closed manifolds

`closed_manifold_flag`. Default = `false`.

Flag to tell if the 3-D surface should be considered as closed and simple (no internal faces) or not.

If `true`, a specific algorithm is used to correct the initial mesh when some elements are under-connected or over-connected (edges not shared exactly by two elements).
The remesher emits a warning (`CM2_NON_CLOSED_MANIFOLD_WARNING`) when it cannot enforce all edges in the initial mesh to be shared by exactly two elements.

Line proximity detection

`line_proximity_detection_flag`. Default = `true`.

Flag to take into account (true) or not (false) the distance between skeleton lines for the size of the elements.

If `true`, the size of the elements along close skeleton line is reduced (depending on the distance between these skeleton lines, `min_h`, `max_gradation_1d`, `max_gradation_2d`).
If `false`, the distance between the skeleton lines isn't considered for the sizes, but their lengths remain considered near their extremities (and these mesh sizes can be propagated along the skeleton lines and inside the surface patches through `max_gradation_1d` and `max_gradation_2d`).

All-quad or quad-dominant mode (CM2 SurfRemesh Q4)

`all_quad_flag`. Default = `false`.

Flag to force the generation of an all-quad mesh (no triangle):

- `false`: generates a mixed triangle-quad mesh (the default).
- `true`: generate a all-quad mesh.

A good all-quad mesh can only be generated in simple cases. If possible, rather use the mixed quad-dominant generation mode.

When this flag is set to `false`, parameter `quadrangle_weight` can be used to control the trade-off between meshes with more numerous quads and meshes with more numerous triangles but of better shape quality.

Remeshing

`remesh_flag`. Default = `true`.

This flag enables the remeshing of the patches.

Set this flag to `false` if you don't want to remesh but only to optimize it (or simply to compute the partition patches in the initial mesh, the skeleton edges and the skeleton nodes).

☞ Note that the initial clean-up and final optimization, if not disabled, may still modify the mesh.

Parity along the skeleton lines

`force_even_flag`. Default = `false`.

Flag to force the number of edges in the final skeleton lines to be even.

This is always the case with **CM2 SurfRemesh Q4** when `all_quad_flag` is set to `true`.

Node smoothing

`node_smoothing_flag`. Default = `true`.

This flag controls the node-smoothing scheme in the optimization step.

When `remeshing_flag` = `false`, node smoothing doesn't change the mesh connectivity, only the coordinates of nodes.

This flag has no effect when the optimization step is disabled (`optim_level` = 0).

Node inserting

`node_inserting_flag`. Default = `true`.

This flag controls the node-inserting scheme in the optimization step.

When `remeshing_flag` = `false`, node inserting increases the number of nodes, changes the mesh connectivity, but doesn't change the other nodes' coordinates.

This flag has no effect when the optimization step is disabled (`optim_level` = 0).

Node removing

`node_removing_flag`. Default = `true`.

This flag controls the node-removing scheme in the optimization step.

When `remeshing_flag` = `false`, node removing decreases the number of nodes, changes the mesh connectivity, but doesn't change the other nodes' coordinates.

This flag has no effect when the optimization step is disabled (`optim_level` = 0).

Shell remeshing

`shell_remeshing_flag`. Default = `true`.

This flag controls the local remeshing scheme in the optimization step.

When `remeshing_flag` = `false`, shell remeshing changes the mesh connectivity, but doesn't change the number of nodes nor their coordinates.

This flag has no effect when the optimization step is disabled (`optim_level` = 0).

Final optimization

`final_optimization_flag`. Default = `true`.

Flag to allow the final inter-patch optimization scheme.

If true, nodes can be merged together if they are within the merging distance (`optim_tolerance`) and edges can be swapped if the induced geometric error is lower than `optim_tolerance`.

Computation of the size-qualities histogram

`compute_Qh_flag`. Default = `false`.

Before exiting the process, this flag tells the mesher to compute the histogram of the size quality of all the edges in the new mesh.

Deep analyzing

`force_deep_analyzing_flag`. Default = `false`.

Flag to force deep analyzing of the model.

If set to `false`, the patch-finding algorithm starts in 'fast' mode: it assumes first that the flat patches have no self-intersection and check the meshability for curved patches only.

In case of remeshing failure (when a flat patch has self-intersection), it falls back to the deep-analyzing mode (where all patches are systematically checked for non self-intersection and meshability).
If you know that your model has flat self-intersections, you can save some CPU time by setting this flag to true from the beginning.

☞ In patch-finding-only mode (`remesh_flag` = `false`, `optim_level` = 0) where no remeshing is performed, the algorithm cannot fall back to deep-analyzing mode. Hence in case of flat self-intersections, correct patch-finding requires this flag to bet set to `true`.

Pattern for structured meshes (CM2 SurfRemesh T3)

`structured_pattern`. Default = -1

This option controls the way the generators does the structured meshes when possible (on rectangular-like patches). It can take four values:

- -1: the meshes are always done with the frontal-Delaunay algorithm (the default).
- 0: when possible, generates structured left-oriented meshes (simply oriented pattern).
- 1: when possible, generates structured right-oriented meshes (simply oriented pattern).
- 2+: when possible, generates structured UJ meshes (Union Jack pattern).

Pattern for structured meshes (CM2 SurfRemesh Q4)

`structured_flag`. Default = `true`.

This option controls the way the generators do the structured meshes when possible (on rectangular-like patches):

- `true`: when possible, generates structured (grid-like) meshes (the default).
- `false`: the quad meshes are always done with the frontal-Delaunay algorithm[6].

Limit on the number of nodes

`nodes_limit`. Default = `UINT_MAX`.

When the mesh generator reaches this limit, the `CM2_NODES_LIMIT_WARNING` is issued and new nodes are no longer created. In this case, the quality of some elements can be far from optimal. When the limit is so low that the remesher cannot even insert all the skeleton nodes the `CM2_NODES_LIMIT_ERROR` is raised and the mesh is aborted.

Optimization level

`optim_level`. Integer between 0 and 10. Default = 3

A null value makes the mesher to skip the optimization step. The speed is maximal but the quality may be poor. From value 1 on, the optimizer algorithm uses several techniques to improve both the shape quality and the size quality of the elements, such as node smoothing, edge swapping, node insertion and node removal.
Level 3 is usually a good trade-off between quality and speed.

Weight on shape quality

`shape_quality_weight`. Value between 0 and 1. Default = 0.6

This parameter controls the trade-off between shape optimization and size optimization. It is the weight of the shape quality in the measure of the global quality of an element.
The default value (0.6) gives a slight preference to the shape quality over the size quality.

Weight on quadrangles (CM2 SurfRemesh Q4)

`quadrangle_weight`. Value between 0 and 1. Default = 0.70

Preference for quadrangles vs. triangles.

Weight between 0 and 1 indicating the preference for quadrangles over triangles, when a all-quad mesh cannot be generated (i.e. when there is no parity on the boundaries or `all_quad_flag` = `false`).

[6] This may give also structured meshes. The true option enforces this and is faster whenever a structured mesh can be generated.

With `quadrangle_weight` = 0, quadrangles are never used. With `quadrangle_weight` = 0.5, quadrangles are used only when they improve the quality of the mesh. For values between 0.5 and 1, quadrangles are more and more used even if this lead to a lesser quality of the mesh. With `quadrangle_weight` = 1, the minimum number of triangles are used (but may not be null).

This parameter is used only when `all_quad_flag` = `false`.

This parameter is not the ratio between quads and triangles. Furthermore, there is no linearity between this weight and the ratio between quads and triangles.

Minimum quadrangle quality (CM2 SurfRemesh Q4)

`min_Q4_angle_quality`. Double value between 0 and 1. Default = 0 (no minimum).

Minimum acceptable angle quality for the quadrangles. This parameter is taken into account in mixed mode only (`all_quad_flag` = `false`).

This quality threshold is based on the angle quality of the quads (not the geometric quality which takes the length ratios also into account). The angle quality is computed as the minimum of the four angles at summits[7]

Set `min_Q4_angle_quality` = 1 - ε to allow rectangles only (quads with right angles only). In this case, be aware that when boundaries are not straight very few rectangles may be generated.

Minimum warp quality (CM2 SurfRemesh Q4)

`min_Q4_warp_quality`. Double value between 0 and 1. Default = 0.50 (maximum warpage of 90°).

Minimum acceptable warp quality for the quadrangles. Quads with warp quality below this threshold are split into two triangles. This parameter is taken into account in mixed mode only (`all_quad_flag` = `false`).

If α is the maximum angle between the normals of two opposite split triangles. The warp quality of a quadrangle is defined as $1 - \alpha/\pi$. The *warp* quality of a flat quad equals to 1 (perfect) whereas its *geometric* quality is only equal to 1 when the quad is a square.

Set `min_Q4_warp_quality` = 1 - ε to allow for flat quads only. In this case, be aware that very few rectangles may be generated (mostly triangles).

Upper bound on edges length

`length_upper_bound`. Value greater than 0. Default = 1.414

This parameter is used to limit the length of the edges in the generated mesh (normalized length). This is not a strict enforcement however. Most of the edges will be shorter than this limit, but some may remain somewhat longer. The default value (1.414) gives the optimal meshes with respect to the size qualities. With this default value, the average edge length tends to be 1 (optimal edge quality on average).

Sometimes, it can be useful to limit the length of the edges to a shorter value (usually between 1 and 1.414), and to accept an average value smaller than 1 (sub-optimal edge qualities on average).

Minimum number of edges along loops

`min_NE_for_loops`. Integer greater or equal to 3. Default = 6

This parameter can be useful when holes or loops need to be remeshed with a minimum number of edges. Works only for surface domains, not solid domains.

[7] The angle quality of any flat rectangle equals to 1 whereas its geometric quality only equals 1 for a square.

Maximum number of edges along loops

`max_NE_for_loops`. Integer greater or equal to 3. Default = `UINT_MAX`.

This parameter can be useful when holes or loops need to be remeshed with a limited number of edges. Works only for surface domains, not solid domains.

Number of threads for parallel processing

`num_threads`. Integer ≥ 0. Default = 0 (= the number of logical processors on the running CPU).

If `num_threads` = 1 the mesh generator runs on a single thread. Values are capped by the number of logical processors on the running CPU.

Display handler

`display_hdl`. Default = `NULL`.

This user-supplied function is used to handle the messages issued by the mesher.

```
typedef void (*display_handler_type) (void* pass_thru, unsigned level, const char* msg);
```

The `pass_thru` parameter is the pointer set by the user in the settings structure.

The `level` parameter gives the importance of the message:

- 0: Important (for instance entering a major step of the process)
- 1: Somewhat important (minor step of the process)
- 2+: Not serious (debug messages that should not be printed for end-users).

The `msg` parameter is the string message (length ≤ 255 characters).

Note: This handler is not called in case of error or warning. At the end of the run, the user must check for an error or a warning in the fields `data_type::error_code` and `data_type::warning_code` and then (in case of error or warning) process the string `data_type::msg1`.

Example

```
void my_display_handler (void* pass_thru, unsigned level, const char* msg)
{
   window_type*   my_window = static_cast<window_type*>(pass_thru);
   my_window->show(msg);
}

cm2::surfremesh_t3::mesher              my_mesher;
cm2::surfremesh_t3::mesher::data_type   my_data(pos, connectB);
window_type                             my_window;   // A “window” instance.

my_window.init(...);          // Initialize the window somehow.
my_mesher.settings.display_hdl = &my_display_handler;
my_mesher.settings.pass_thru = static_cast<void*>(&my_window);
my_mesher.run(my_data);       // Will call my_display_handler  with “my_window”
                              // in pass_thru parameter.
```

Interrupt handler

`interrupt_hdl`. Default = `NULL`. Used in all modes.

Can be useful for big meshes (several thousands of elements).

```
typedef bool (*interrupt_handler_type) (void* pass_thru, double progress);
```

This handler, if any, is called periodically by the tool to check for a stop signal. When the handler returns true, the tool aborts and the final mesh is usually non valid (some patches may be missing).

An interruption also raises the `CM2_INTERRUPTION` warning.

The `pass_thru` parameter is the `void*` pointer set by the user in `settings_type::pass_thru` (the same parameter is also passed to the display handler).
The parameter `progress` (between 0 and 1) gives a hint about the progress of the meshing.

Example (time-out)

```
bool my_interrupt_handler (void* pass_thru, double progress)
{
   clock_t*   t_timeout = static_cast<clock_t*>(pass_thru);
   return clock() > (*t_timeout);
}

cm2::surfremesh_t3::mesher            my_mesher;
cm2::surfremesh_t3::mesher::data_type my_data(pos, connectB);
clock_t                               my_t_timeout(::clock() + 1E3*CLOCKS_PER_SEC);

my_mesher.settings.interrupt_hdl = &my_interrupt_handler;
my_mesher.settings.pass_thru = static_cast<void*>(&my_t_timeout);
my_mesher.run(my_data);         // Will stop if duration > 1000 s.
```

Example (progress bar)

```
bool my_interrupt_handler (void* pass_thru, double progress)
{
   window_type*         my_window = static_cast<window_type*>(pass_thru);
   std::string          msg(“Progress: “);

   msg += std::to_string(static_cast<int>(100.*progress)) + “ %”;
   my_window->show(msg);
   return false;
}

cm2::surfremesh_t3::mesher             my_mesher;
cm2::surfremesh_t3::mesher::data_type  my_data(pos, connectB);
window_type                            my_window;   // A window instance.

my_window.init(...);          // Initialize the window somehow.
my_mesher.settings.interrupt_hdl = &my_interrupt_handler;
my_mesher.settings.pass_thru = static_cast<void*>(&my_window);
my_mesher.run(my_data);       // Will call my_interrupt_handler with “my_window”
                              // in pass_thru parameter.
```

Pass-through pointer

`pass_thru`. Default = `NULL`.

Void pointer (to be cast) to pass some user structure/class to the display handler and to the interruption

handler.

```
struct settings_type
{
   double                   min_h;
   double                   max_h;
   double                   target_h;
   double                   max_chordal_error;
   double                   max_gradation_1d;
   double                   max_gradation_2d;
   double                   patch_angle_tolerance;
   double                   fix_tolerance;
   double                   optim_tolerance;
   double                   strain_tolerance;
   bool                     initial_cleanup_flag;
   bool                     notch_closing_flag;
   bool                     closed_manifold_flag;
   bool                     line_proximity_detection_flag;
   bool                     remesh_flag;
   bool                     force_even_flag;
   bool                     node_smoothing_flag;
   bool                     node_inserting_flag;
   bool                     node_removing_flag;
   bool                     shell_remeshing_flag;
   bool                     final_optimization_flag;
   bool                     compute_Qh_flag;
   bool                     force_deep_analyzing_flag;
   int                      structured_pattern;
   unsigned                 nodes_limit;
   unsigned                 optim_level;
   double                   shape_quality_weight;
   double                   length_upper_bound;
   unsigned                 min_NE_for_loops;
   unsigned                 max_NE_for_loops;
   unsigned                 num_threads;
   display_handler_type     display_hdl;
   interrupt_handler_type   interrupt_hdl;
   void*                    pass_thru;
};
```

Table 7 – The `settings_type` structure (**CM2 SurfRemesh T3**).

```
struct settings_type
{
   double                   min_h;
   double                   max_h;
   double                   target_h;
   double                   max_chordal_error;
   double                   max_gradation_1d;
   double                   max_gradation_2d;
   double                   patch_angle_tolerance;
   double                   fix_tolerance;
   double                   optim_tolerance;
   double                   strain_tolerance;
   bool                     initial_cleanup_flag;
   bool                     notch_closing_flag;
   bool                     closed_manifold_flag;
   bool                     line_proximity_detection_flag;
   bool                     remesh_flag;
   bool                     all_quad_flag;
   bool                     force_even_flag;
   bool                     node_smoothing_flag;
   bool                     node_inserting_flag;
   bool                     node_removing_flag;
   bool                     shell_remeshing_flag;
   bool                     final_optimization_flag;
   bool                     compute_Qh_flag;
   bool                     force_deep_analyzing_flag;
   bool                     structured_flag;
   unsigned                 nodes_limit;
   unsigned                 optim_level;
   double                   shape_quality_weight;
   double                   quadrangle_weight;
   double                   min_Q4_angle_quality;
   double                   length_upper_bound;
   unsigned                 min_NE_for_loops;
   unsigned                 max_NE_for_loops;
   unsigned                 num_threads;
   display_handler_type     display_hdl;
   interrupt_handler_type   interrupt_hdl;
   void*                    pass_thru;
};
```

Table 8 – The `settings_type` structure (**CM2 SurfRemesh Q4**).

Most useful fields are `min_h`, `max_h`, `optim_level` (plus `all_quad_flag` for **CM2 SurfRemesh Q4**) and possibly the handlers. Users can also adapt `patch_angle_tolerance`, `max_chordal_error` or `optim_tolerance` to the specificity of her/his surfaces (to reduce further the Hausdorff error distance between the meshes). Other parameters are rarely useful and should be left to expert users only.

Function members

The `settings_type` is equipped with with default constructor and some utility members:

```
// Constructor.
settings_type();

// Members.
bool valid() const;
int check() const;
void reset();
void save (const char* filename) const;
int load (const char* filename);
```

Table 9 – The `settings_type` function members.

4. General scheme of the remeshers

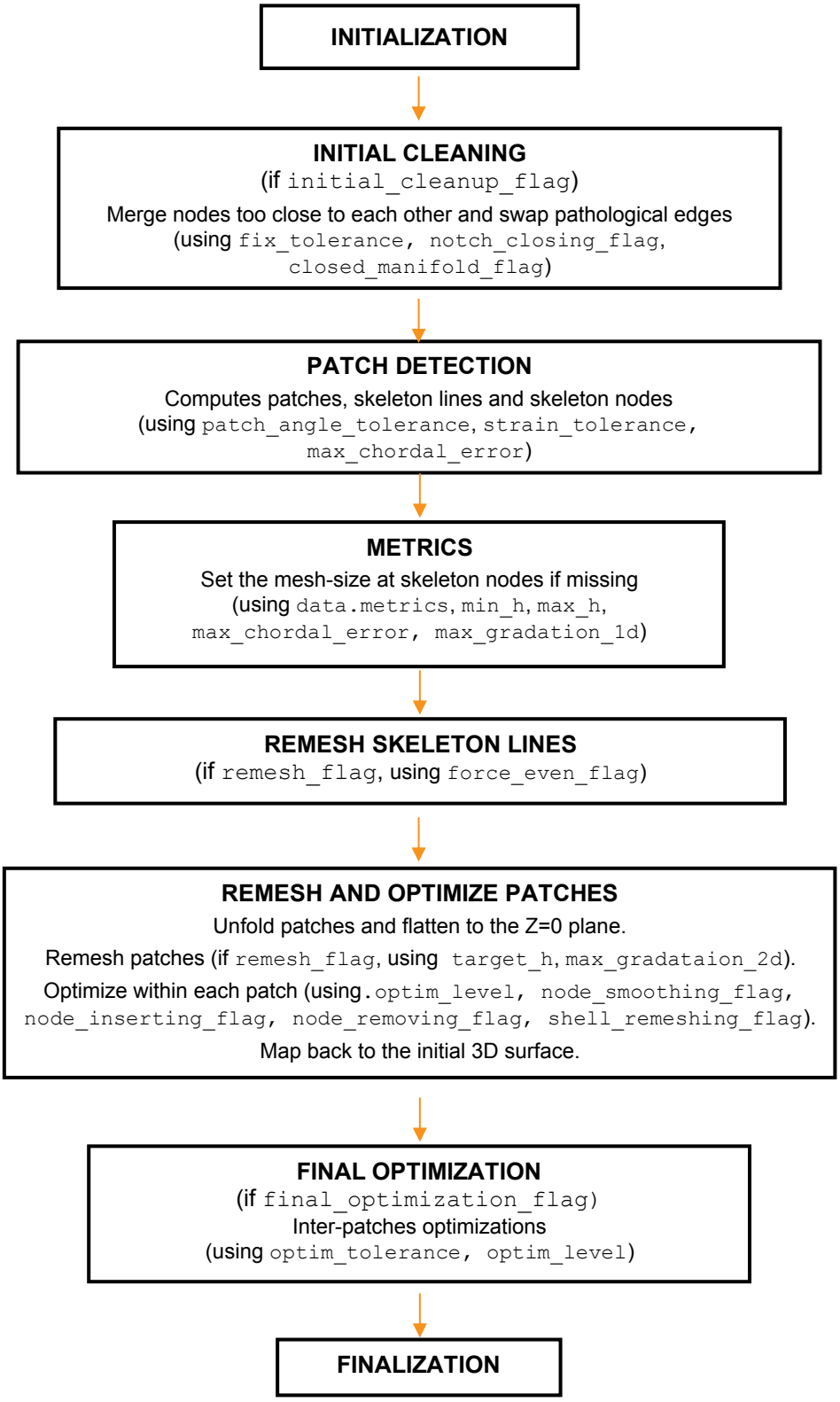

Figure 2 – General scheme of the remeshers.

COMPUTING
OBJECTS

https://www.computing-objects.com