COMPUTING OBJECTS

CM2 TriaMesh® Iso/Aniso
CM2 QuadMesh® Iso/Aniso

Version 5.6

reference manual

Revision February 2025.

https://www.computing-objects.com

© Computing Objects SARL - 25 rue du Maréchal Foch, 78000 Versailles, France.

Forewords

This manual is a reference manual for the structures `data_type` and `settings_type` of the 2-D mesh generators of the **CM2 MeshTools®** SDK:

- The isotropic meshers **CM2 TriaMesh® Iso** and **CM2 QuadMesh® Iso**,
- The anisotropic meshers **CM2 TriaMesh® Aniso** and **CM2 QuadMesh® Aniso**.

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

For examples, please refer to the manual **CM2 TriaMesh & CM2 QuadMesh Iso/Aniso - tutorials**.

Table of contents

Table of contents

1. Data of the mesh generators

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

```
void triamesh_iso::mesher::run (triamesh_iso::mesher::data_type& data);
void quadmesh_iso::mesher::run (quadmesh_iso::mesher::data_type& data);
void triamesh_aniso::mesher::run (triamesh_aniso::mesher::data_type& data);
void quadmesh_aniso::mesher::run (quadmesh_aniso::mesher::data_type& data);
```

The `data_type` structure contains the following fields:

Coordinates of the points

Matrix `pos` of dimensions 2 x `total_nods` (IN/OUT).

This matrix 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 `total_nods` - 1). The *X* coordinates are in the first row and the *Y* coordinates in the second 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.

Hard edges

Matrix `connectB` of dimensions 2 x `hard_edges_in` (IN/OUT).

This matrix stores the connectivity of the edges of the boundary mesh and, if any, the edges of the constrained internal lines. These edges are also called *hard* edges. The node IDs of the edges are stored column-wise. `connectB(i, j)` is the ith (0 or 1) node of the jth edge.

This 1D mesh must normally comply with three conditions:

- it must be valid, i.e. no edge cuts, no degenerated edges[1];
- it must contain at least one closed external boundary;
- if holes or several external boundaries are present, all the edges of the external contour(s) must be oriented the same way (for instance, all counter-clockwise). Edges in the contour of the holes must be oriented the opposite way (for instance clockwise)[2];

This matrix can be left empty upon entry when the mesher is used in `REGULARIZE_MODE` or in `CONVEX_HULL_MODE` (cf. Section 3, Basic operating mode). In `REGULARIZE_MODE`, the mesher will recalculate this boundary mesh. In this case, the `connectB` matrix is also an output data.

The ordering of the edges in the matrix, i.e. ordering of the columns, is irrelevant[3]. Edges of holes can for instance be mixed with edges of the external boundary.

Edges can also be oriented both ways. For instance, some edges of the contour of a hole (so-called negative contour) can be shared with edges of a positive contour (same nodes but opposite orientation).

[1] Condition #1 can be partially leveraged with flag `strict_constraints_flag` = false. With this flag down, cuts between boundary edges are allowed as well as isolated nodes inside an edge (but a warning is returned). See Section 2.

[2] Condition #3 can be leveraged with `subdomains_forcing`. See Section 3.

[3] However, the ordering is taken into account when boundaries are intersecting and mode is non-strict enforcement. In this case, only the lowest intersecting edges are kept.

Isolated (embedded) hard nodes

Vector `isolated_nodes` (IN).

This vector stores the index of the points in the `pos` matrix that the user wants to be present in the final mesh[4] in top of the hard edges nodes. These points must be geometrically distinct and not located on a hard edge[5].

These nodes together with the nodes of the hard edges make the set of the so-called *hard nodes*.

Repulsive points

Vector `repulsive_points` (IN).

This vector contains the index of the points stored in the `pos` matrix that will define areas of node repulsion in the final mesh. These repulsive points must be associated with valid metrics (mesh sizes) to be properly taken into account.

Because of the mesh sizes at the neighboring hard nodes, the disks are mere repulsion disks, not exclusion disks. The closer to the repulsive points, the smaller the probability to find a generated node.

Background mesh

Matrix `background_mesh` (IN) of dimensions 3 x MB.

This is the connectivity matrix of an auxiliary tet mesh used to interpolate the metrics map. Like the other connectivity matrices (`connectB`, `connectM`), the indices refer to columns in the same `pos` coordinates matrix. The metrics at the nodes of this background mesh are stored in the `metrics` array. All nodes of the background mesh must have a valid metric in this array (i.e. a positive value for the isotropic meshers and positive-definite 2 x 2 sym matrix for the anisotropic meshers). This mesh must not contain any degenerated or badly oriented elements.

The background mesh is used to control precisely the size of the elements on the entire domain, not only from the hard nodes of the boundaries. This is very useful for instance in a mesh adaptivity scheme in FEA computations. The mesh at step N + 1 (`connectM`) is generated using the mesh at step N as the background mesh.

The background mesh does not need to cover the entire domain. You can define a background mesh only on a part of the domain and leave the default metric interpolation outside.

If left empty, a default background mesh is used instead for the interpolation of the metrics.

Metric map

Vector (resp. 3-row matrix) `metrics` for the isotropic (resp. anisotropic) meshers (IN/OUT).

Upon entry, the user can specify a mesh size on each hard node or each node of the background mesh. If the value for a node is zero or negative (resp. non positive-definite) or not present, a default metric will be used instead[6]. The 2-D mesh is then generated to fit best the metrics map all over the domain.

For better results, it is recommended to specify a metric value (resp. a metric matrix) only on isolated nodes and leave the default values, i.e. set to zero, on the nodes of the hard edges.

Another solution is to use a background mesh but this requires a valid metric on each of its nodes.

[4] However, if such an isolated point is located outside the domain (or inside a hole), it will not be present in the final mesh. But its coordinates are left unchanged in the `pos` matrix.

[5] Except if flag `strict_constraints_flag` = `false` (see Section 2).

[6] For a node of a boundary or internal line, the default size is the average length of the coincident edges. For an isolated node, the default size is based on the size value of the nearest nodes.

Note that a steep gradient in the metrics map renders the task of the mesher more difficult and surely affects the quality of the elements.

Elements

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

The node IDs of the elements are stored column-wise: `connectM(i, j)` is the ith node of the jth element. They refer to columns in the coordinates matrix `pos`. The local numbering of the nodes is shown Figure 1.

For the triangle meshers the dimensions of this matrix are always 3 x `nefs`.

For the quadrangle meshers, the dimensions are either 4 x `nefs` (in `MESH_MODE` and in `REGULARIZE_MODE`) or 3 x `nefs` (in `CONVEX_HULL_MODE` because only triangles are generated in this case). In MESH_MODE and in `REGULARIZE_MODE`, the leading part of this matrix (from columns 0 to `nefs_Q4` - 1) is the connectivity of the quadrangle elements. The tailing part of this matrix (from columns `nefs_Q4` to `nefs` - 1) is the connectivity of the triangles elements. The fourth node ID in this part of the matrix is always `CM2_NONE` (i.e. unsigned(-1)).

This matrix can be non-empty upon entry when the mesher is used as an optimizer of an already existing mesh (`REGULARIZE_MODE`). Otherwise, it is always an output matrix.

The ordering of the elements in this matrix is irrelevant.

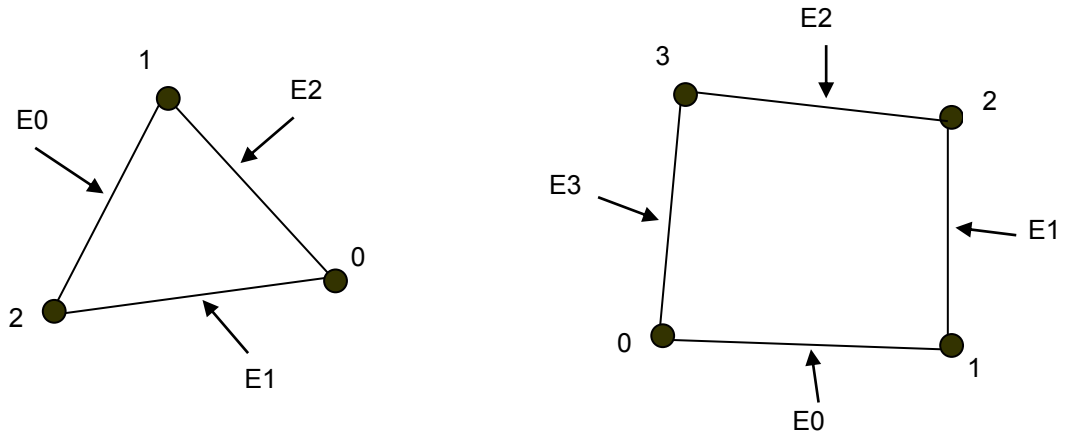

Figure 1 – Nodes and edges local numbering in triangles and quads.

The elements (triangles or quads) are always oriented counter-clockwise - normal towards positive *Z* - whatever the orientation of the edges of the external contour.

Unenforced entities

Vectors `unenforced_boundary_IDs` and `unenforced_node_IDs` (OUT).

Upon exit, these two arrays store the IDs of the entities that could not be enforced (because of intersections between hard entities or because located outside the domain).
In strict-constraints mode (`settings.strict_constraints_flag` = `true`, see Section 3), an error (`CM2_NODE_ERROR` or `CM2_EDGE_ERROR`) is raised when at least one such hard entity cannot be enforced because of intersection. A warning (`CM2_NODE_DISCARDED` or `CM2_EDGE_DISCARDED`) is raised in all other cases (non-strict constraints mode or entities located outside the domain).

Pathological boundaries

Vector `pathological_boundary_IDs` (OUT).

This vector gives the column IDs in the `connectB` matrix of the edges that intersect other hard edge(s)/node(s).

In non-strict mode (`strict_constraints_flag` = false), some of them will be present in the final mesh (the first in `connectB`), some will be missing (the highest in `connectB`).

In strict-constraints mode (`settings.strict_constraints_flag` = `true`), the generator stops with error `CM2_EDGE_ERROR` when this vector is not empty.

Elements' color

Vector `colors` of size `nefs` (IN/OUT).

For each element, this vector gives the index, the so-called color, of the sub-domain[7] in which it belongs. Upon exit, the size of this vector equals to the number of output elements, i.e. number of columns in matrix `connectM`. The color values start at 0 for the most external domain and are incremented each time an internal boundary is encountered from the exterior towards the interior[8].

After meshing, the element colors can be changed to reflect colors on boundaries with `set_colors_by_boundaries`.

In `REGULARIZE_MODE`, the user can provide upon entry a specific `colors` vector[9] (with size equals to the number of input elements) otherwise the meshers will affect color 0 to all elements.

Neighbors

Matrix `neighbors` (OUT) of dimensions 3 x `nefs` for the triangles and 4 x `nefs` for the quads.

This matrix gives, for each element in the final mesh, the indices of the three or four neighboring elements (`CM2_NONE` = unsigned(-1) if none). `neighbors(i, j)` is the neighbor of the jth element sharing the ith edge.

See Figure 1 for the local numbering of the edges.

Ancestors

Vector `ancestors` (OUT) of size `total_nods`.

This vector gives, for each point, the index of one of the elements in which it belongs (`CM2_NONE` = unsigned(-1) if none, i.e. if the point is not in the final mesh).

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

Shape qualities

Vector `shape_qualities` (OUT) of size `nefs`.

This vector gives the shape quality of each element.

[7] A sub-domain means a set of connected elements not fully separated by a hard boundary.

[8] If several sub-domains have the same "level" with respect to the outer boundary, their relative color order is undefined.

[9] Edges between elements with different colors will be considered also as hard boundaries (in addition to those in `connectB`).

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}	Max length of the three sides 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:

$$Q_w = 1 - \frac{a\cos\left(\max\left(\langle N_0, N_2\rangle, \langle N_1, N_3\rangle\right)\right)}{\pi}$$

N_i Normal to the quad at node i.

Histograms

The `histo_Qs` and `histo_Qh` histograms can be used to check for the shape and size qualities of the mesh (OUT).

In the anisotropic case, the shape quality of an element is computed using the metrics at its nodes (minimum value of the qualities computed with the nodal metric transformations).

`histo_Qs` is the histogram of the `shape_qualities` vector. `histo_Qh` is 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.

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 empty. 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

We gather in this section all the remaining fields of the data structure. They are all output values about the final mesh (OUT):

- `nefs`: the number of elements in the mesh - also equals to the number of columns in `connectM`. `nefs` = `nefs_T3` + `nefs_Q4`.
- `nefs_T3`: the number of triangles (same as `nefs` for TriaMesh Iso/Aniso).
- `nefs_Q4`: the number of quads (always null with TriaMesh Iso/Aniso) and the number of triangles (null with QuadMesh Iso/Aniso when `all_quad_flag` = true).
- `total_nods`: the number of points in the the `pos` matrix (i.e. number of columns).
- `nods`: the number of nodes in the mesh (less than or equal to `total_nods`).
- `hard_nodes_in`, `hard_nodes_out:` the number of input and output hard nodes. In strict mode, these two quantities must be equal.
- `hard_edges_in`, `hard_edges_out`: the number of input and output hard edges. In strict mode, these two quantities must be equal.
- `area`, `area_T3`, `area_Q4`: the total (resp. triangles, quads) area of the mesh.
- `Qmin_in`: the worst shape quality of the elements in the mesh upon entry (`REGULARIZE_MODE` only).
- `Qmin`: the worst shape quality of the elements in the final mesh - also available in `histo_Qs`.
- `min_box`, `max_box`: the bounding box of the mesh
- `subdomains`: the number of sub-domains - i.e. the number of different values in the `colors` vector.
- `boundary_sgn`: the orientation of the external boundary: +1 if the contour is counter-clockwise, -1 if the contour is clockwise.
- `error_code`, `warning_code`: the error and warning codes (see Section 2).
- `msg1`: the error message.
- `front_time`, `refine_time`, `optim_time`: the times spent in the successive steps of the mesher.
- `total_time`: the total time spent in the mesher.
- `speed`: the global speed of the mesher (number of generated elements per second).

```
struct data_type
{
   DoubleMat               pos;
   UIntMat                 connectB;
   UIntVec                 isolated_nodes;
   UIntVec                 repulsive_points;
   UIntMat                 background_mesh;
   DoubleVec               metrics;
   UIntMat                 connectM;

   UIntVec                 unenforced_boundary_IDs;
   UIntVec                 unenforced_node_IDs;
   UIntVec                 pathological_boundary_IDs;
   UIntVec                 colors;
   UIntMat                 neighbors;
   UIntVec                 ancestors;
   DoubleVec               shape_qualities;
   misc::histogram         histo_Qs;
   misc::histogram         histo_Qh;

   unsigned                nefs;
   unsigned                nefs_Q4;
   unsigned                nefs_T3;
   unsigned                nods;
   unsigned                total_nods;
   unsigned                hard_nodes_in;
   unsigned                hard_edges_in;
   unsigned                hard_nodes_out;
   unsigned                hard_edges_out;
   double                  area;
   double                  area_Q4;
   double                  area_T3;
   DoubleVec2              min_box;
   DoubleVec2              max_box;
   double                  Qmin_in;
   double                  Qmin;
   unsigned                subdomains;
   int                     boundary_sgn;

   double                  front_time;
   double                  refine_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 (**CM2 TriaMesh Iso, CM2 QuadMesh Iso**).

```
struct data_type
{
   DoubleMat               pos;
   UIntMat                 connectB;
   UIntVec                 isolated_nodes;
   UIntVec                 repulsive_points;
   UIntMat                 background_mesh;
   DoubleMat               metrics;
   UIntMat                 connectM;

   UIntVec                 unenforced_boundary_IDs;
   UIntVec                 unenforced_node_IDs;
   UIntVec                 pathological_boundary_IDs;
   UIntVec                 colors;
   UIntMat                 neighbors;
   UIntVec                 ancestors;
   DoubleVec               shape_qualities;
   misc::histogram         histo_Qs;
   misc::histogram         histo_Qh;

   unsigned                nefs;
   unsigned                nefs_Q4;
   unsigned                nefs_T3;
   unsigned                nods;
   unsigned                total_nods;
   unsigned                hard_nodes_in;
   unsigned                hard_edges_in;
   unsigned                hard_nodes_out;
   unsigned                hard_edges_out;
   double                  area;
   double                  area_Q4;
   double                  area_T3;
   DoubleVec2              min_box;
   DoubleVec2              max_box;
   double                  Qmin_in;
   double                  Qmin;
   unsigned                subdomains;
   int                     boundary_sgn;

   double                  front_time;
   double                  refine_time;
   double                  optim_time;
   double                  total_time;
   double                  speed;

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

Table 2 – The `data_type` structures (**CM2 TriaMesh Aniso, CM2 QuadMesh Aniso**).

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 (UIntVec& color_order, UIntVec& xColor) const;
int get_colors_on_boundaries
    (const UIntMat& B, int orientation_code, UIntVec& colorsB) const;
int get_colors_on_boundaries (UIntVec& colorsB) const;
int set_colors_by_boundaries
    (const UIntMat& B, const UIntVec& colorsB,
     int multicolor_code, int orientation_code, UIntVec& colors_map);
int set_colors_by_boundaries (const UIntVec& colorsB, UIntVec& colors_map);
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 3 – The `data_type` function members (**CM2 TriaMesh Iso/Aniso, CM2 QuadMesh Iso/Aniso**).

The `IDs_sorted_by_color` member gets the order of elements according to their color.
The `get_colors_on_boundaries` members returns the colors of the elements adjacent to the specified boundary faces.
The `set_colors_by_boundaries` members changes the colors of the elements adjacent to the specified boundary faces.

Refer to the HTML reference manual for more details.

2. Error and warning codes

Error codes

```
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_NODE_ERROR,                  // -104
   CM2_EDGE_ERROR,                  // -105
   CM2_BOUNDARY_ERROR,              // -106
   CM2_DEGENERATED_ELEMENT,         // -107
   CM2_BACKGROUND_MESH_ERROR,       // -108
   CM2_SYSTEM_MEMORY_ERROR,         // -199
   CM2_INTERNAL_ERROR               // -200
};
```

Table 4 – Error codes enum for **CM2 TriaMesh Iso/Aniso**.

```
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_NODE_ERROR,                  // -104
   CM2_EDGE_ERROR,                  // -105
   CM2_BOUNDARY_ERROR,              // -106
   CM2_BOUNDARY_PARITY_ERROR,       // -107
   CM2_IRREGULAR_BOUNDARY_ERROR,    // -108
   CM2_DEGENERATED_ELEMENT,         // -109
   CM2_BACKGROUND_MESH_ERROR,       // -110
   CM2_SYSTEM_MEMORY_ERROR,         // -199
   CM2_INTERNAL_ERROR               // -200
};
```

Table 5 – Error codes enum for **CM2 QuadMesh Iso/Aniso**.

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

Example

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

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 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 limitation on nodes number is too low.
CM2_NODE_ERROR	Invalid hard node(s): at least one hard node is in hard entity (node or edge). Strict mode only.
CM2_EDGE_ERROR	Invalid hard edge(s): at least one hard edge is intersecting another hard entity (edge). Strict mode only.
CM2_BOUNDARY_ERROR	The external boundary mesh is flat. There is no domain to be meshed.
CM2_BOUNDARY_PARITY_ERROR	At least one of the hard lines (external boundary or internal line) has an odd number of edges. CM2 QuadMesh Iso/Aniso only.
CM2_IRREGULAR_BOUNDARY_ERROR	At least one of the hard lines (external boundary or internal line) is too irregular. Try to enforce the even condition on each segment of the line. CM2 QuadMesh Iso/Aniso only.
CM2_DEGENERATED_ELEMENT	At least one of the elements is invalid (null or negative area). Check the hard edges and hard nodes.
CM2_BACKGROUND_MESH_ERROR	The background mesh is not valid (not a triangle mesh, nodes not in the pos matrix or degenerated elements).
CM2_SYSTEM_MEMORY_ERROR	Insufficient memory available. Mesh is too big to be generated.
CM2_INTERNAL_ERROR	Unexpected error. Save the data by calling data_type::save and send the file zipped to <support@computing-objects.com>.

Table 6 – Error codes.

This table reflects the priority of the error codes. The highest in the table, the highest priority.

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

Warning codes

```
enum warning_type
{
   CM2_NO_WARNING,                      // 0
   CM2_INTERRUPTION,                    // -10
   CM2_NODES_LIMIT_WARNING,             // -11
   CM2_NODE_DISCARDED,                  // -12
   CM2_EDGE_DISCARDED,                  // -13
   CM2_SHAPE_QUALITY_WARNING            // -14
};
```

Table 7 – Warning codes enum.

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

Example

```
if (data.warning_raised())
{
  // Warning, do something according to data.warning_code (send message to window...)
}
```

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_NODE_DISCARDED	Invalid hard node(s): at least two hard nodes are coincident and one is discarded (when strict_constraints_flag = false) or one is located outside the domain and is discarded.
CM2_EDGE_DISCARDED	Invalid hard edge(s): at least one hard edge is intersecting another hard entity (node or edge) and is discarded (when strict_constraints_flag = false) or one is located outside the domain (but not its nodes) and is discarded.
CM2_BOUNDARY_WARNING	The boundary contour is not close and the mesh is empty or the boundary edges are not uniformly oriented.
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). Check that the discretization of connected lines are not too different.

Table 8 – Warning codes.

This table reflects the priority of the warning codes. The highest in the table, the highest priority.

For warning code CM2_SHAPE_QUALITY_WARNING, check your boundary mesh (connectB and pos).
If good, save the data by calling data_type::save and send the file zipped to <support@computing-objects.com>.

3. Settings of the mesh generators

CM2 TriaMesh Iso and **CM2 QuadMesh Iso** and their anisotropic counterparts can be used as regular meshers or as optimizer of some already existing meshes. For that matter, flags and parameters can be used to adapt the meshers to many various needs.

Some settings are relevant only in the meshing mode, some only in the optimizing mode. They are all gathered into a structure of type `settings_type` as a public member field of the meshers:

```
cm2::triamesh_iso::mesher::settings_type      settings;
cm2::quadmesh_iso::mesher::settings_type      settings;
cm2::triamesh_aniso::mesher::settings_type    settings;
cm2::quadmesh_aniso::mesher::settings_type    settings;
```

Basic operating mode

`basic_mode`. Default = `MESH_MODE`.

The meshers have three distinct operating modes:

- `MESH_MODE` (the default mode) : a mesh is generated from a set of hard edges and possibly some internal hard nodes; the `connectB` matrix and `isolated_nodes` in the data structure are input fields whereas the `connectM` matrix is an output of the mesher;
- `REGULARIZE_MODE` : the mesher is used to improve the quality of an already existing mesh; the `connectM` matrix is both input and output of the mesher;
- `CONVEX_HULL_MODE` : the mesher builds the convex hull of the points; no new node is added; the convex hull is always a *triangle* mesh even with **CM2 QuadMesh® Iso/Aniso;**

Strict constraints enforcement

`strict_constraints_flag`. Default = `true`. Used in `MESH_MODE` only.

This flag tells the mesher to stop on an aborting error (`CM2_EDGE_ERROR, CM2_NODE_ERROR`) if at least one of the constraints is pathological or to simply issue a warning (`CM2_EDGE_DISCARDED, CM2_NODE_DISCARDED`) and go on with the process if possible.

Beware that a rejected edge can make a contour non-closed and give a hole in the domain or merge sub-domains.

Keeping or removing internal holes

`subdomains_forcing`. Default = 0. Used in `MESH_MODE` only.

Forcing mode for intern sub-domains. Possible values are:

- -1: all intern sub-domains are considered as holes regardless of the orientation of their enclosing boundary;
- 0: intern sub-domains are considered as holes when the orientation of their enclosing boundary is opposite to the orientation of the most enclosing domain, otherwise, they are meshed;
- +1: all intern sub-domains are meshed regardless of the orientation of their enclosing boundary;

	strict_constraints_flag	
	false	true
Two hard nodes are coincident (or too close from each other)	Warning CM2_NODE_DISCARDED The highest node is discarded.	Error CM2_NODE_ERROR
A hard node is located inside a hard edge.	Warning CM2_EDGE_DISCARDED The edge is discarded.	Error CM2_NODE_ERROR
Two hard edges cross each other.	Warning CM2_EDGE_DISCARDED The highest edge is discarded.	Error CM2_EDGE_ERROR
A hard node is located outside the domain (or in a hole).	Warning CM2_NODE_DISCARDED The node is discarded.	Warning CM2_NODE_DISCARDED The node is discarded.
A hard edge is located outside the domain (or in a hole) but with nodes on boundary.	Warning CM2_EDGE_DISCARDED The edge is discarded.	Warning CM2_EDGE_DISCARDED The edge is discarded.
A hard edge is located outside the domain (or in a hole), with nodes outside also.	Warning CM2_NODE_DISCARDED The node(s) and the edge are discarded.	Warning CM2_NODE_DISCARDED The node(s) and the edge are discarded.
The contour mesh is not close.	Warning CM2_BOUNDARY_WARNING The mesh is empty.	Warning CM2_BOUNDARY_WARNING The mesh is empty.
The contour mesh is not consistently oriented.	Warning CM2_BOUNDARY_WARNING The ambiguous sub-domains are filled.	Warning CM2_BOUNDARY_WARNING The ambiguous sub-domains are filled.

Table 9 – Effects of the strict_constraints_flag on invalid constraints.

All-quad or quad-dominant mode (CM2 QuadMesh Iso/Aniso only)

all_quad_flag. Default = true. Used in MESH_MODE only.

This flag tells the mesher to generate only quadrangles. When this flag is on, parity of the 1-D meshes along the boundaries and internal lines is generally needed and sufficient to get an all-quad mesh. Otherwise the mesher may fail with a CM2_BOUNDARY_PARITY error[10].

Note that when this flag is off (quad-dominant mode) triangles are not necessarily present in the final mesh (i.e. an all-quad mesh may still be achieved in some cases).

Usually the mesher generates better meshes in quad-dominant mode because of this possibility to use triangles.

[10] The rule of thumb for all-quad meshing is that the domain defined by the twice-coarser boundary (using every other boundary node) is valid (no self intersection).

Refinement

`refine_flag`. Default = `true`. Used in `MESH_MODE` only.

This flag enables the generation of new points inside the domain in order to get good element / size qualities. For **CM2 TriaMesh Iso/Aniso**, this flag off makes the mesher to only triangulate the domain, i.e. it stops after the front mesh step. Only the hard nodes will be in the final mesh and matrix `pos` will be unchanged.

For **CM2 QuadMesh Iso/Aniso** however, a minimal amount of new nodes has to be generated in order to mesh a domain with quads only even with `refine_flag` = `false`.

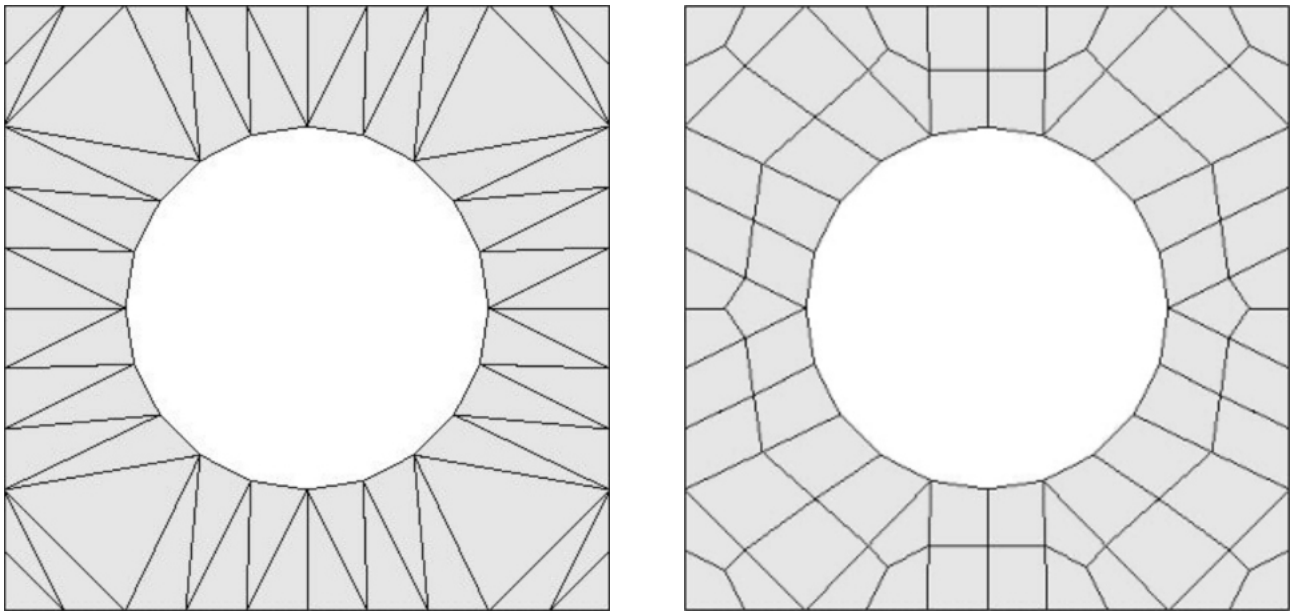

Figure 2 – Mode `refine_flag` = false (T3 and Q4) for the example 3.

Node smoothing

`node_smoothing_flag`. Default = `true`. Used in `MESH_MODE` and `REGULARIZE_MODE`.

This flag controls the node-smoothing scheme in the optimization step. 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`. Used in `MESH_MODE` and `REGULARIZE_MODE`.

This flag controls the node-inserting scheme in the optimization step.
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`. Used in `MESH_MODE` and `REGULARIZE_MODE`.

This flag controls the node-removing scheme in the optimization step.
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`. Used in `MESH_MODE` and `REGULARIZE_MODE`.

This flag controls the edge-swapping scheme in the optimization step.
Edge swapping 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).

Computation of the size-qualities histogram

`compute_Qh_flag`. Default = `false`. Used in all modes.

Before exiting the process, this flag tells the mesher to compute the histogram of the size quality of all the edges in the mesh. The users are rarely interested in this histogram, so the default state is false.

Pattern for structured meshes (CM2 TriaMesh Iso/Aniso only)

`structured_pattern`. Default = -1. Used in `MESH_MODE` only.

This option controls the way the generators does the structured (grid) meshes when possible (on rectangular-like domains). `structured_pattern` can take four values:

- -1: the triangular 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);

Note: The structured-mesh scheme is disabled by isolated edges, nodes and repulsive points (even located outside the domain).

Pattern for structured meshes (CM2 QuadMesh Iso/Aniso only)

`structured_flag`. Default = `true`. Used in `MESH_MODE` only.

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

- `true`: when possible, generates structured (grid-like) meshes (the default);
- `false`: the quad meshes are always done with the frontal-Delaunay algorithm[11];

Note: The structured-mesh scheme is disabled by isolated edges, nodes and repulsive points (even located outside the domain).

[11] The frontal-Delaunay algorithm may not generate grid-like meshes in all possible cases. Setting this flag enforces the grid-like generation (and is much faster) whenever such a structured mesh can be generated.

Multi-structured sub-domains

`multi_structured_flag`. Default = `false`. Used in `MESH_MODE` only.

Triggers a more complete search for structurable sub-domains than normally (at the price of a lower speed). With this flag on, several rectangular sub-domains can be meshed structurably whereas it is not always the case with this flag down.

Used only when `refine_flag` = `true` and `structured_flag` = `true` (**CM2 QuadMesh Iso/Aniso**), `structured_pattern` ≠ -1 (**CM2 TriaMesh Iso/Aniso**) and when there is no isolated node and no repulsive point.

Avoiding clamped edges (CM2 TriaMesh Iso/Aniso only)

`no_edge_with_nodes_below`. Default = 0. Used in `MESH_MODE` and `REGULARIZE_MODE`.

Forbids edges with two nodes below this node id.

To forbid edges between two hard nodes (nodes of the boundaries or embedded nodes) set this value to `data.pos.cols()` upon entry. To disable this control, set this value to 0.

This control can be limited by `nodes_limit`.

This setting has no effect when the optimization step is disabled (`optim_level` = 0) or when `node_inserting_flag` = `false`.

Limit on the number of nodes

`nodes_limit`. Default = `UINT_MAX`. Used in all modes.

When this limit is reached by the mesher[12], the `CM2_NODES_LIMIT_WARNING` is issued. The algorithm generates a mesh but the quality can be far from optimal. When the limit is so low that the mesher cannot even insert all the hard nodes, the `CM2_NODES_LIMIT_ERROR` is raised and the mesh is aborted.

Optimization level

`optim_level`. Integer between 0 and 10. Default = 3. Used in `MESH_MODE` and `REGULARIZE_MODE`.

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.

Target metric

`target_metric`. Double value (isotropic) or `DoubleSym2` (anisotropic). Default = 0. Used in `MESH_MODE` only.

Global element size inside the domain. The elements size tends toward this value as they get away from the hard (boundary) edges. This parameter is often used to reduce the total number of elements (when `target_metric` > default sizes based on boundary edge lengths) and to save FEA computation time without loosing much on element shape quality.

The `target_metric` may be smaller or bigger than the default sizes based on boundary edge lengths.

[12] This limit is not strict. When reached, the number of nodes actually in the final mesh may be slightly different from this limit (lesser or greater).

Max gradation

`max_gradation`. Double value greater than 0. Default = 0.5. Used in `MESH_MODE` only.

This parameter controls the gradation of the elements size from the boundary sizes to the size defined by `target_metric` inside the domain. A value close to 0 leads to a more progressive variation of mesh size (smoother).

Weight on shape quality

`shape_quality_weight`. Double value between 0 and 1. Default = 0.60. Used in `MESH_MODE` and `REGULARIZE_MODE`.

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 QuadMesh Iso/Aniso only)

`quadrangle_weight`. Double value between 0 and 1. Default = 0.70

Used in `MESH_MODE` and `REGULARIZE_MODE`.

This parameter controls the trade-off between a higher ratio of quads and a better mesh with more triangles.

- 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 (when a quad is better than two triangles);
- for values between 0.5 and 1, quadrangles are more and more used even if this lead to a lesser global quality of the mesh;
- with `quadrangle_weight` = 1, the minimum number of triangles are used to get a valid mesh (but may be of poor quality);

The default value (0.70) gives a significant preference to the quad/triangle ratio over the mesh quality.

Minimum quad quality (CM2 QuadMesh Iso/Aniso only)

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

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[13]. Set `min_Q4_angle_quality` = 1 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 (mostly triangles).

Upper bound on edges length

`length_upper_bound`. Double value greater than 0. Default = 1.414. Used in `MESH_MODE` only.

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).

[13] The angle quality of any rectangle equals to 1 (perfect) whereas the geometric quality is equal to 1 only for a square.

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).

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. The value is capped by the number of logical processors on the running CPU.

Display handler

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

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::triamesh_iso::mesher              my_mesher;
cm2::triamesh_iso::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_hdl 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 mesher to check for a stop signal. When the handler returns true, the mesher aborts its current step. If the interruption occurs early in the meshing stage - for instance in the front mesh step - the mesh is invalid, so it is cleared. From the refine step on however, the user can get a valid mesh upon exit, though probably of poor quality.

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::triamesh_iso::mesher              my_mesher;
cm2::triamesh_iso::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::triamesh_iso::mesher              my_mesher;
cm2::triamesh_iso::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.

```
enum basic_mode_type
{
   MESH_MODE,
   REGULARIZE_MODE,
   CONVEX_HULL_MODE
};
```

Table 10 – The `basic_mode_type` (all plane meshers).

```
struct settings_type
{
   basic_mode_type          basic_mode;
   bool                     strict_constraints_flag;
   int                      subdomains_forcing;
   bool                     refine_flag;
   bool                     node_smoothing_flag;
   bool                     node_inserting_flag;
   bool                     node_removing_flag;
   bool                     shell_remeshing_flag;
   bool                     compute_Qh_flag;
   int                      structured_pattern;
   bool                     multi_structured_flag;
   unsigned                 no_edge_with_nodes_below;
   unsigned                 nodes_limit;
   unsigned                 optim_level;
   double                   target_metric;
   double                   max_gradation;
   double                   shape_quality_weight;
   double                   length_upper_bound;
   unsigned                 num_threads;
   display_handler_type     display_hdl;
   interrupt_handler_type   interrupt_hdl;
   void*                    pass_thru;
};
```

Table 11 – The `settings_type` structure (CM2 TriaMesh Iso).

```
struct settings_type
{
   basic_mode_type          basic_mode;
   bool                     strict_constraints_flag;
   int                      subdomains_forcing;
   bool                     refine_flag;
   bool                     node_smoothing_flag;
   bool                     node_inserting_flag;
   bool                     node_removing_flag;
   bool                     shell_remeshing_flag;
   bool                     compute_Qh_flag;
   int                      structured_pattern;
   bool                     multi_structured_flag;
   unsigned                 no_edge_with_nodes_below;
   unsigned                 nodes_limit;
   unsigned                 optim_level;
   DoubleSym2               target_metric;
   double                   max_gradation;
   double                   shape_quality_weight;
   double                   length_upper_bound;
   unsigned                 num_threads;
   display_handler_type     display_hdl;
   interrupt_handler_type   interrupt_hdl;
   void*                    pass_thru;
};
```

Table 12 – The `settings_type` structure (CM2 TriaMesh Aniso).

```
struct settings_type
{
   basic_mode_type          basic_mode;
   bool                     strict_constraints_flag;
   int                      subdomains_forcing;
   bool                     all_quad_flag;
   bool                     refine_flag;
   bool                     node_smoothing_flag;
   bool                     node_inserting_flag;
   bool                     node_removing_flag;
   bool                     shell_remeshing_flag;
   bool                     compute_Qh_flag;
   bool                     structured_flag;
   bool                     multi_structured_flag;
   unsigned                 nodes_limit;
   unsigned                 optim_level;
   double                   target_metric;
   double                   max_gradation;
   double                   shape_quality_weight;
   double                   quadrangle_weight;
   double                   min_Q4_angle_quality;
   double                   length_upper_bound;
   unsigned                 num_threads;
   display_handler_type     display_hdl;
   interrupt_handler_type   interrupt_hdl;
   void*                    pass_thru;
};
```

Table 13 – The `settings_type` structure (CM2 QuadMesh Iso).

```
struct settings_type
{
   basic_mode_type          basic_mode;
   bool                     strict_constraints_flag;
   int                      subdomains_forcing;
   bool                     all_quad_flag;
   bool                     refine_flag;
   bool                     node_smoothing_flag;
   bool                     node_inserting_flag;
   bool                     node_removing_flag;
   bool                     shell_remeshing_flag;
   bool                     compute_Qh_flag;
   bool                     structured_flag;
   bool                     multi_structured_flag;
   unsigned                 nodes_limit;
   unsigned                 optim_level;
   DoubleSym2               target_metric;
   double                   max_gradation;
   double                   shape_quality_weight;
   double                   quadrangle_weight;
   double                   min_Q4_angle_quality;
   double                   length_upper_bound;
   unsigned                 num_threads;
   display_handler_type     display_hdl;
   interrupt_handler_type   interrupt_hdl;
   void*                    pass_thru;
};
```

Table 14 – The `settings_type` structure (CM2 QuadMesh Aniso).

Function members

The `settings_type` class is equipped 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 15 – The `settings_type` function members (all plane meshers).

4. General scheme of the generators

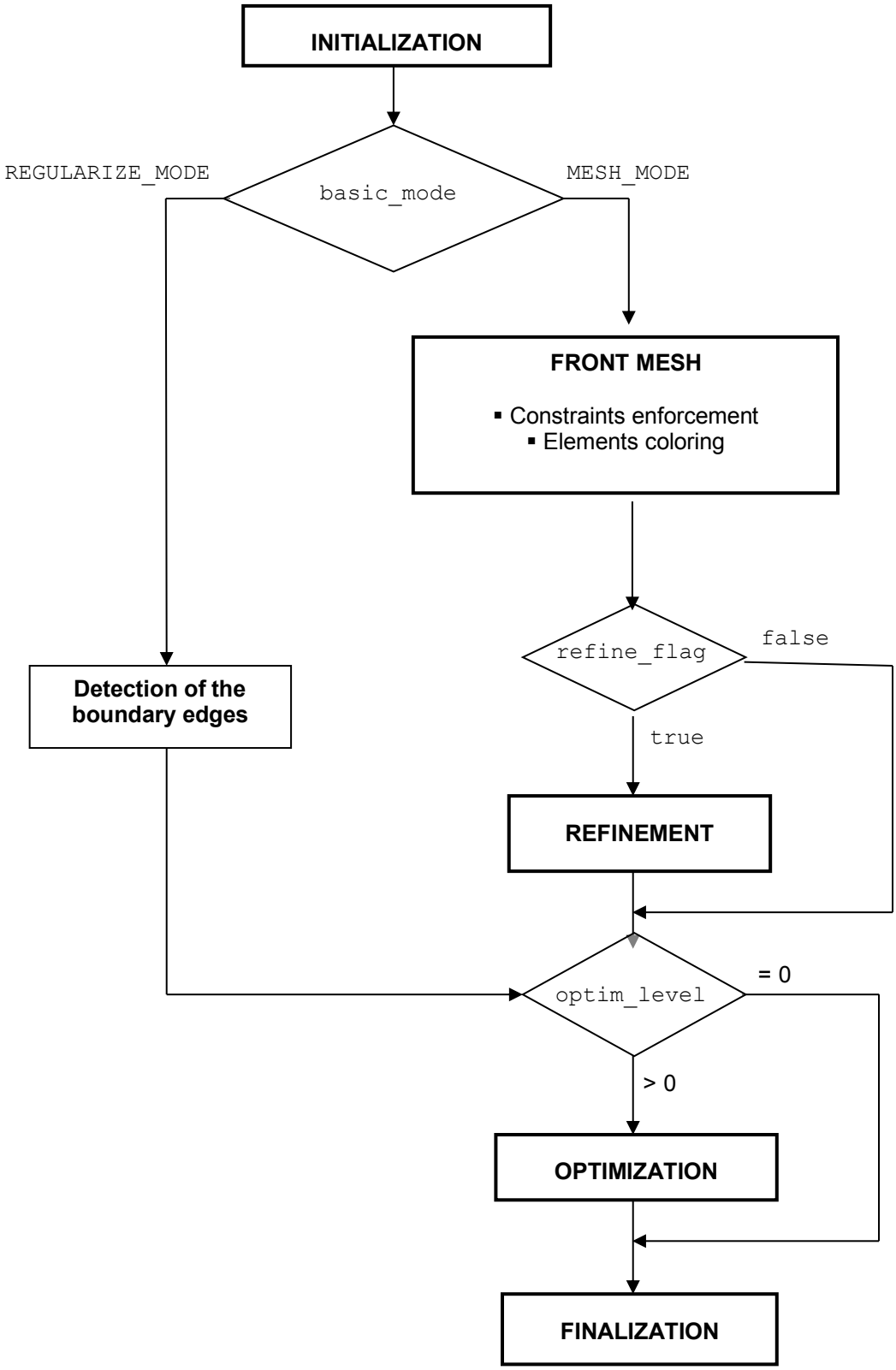

Figure 3 – General scheme of the mesh generators.

COMPUTING
OBJECTS

https://www.computing-objects.com

© Computing Objects SARL - 25 rue du Maréchal Foch, 78000 Versailles, France.

Limited Liability Company with a capital of 100 000 €.
Registered at Versailles RCS under SIRET number 422 791 038 00033.
EU VAT registration FR59422791038.