COMPUTING OBJECTS

CM2 MeshTools Core

Version 5.6

overview

Revision May 2025.

https://www.computing-objects.com

Forewords

This manual presents a mere listing of the functions and classes of the core libraries of the **CM2 MeshTools®** SDK.

For more information on each function or class, please refer to the HTML manual.

Table of contents

Forewords **2**

cm2::meshtools functions (203) **6**

Points/mesh translation 6

Points/mesh rotation 6

Points/mesh zoom 6

Points/mesh copying 6

Points/mesh merging 7

Nodes/mesh simplifying 7

Ancestors, neighbors and boundaries 8

Nodes -> elements graph, nodes -> nodes graph, boundaries -> elements mappings 8

Sub-domains 8

Jacobians / shape qualities 8

Lengths / edge qualities 9

Ids checking / retrieving / permutation 9

Centroids 10

Metrics generation / bounding / checking / transformation (isotropic & anisotropic) 10

Eigen values and Eigen pairs of metrics 10

NASTRAN input/output 10

FEMAP input/output 11

STL input/output 11

VTK output 11

VIZIR output 11

cm2::meshtools class **12**

node_detector 12

cm2::meshtools1d functions (59) **13**

Transformations / conversion 13

Extrude translate into an E2 mesh 13

Extrude rotate into an E2 mesh 13

CAD meshing on parametric curves. 14

Generates meshes along lines with internal specific sizes 14

Mesh splines 14
Mesh straight lines and broken lines 14
Remeshing of lines 15
Reorientation of E2 elements 15
Normals and curvatures computation routine along E2 meshes 15
Miscellaneous routines for E2 meshes 15
Anisotropic metrics generation routines along E2 meshes 16
Interpolation of metrics on E2 meshes 16
cm2::meshtools2d functions (97) 17
Boolean operations 17
Transformations / conversion 17
Extrude translate into T3 or Q4 mesh 17
Extrude rotate into T3 or Q4 mesh 18
Extrudes with both rotation and translation into a T3 or Q4 mesh 18
Extrudes normal into a T3 or Q4 mesh 18
Fields interpolation on 2-D meshes 19
Structured T3 or Q4 mesh (regular grid) 19
Pseudo-structured disk (quarter or full, T3 or Q4) 19
Pseudo-structured sphere (1/8 or full, T3 or Q4) 19
Structured surface meshes on parallelepipeds (regular grids) 20
CAD meshing on parametric surfaces. 20
Offsets 20
Special transformation for degenerated quadrangles 20
Area computation for T3 and Q4 meshes (planar or not) 21
Voronoi cells for T3 meshes (planar or not) 21
Angle computation in T3 and Q4 meshes (planar or not) 21
Normal computations on surface meshes (triangles and quadrangles) 21
Curvature computations on surface meshes (triangles only) 21
Anisotropic metrics generation routines on 2-D triangle meshes 21
Interpolation of isotropic metrics on 2-D meshes 21
Interpolation of anisotropic metrics on 2-D meshes 21
Element localization in 2-D and surface meshes 21
cm2::meshtools3d functions (48) 23

Boolean operations 23
Transformations / conversion 23
Extrude translate a 2-D mesh into a 3-D mesh 23
Extrude rotate a 2-D mesh into a 3-D mesh 23
Extrudes with both rotation and translation a 2-D mesh into a 3-D mesh 23
Extrudes a 2-D mesh along specific directions into a 3-D mesh 23
Extrudes a 2-D mesh along a line into a 3-D mesh 24
Interpolation of fields on 3-D tetrahedral meshes 24
Reorientation of 3-D elements 24
Special transformations of 3-D meshes 24
Volume computation 24
Voronoi cells for tetrahedron meshes 25
Angle computation 25
Interpolation of isotropic metrics on 3-D meshes 25
Interpolation of anisotropic metrics on 3-D meshes 25
Element localization in 3-D meshes 25

cm2::meshtools functions (203)

Points/mesh translation

```
// Translates all nodes.
int translate (DoubleMat &pos, const DoubleVec2 &T);

// Translates a batch of nodes.
int translate (DoubleMat &pos, const DoubleVec2 &T, const UIntVec &indices);

// Translates the nodes of a mesh.
int translate (DoubleMat &pos, const DoubleVec2 &T, const UIntMat &connect);

// Translates all nodes.
int translate (DoubleMat &pos, const DoubleVec3 &T);

// Translates a batch of nodes.
int translate (DoubleMat &pos, const DoubleVec3 &T, const UIntVec &indices);

// Translates the nodes of a mesh.
int translate (DoubleMat &pos, const DoubleVec3 &T, const UIntMat &connect);
```

Points/mesh rotation

```
// Rotates all nodes.
int rotate (DoubleMat &pos, const DoubleVec2 &C, double R);

// Rotates a batch of nodes.
int rotate (DoubleMat &pos, const DoubleVec2 &C, double R, const UIntVec &indices);

// Rotates the nodes of a mesh.
int rotate (DoubleMat &pos, const DoubleVec2 &C, double R, const UIntMat &connect);

// Rotates all nodes.
int rotate (DoubleMat &pos, const DoubleVec3 &C, const DoubleVec3 &R);

// Rotates a batch of nodes.
int rotate (DoubleMat &pos, const DoubleVec3 &C, const DoubleVec3 &R, const UIntVec &indices);

// Rotates the nodes of a mesh.
int rotate (DoubleMat &pos, const DoubleVec3 &C, const DoubleVec3 &R, const UIntMat &connect);
```

Points/mesh zoom

```
// Zooms all nodes.
int zoom (DoubleMat &pos, const DoubleVec2 &C, double Zv);

// Zooms a specific batch of nodes.
int zoom (DoubleMat &pos, const DoubleVec2 &C, double Zv, const UIntVec &indices);

// Zooms the nodes of a mesh.
int zoom (DoubleMat &pos, const DoubleVec2 &C, double Zv, const UIntMat &connect);

// Zooms all nodes.
int zoom (DoubleMat &pos, const DoubleVec3 &C, double Zv);

// Zooms a specific batch of nodes.
int zoom (DoubleMat &pos, const DoubleVec3 &C, double Zv, const UIntVec &indices);

// Zooms the nodes of a mesh.
int zoom (DoubleMat &pos, const DoubleVec3 &C, double Zv, const UIntMat &connect);
```

Points/mesh copying

```
// Copies a mesh (nodes and connectivity);
int copy_mesh (DoubleMat &pos, UIntMat &connect_tar, const UIntMat &connect_src);

// Copies and symmetrises a mesh (nodes and connectivity).
int copy_sym_mesh (DoubleMat &pos, UIntMat &connect_tar, const DoubleVec2 &C, const DoubleVec2 &D,
const UIntMat &connect_src);

// Copies and symmetrises a mesh (nodes and connectivity).
int copy_sym_mesh (DoubleMat &pos, UIntMat &connect_tar, const DoubleVec3 &C, const DoubleVec3 &D,
const UIntMat &connect_src);

// Duplicates a mesh N-times (nodes and connectivity), translate and rotate each copy.
int copy_mesh (DoubleMat &pos, UIntMat &connect_tar, const DoubleVec2 &T, const DoubleVec2 &C, double
R, unsigned N, const UIntMat &connect_src);

// Duplicates a mesh (nodes and connectivity).
int copy_mesh (DoubleMat &pos, UIntMat &connect_tar, const DoubleVec3 &T, const DoubleVec3 &C, const
DoubleVec3 &R, unsigned N, const UIntMat &connect_src);
```

Points/mesh merging

```
// Merges points that are closer to each other than a given distance.
int merge (DoubleMat &pos, UIntVec &new_node_IDs, double distance);

// Same as above without the new_node_IDs parameter.
int merge (DoubleMat &pos, double distance);

// Merges points that are closer to each other than a given distance.
int merge (const DoubleMat &pos, const UIntVec &src_node_IDs, const UIntVec &tgt_node_IDs, UIntVec
&new_node_IDs, double distance);

// Merges points that are closer to each other than a given distance.
int merge (const DoubleMat &pos, const UIntVec &node_IDs, UIntVec &new_node_IDs, double distance);

// Merges nodes in a mesh that are closer to each other than a given distance.
int merge (const DoubleMat &pos, UIntMat &connect, UIntVec &new_node_IDs, double distance, unsigned
merge_type);

// Same as above without the new_node_IDs parameter.
int merge (const DoubleMat &pos, UIntMat &connect, double distance, unsigned merge_type);

// Merges nodes in two meshes that are closer to a given distance.
int merge (const DoubleMat &pos, UIntMat &connect1, UIntMat &connect2, UIntVec &new_node_IDs, double
distance, unsigned merge_type=0);

// Same as above without the new_node_IDs parameter.
int merge (const DoubleMat &pos, UIntMat &connect1, UIntMat &connect2, double distance, unsigned merge_
type=0);

// Merges nodes in several meshes that are closer to a given distance.
int merge (const DoubleMat &pos, UIntMat *connect_beg, UIntMat *connect_end, UIntVec &new_node_IDs,
double distance, unsigned merge_type=0);

// Same as above without the new_node_IDs parameter.
int merge (const DoubleMat &pos, UIntMat *connect_beg, UIntMat *connect_end, double distance, unsigned
merge_type=0);
```

Nodes/mesh simplifying

```
// Gets the unique values in a vector of indices.
int unique_indices (UIntVec &indices_tar, const UIntVec &indices_src, const UIntVec remove_indices=cm2:
:UIntVec());

// Gets the unique values in a matrix of indices.
int unique_indices (UIntVec &indices_tar, const UIntMat &connect_src, const UIntVec remove_indices=cm2:
:UIntVec());

// Gets the unique values in several matrices of indices.
int unique_indices (UIntVec &indices_tar, const UIntMat* connect_src_beg, const UIntMat* connect_src_
end, const UIntVec remove_indices=cm2::UIntVec());

// Counts the nodes referenced in a connectivity matrix (except CM2_NONE), with no duplicates.
int count_unique_indices (const UIntMat &connect);

// Counts the nodes referenced in several matrices (except CM2_NONE), with no duplicates.
int count_unique_indices (const UIntMat *connect_beg, const UIntMat *connect_end);

// Eliminates the unreferenced nodes in several meshes.
int simplify (DoubleMat &pos, UIntMat *connect_beg, UIntMat *connect_end, UIntVec &new_node_IDs,
unsigned N0=0);

// Eliminates the unreferenced nodes in a mesh.
int simplify (DoubleMat &pos, UIntMat &connect, UIntVec &new_node_IDs, unsigned N0=0);

// Eliminates the unreferenced nodes in a mesh.
int simplify (DoubleMat &pos, UIntMat &connect, unsigned N0=0);

// Eliminates the unreferenced nodes simultaneously in two meshes.
int simplify (DoubleMat &pos, UIntMat &connect0, UIntMat &connect1, unsigned N0=0);

// Eliminates the duplicate elements (keep only one).
int unique_elements (UIntMat &connect, cm2::element_type FE_type, bool orient_sensitive_flag);

// Eliminates the elements with multiple identical node IDs.
int remove_degenerated (UIntMat &connect, bool aggress_flag=true);

// Eliminates the elements with multiple identical node IDs.
int remove_degenerated (UIntMat &connect, UIntVec &new_element_IDs, bool aggress_flag=true);

// Clipping of a mesh (ecorche).
int clip (const DoubleMat &pos, const UIntMat &connect_src, const DoubleVec3 &P0, const DoubleVec3 &D,
UIntMat &connect_tar);
```

Ancestors, neighbors and boundaries

```
// For each node in a mesh finds an element connected to it.
int get_ancestors (const UIntMat &connect, UIntVec &ancestors);

// Computes the neighbors matrix of a mesh given by a connectivity matrix.
int get_neighbors (const UIntMat &connect, cm2::element_type FE_type, bool accept_multiple_neighbors,
UIntMat &neighbors);

// Computes the boundaries of a mesh (boundaries connected to only one element).
int get_mesh_boundaries (const UIntMat &connect, const UIntMat &neighbors, cm2::element_type FE_type,
UIntMat &boundaries);

// Computes the boundaries of a mesh (boundaries connected to only one element).
int get_mesh_bounding_elements (const UIntMat &connect, const UIntMat &neighbors, cm2::element_type FE_
type, UIntVec &KBs, unsigned &nbr_elfe_boundaries);

// Computes the boundaries of all elements in a mesh.
int get_element_boundaries (const UIntMat &connect, cm2::element_type FE_type, UIntMat &boundaries);
```

Nodes -> elements graph, nodes -> nodes graph, boundaries -> elements mappings

```
// Computes the elements connected to each node.
int get_connected_elements (const UIntMat &connect, UIntVec &xadj, UIntVec &adjncy);

// Computes the nodes connected to each node (nodal graph).
int get_node_graph (const UIntMat &connect, UIntVec &xadj, UIntVec &adjncy, cm2::element_type FE_type);

// Retrieves the elements sharing at least one specified boundary.
int get_elements_on_boundaries (const UIntMat &connectB, const UIntMat &connectM, const UIntMat
&neighbors, const UIntVec &ancestors, cm2::element_type FE_type, UIntVec &KBs, unsigned &nbr_elfe_
boundaries);
```

Sub-domains

```
// Finds the groups of connected elements.
int get_subdomains (const UIntMat &neighbors, UIntVec &ordered_elements, UIntVec &begin_parts);

// Reorders the elements in a mesh.
int get_subdomains (UIntMat &connect, UIntMat &neighbors, UIntVec &begin_parts);

// Finds the boundaries between elements of different colors.
int get_colors_boundaries (const UIntMat &connect, const UIntMat &neighbors, const UIntVec &colors,
cm2::element_type FE_type, bool sym_flag, bool colored_only_flag, UIntMat &boundaries);

// Reorders a mixed connectivity matrix according to the number of nodes of the elements.
int sort_elfes (UIntMat &connect, UIntVec &nbr_nodes, UIntVec &past_group, UIntVec &permutation_IDs);

// Reorders a mixed connectivity matrix according to the number of nodes of the elements, with
neighbors matrix.
int sort_elfes (UIntMat &connect, UIntMat &neighbors, UIntVec &nbr_nodes, UIntVec &past_group, UIntVec
&permutation_IDs);
```

Jacobians / shape qualities

```
// Computes the jacobians of all elements of a mesh.
int jacobians (const DoubleMat &pos, const UIntMat &connect, cm2::element_type FE_type, DoubleVec &J);

// The elemnent Ids with at least one negative or null jacobian in the quadrature.
int invalid_jacobians (const DoubleMat &pos, const UIntMat &connect, cm2::element_type FE_type, UIntVec
&invalid_IDs);

// Computes the shape qualities of all elements of a mesh.
int shape_qualities (const DoubleMat &pos, const UIntMat &connect, cm2::element_type FE_type, DoubleVec
&Qs);

// Finds the element with minimum shape quality in a mesh.
int min_quality (const DoubleMat &pos, const UIntMat &connect, cm2::element_type FE_type, double &qmin,
unsigned &jmin);

// Adds in an histogram the shape qualities of all elements of a mesh (very simple function provided
for convenience).
int shape_qualities (const DoubleMat &pos, const UIntMat &connect, cm2::element_type FE_type, misc:
:histogram &histo_Qs);
```

Lengths / edge qualities

```
// Gets all the edges (unique) in a mesh.
int copy_edges (const UIntMat &connectM, cm2::element_type FE_type, UIntMat &connectE, const UintMat
&excludedE = UintMat());

// Gets all the edges (unique) in several meshes.
int copy_edges (const UIntMat *connect_beg, const UIntMat *connect_end, const IntVec &FE_types, UIntMat
&connectE, const UintMat &excludedE = UintMat());

// Computes the length measure of a segment with two size values at its summits.
int edge_quality (double L, double h0, double h1, double &qh);

// Adds in an histogram the qualities of a batch of edges (isotropic version).
int edge_qualities (const DoubleMat &pos, const UIntMat &connectM, cm2::element_type FE_type, const
DoubleVec &metrics, misc::histogram &histo_Qh, const UIntMat &excludedE=UIntMat());

// Adds in an histogram the qualities of a batch of edges (anisotropic version).
int edge_qualities (const DoubleMat &pos, const UIntMat &connectM, cm2::element_type FE_type, const
DoubleMat &metrics, misc::histogram &histo_Qh, const UIntMat &excludedE=UIntMat());

// Adds in an histogram the qualities of a batch of edges (isotropic version).
int edge_qualities (const DoubleMat &pos, const UIntMat *connect_beg, const UIntMat *connect_end, const
IntVec &FE_types, const DoubleVec &metrics, misc::histogram &histo_Qh, const UIntMat
&excludedE=UIntMat());

// Adds in an histogram the qualities of a batch of edges (anisotropic version).
int edge_qualities (const DoubleMat &pos, const UIntMat *connect_beg, const UIntMat *connect_end, const
IntVec &FE_types, const DoubleMat &metrics, misc::histogram &histo_Qh, const UIntMat
&excludedE=UIntMat());

// Computes the H-shock in a mesh (isotropic version).
int H_shock (const DoubleMat &pos, const UIntMat &connectE, const DoubleVec &metrics, DoubleVec
&Hshock);

// Computes the H-shock in a mesh (anisotropic version).
int H_shock (const DoubleMat &pos, const UIntMat &connectE, const DoubleMat &metrics, DoubleVec
&Hshock);
```

Ids checking / retrieving / permutation

```
// The highest ID different than CM2_NONE.
unsigned max_ID (const UIntVec &indices);

// The highest ID different than CM2_NONE.
unsigned max_ID (const UIntMat &connect);

// Checks that all the input indices are lesser than a given value.
bool check_IDs (const UIntVec &indices, unsigned lesser_than);

// Checks that all the indices in a connectivity matrix are lesser than a given value.
bool check_IDs (const UIntMat &connect, unsigned lesser_than);

// Checks that a connectivity matrix is valid and that all its indices are lesser than a given value.
bool check_IDs (const UIntMat &connect, cm2::element_type FE_type, unsigned lesser_than);

// Finds positions of values in two vectors.
int get_IDs (const UIntVec &V0, const UIntVec &V1, UIntVec &IDs, UIntVec &inv_IDs);

// Finds positions of columns in two matrices.
int get_IDs (const UIntMat &M0, const UIntMat &M1, UIntVec &IDs, UIntVec &inv_IDs, bool strict_compare_
flag);

// Shifts indices in a vector (for instance before appending a new mesh to an old one).
int shift_indices (UIntVec &IDs, int shift);

// Shifts indices in a connectivity matrix (for instance before appending a new mesh to an old one).
int shift_indices (UIntMat &connect, int shift, cm2::element_type FE_type);

// Changes the values in a 1-D array.
void change_indices (UIntVec &vals, const UIntVec &new_vals);

// Changes the values in a 2-D array.
void change_indices (UIntMat &vals, const UIntVec &new_vals);

// Permutes the values in a 1-D array (UintVec version).
int permutation (UIntVec &vals, const UIntVec &new_indices);

// Permutes the values in a 1-D array (DoubleVec version).
int permutation (DoubleVec &vals, const UIntVec &new_indices);

// Gets the order of elements according to their values.
int get_sorted_order (const UIntVec &values, UIntVec &order, UIntVec &xOrder);
```

Bounding box

```
// Enlarges a 2-D bounding box to enclose a set of points.
int inc_bounding_box (const DoubleMat &pos, DoubleVec2 &minBox, DoubleVec2 &maxBox);

// Enlarges a 2-D bounding box to enclose a set of points.
int inc_bounding_box (const DoubleMat &pos, const UIntVec &indices, DoubleVec2 &minBox, DoubleVec2
&maxBox);

// Enlarges a 2-D bounding box to enclose a mesh.
int inc_bounding_box (const DoubleMat &pos, const UIntMat &connect, DoubleVec2 &minBox, DoubleVec2
&maxBox);

// Enlarges a 3-D bounding box to enclose a set of points.
int inc_bounding_box (const DoubleMat &pos, DoubleVec3 &minBox, DoubleVec3 &maxBox);

// Enlarges a 3-D bounding box to enclose a set of points.
int inc_bounding_box (const DoubleMat &pos, const UIntVec &indices, DoubleVec3 &minBox, DoubleVec3
&maxBox);

// Enlarges a 3-D bounding box to enclose a mesh.
int inc_bounding_box (const DoubleMat &pos, const UIntMat &connect, DoubleVec3 &minBox, DoubleVec3
&maxBox);
```

Centroids

```
// The centroid of all elements of a mesh.
int centroids (const DoubleMat &pos, const UIntMat &connect, DoubleMat &PC, cm2::element_type FE_type);
```

Metrics generation / bounding / checking / transformation (isotropic & anisotropic)

```
// Generates isotropic metrics on the nodes of a given mesh (1-D, 2-D or 3-D).
int metrics_gen_iso (const DoubleMat &pos, const UIntMat &connectM, cm2::element_type FE_type,
DoubleVec &H, double w);

...
```

Eigen values and Eigen pairs of metrics

```
// The Eigen values (increasing order) of a 2x2 symmetric matrix.
void eigen_values (const DoubleSym2 &A, DoubleVec2 &eigenvalues);

// The Eigen values (increasing order) of a 3x3 symmetric definitive positive matrix.
void eigen_values (const DoubleSym3 &A, DoubleVec3 &eigenvalues);

// The Eigen values (increasing order) and Eigen vectors of a 2x2 symmetric matrix.
bool eigen_pairs (const DoubleSym2 &A, DoubleVec2 &eigenvalues, DoubleMat2x2 &eigenvectors);

// The Eigen values (increasing order) and eigen vectors of a 3x3 symmetric definitive positive matrix.
bool eigen_pairs (const DoubleSym3 &A, DoubleVec3 &eigenvalues, DoubleMat3x3 &eigenvectors);

// The Eigen pairs (Eigen vectors and Eigen values) of the matrix A-1 * B.
bool eigen_pairs (const DoubleSym2 &A, const DoubleSym2 &B, DoubleVec2 &eigenvalues, DoubleMat2x2
&eigenvectors);

// The Eigen pairs (Eigen vectors and Eigen values) of the matrix A-1 * B.
bool eigen_pairs (const DoubleSym3 &A, const DoubleSym3 &B, DoubleVec3 &eigenvalues, DoubleMat3x3
&eigenvectors);
```

NASTRAN input/output

```
// NASTRAN input of a mesh.
int NASTRAN_input (FILE *file, DoubleMat &pos, UIntMat &connect, cm2::element_type &FE_type);

// NASTRAN input of a mesh (two types of element).
int NASTRAN_input (FILE *file, DoubleMat &pos, UIntMat &connect0, UIntMat &connect1, cm2::element_type
&FE_type0, cm2::element_type &FE_type1);

// NASTRAN input of a mesh (many types of element).
int NASTRAN_input (FILE *file, DoubleMat &pos, UIntMat &connect, UIntVec &xConnect, IntVec &fe_types,
UIntVec &refs);

// NASTRAN input of a mesh (const char* version).
int NASTRAN_input (const char *filename, DoubleMat &pos, UIntMat &connect, cm2::element_type &FE_type);

// NASTRAN output of a mesh (single type of element).
int NASTRAN_output (FILE *file, const DoubleMat &pos, const UIntMat &connect, cm2::element_type FE_
type);

// NASTRAN output of a mesh (two types of element).
int NASTRAN_output (FILE *file, const DoubleMat &pos, const UIntMat &connect0, const UIntMat &connect1,
```

```
cm2::element_type FE_type0, cm2::element_type FE_type1);

// NASTRAN output of a mesh (many types of element).
int NASTRAN_output (FILE *file, const DoubleMat &pos, const UIntMat &connect, const UIntVec &xConnect,
const IntVec &fe_types, const UIntVec &refs);

// NASTRAN output of a mesh (const char* version, single type of element).
int NASTRAN_output (const char *filename, const DoubleMat &pos, const UIntMat &connect, cm2::element_
type FE_type);

...
```

FEMAP input/output

```
// FEMAP input of a mesh (single type of element).
int NASTRAN_input (FILE *file, DoubleMat &pos, UIntMat &connect, cm2::element_type &FE_type);

// FEMAP  output of a mesh (single type of element).
int NASTRAN_output (FILE *file, const DoubleMat &pos, const UIntMat &connect, cm2::element_type FE_
type);

...
```

STL input/output

```
// STL input of a triangle mesh.
int STL_input (FILE *file, DoubleMat &pos, UIntMat &connect);

// STL output of a triangle mesh.
int STL_output (FILE *file, const DoubleMat &pos, const UIntMat &connect, bool ASCII_flag=true);

...
```

Wavefront OBJ input/output

```
// Wavefront OBJ input of a mesh.
int WavefrontOBJ_input (const char *filename, DoubleMat &pos, DoubleMat &normals, DoubleMat &textures,
UIntMat &connectT, UIntMat &connectQ);

// Wavefront obj output of a mesh.
int WavefrontOBJ_output (const char *filename, const DoubleMat &pos, const UIntMat &connect, cm2:
:element_type FE_type);

...
```

VTK output

```
// VTK output of a mesh (format version 2.0).
int vtk_output (const char *filename, const DoubleMat &pos, const UIntMat &connect, cm2::element_type
FE_type);

...
```

VIZIR output

```
// VIZIR output of a mesh (INRIA's ASCII format).
int vizir_output (const char *filename, const DoubleMat &pos, const UIntMat &connect, cm2::element_type
FE_type);

// VIZIR output of solutions (INRIA's ASCII format).
int vizir_sols (const char *filename, unsigned dim, const UIntVec& sol_types, const DoubleMat& M);

// VIZIR output of a scalar solution (INRIA's ASCII format).
int vizir_scalar (const char *filename, unsigned dim, const DoubleVec& V);

// VIZIR output of a vector solution (INRIA's ASCII format).
int vizir_vector (const char *filename, unsigned dim, const DoubleMat& M);

// VIZIR output of a tennor solution (INRIA's ASCII format).
int vizir_tensor (const char *filename, unsigned dim, const DoubleMat& M);
```

cm2::meshtools class

node_detector

```
node_detector
{
...

// Clears the bisection tree and reinitializes it with new points.
int reinit (const DoubleMat &pos);

// Clears the bisection tree and reinitializes it with new points.
int reinit (const DoubleMat &pos, const UIntVec &indices);

// Clears the bisection tree and reinitialize it with new nodes.
int reinit (const DoubleMat &pos, const UIntMat &connect);

// Clears the bisection tree and reinitializes it with new points.
int reinit (const DoubleMat &pos, const UIntMat *connect_beg, const UIntMat *connect_end);

// Inserts new nodes in the bisection tree.
unsigned    insert (const UIntVec &indices);

// Inserts new nodes in the bisection tree.
unsigned insert (const UIntMat &connect);

// A node close to a given point (not always the closest).
unsigned find_close (const DoubleVec2 &P) const;

// A node close to a given point (not always the closest).
unsigned find_closest (const DoubleVec3 &P) const;

// A node close to a given node (not always the closest).
unsigned find_closest (unsigned N) const;

// The closest node to a given point.
unsigned find_closest (const DoubleVec2 &P) const;

// The closest node to a given point.
unsigned find_closest (const DoubleVec3 &P) const;

// The closest node to a given node.
unsigned find_closest (unsigned N) const;

// Gets the nodes located in a given 2-D box.
void find_in_box (const DoubleVec2 &MinBox, const DoubleVec2 &MaxBox, UIntVec &nodes) const;

// Gets the nodes located in a given 3-D box.
void find_in_box (const DoubleVec3 &MinBox, const DoubleVec3 &MaxBox, UIntVec &nodes) const;

};
```

cm2::meshtools1d functions (59)

Transformations / conversion

```
// Transforms a series of node indices into a mesh of E2 elements.
int indices_to_connectE2 (const UIntVec &indices, UIntMat &connectE2);

// Transforms a series of node indices into a mesh of EDGE3 elements.
int indices_to_connectE3 (const UIntVec &indices, UIntMat &connectE3);

// Transforms a series of node indices into a mesh of EDGE4' elements.
int indices_to_connectE4 (const UIntVec &indices, UIntMat &connectE4);

// Converts E2 elements into higher degree elements (E3, E4 and over), creating new high-order nodes.
int convert_into_high_order (DoubleMat &pos, const UIntMat &connectL, unsigned Ne, UIntMat &connectQ);

// Converts in-place E2 elements into high-order elements (EDGE3 and over) creating new high-order
nodes.
int convert_into_high_order (DoubleMat &pos, UIntMat &connectM, unsigned Ne);

// Converts in-place E2 elements into EDGE3 elements creating new high-order nodes.
int convert_into_quadratic (DoubleMat &pos, UIntMat &connectM);

// Converts in-place E2 elements into EDGE4 elements creating new high-order nodes.
int convert_into_cubic (DoubleMat &pos, UIntMat &connectM);

// Converts in-place high-order elements (EDGE3 or higher) into linear elements.
int convert_into_linear (UIntMat &connectM);

// Transforms a 3-node edge connectivity matrix into a 2-node edge connectivity matrix (doubled).
int split_into_linear (const UIntMat &connectE3, UIntMat &connectE2);
```

Extrude translate into an E2 mesh

```
// Creates a straight line mesh from a node and a translation (2-D version).
int extrude_translate (DoubleMat &pos, unsigned N0, const DoubleVec2 &T, unsigned NE, UIntVec
&indices);

// Creates a straight line mesh from a node and a translation (3-D version).
int extrude_translate (DoubleMat &pos, unsigned N0, const DoubleVec3 &T, unsigned NE, UIntVec
&indices);

// Creates a straight line mesh from a point and a translation (2-D version).
int extrude_translate (DoubleMat &pos, const DoubleVec2 &P0, const DoubleVec2 &T, unsigned NE, UIntVec
&indices);

// Creates a straight line mesh from a point and a translation (3-D version).
int extrude_translate (DoubleMat &pos, const DoubleVec3 &P0, const DoubleVec3 &T, unsigned NE, UIntVec
&indices);

// Creates a straight line mesh from a node and a translation with specific sizes (2-D version).
int extrude_translate (DoubleMat &pos, unsigned N0, const DoubleVec2 &T, double h0, double h1, bool
force_even, UIntVec &indices);

// Creates a straight line mesh from a node and a translation with specific sizes (3-D version).
int extrude_translate (DoubleMat &pos, unsigned N0, const DoubleVec3 &T, double h0, double h1, bool
force_even, UIntVec &indices);

// Creates a straight line mesh from a point and a translation with specific sizes (2-D version).
int extrude_translate (DoubleMat &pos, const DoubleVec2 &P0, const DoubleVec2 &T, double h0, double h1,
bool force_even, UIntVec &indices);

// Creates a straight line mesh from a point and a translation with specific sizes (3-D version).
int extrude_translate (DoubleMat &pos, const DoubleVec3 &P0, const DoubleVec3 &T, double h0, double h1,
bool force_even, UIntVec &indices);
```

Extrude rotate into an E2 mesh

```
// Creates an arc mesh from a node and a rotation.
int extrude_rotate (DoubleMat &pos, unsigned N0, const DoubleVec3 &C, const DoubleVec3 &R, unsigned NE,
UIntVec &indices);

// Creates an arc mesh from a node and a rotation in the XY plane (2-D version).
int extrude_rotate (DoubleMat &pos, unsigned N0, const DoubleVec2 &C, double Rz, unsigned NE, UIntVec
&indices);

// Creates an arc mesh from a point and a rotation (3-D version).
int extrude_rotate (DoubleMat &pos, const DoubleVec3 &P0, const DoubleVec3 &C, const DoubleVec3 &R,
unsigned NE, UIntVec &indices);

// Creates an arc mesh from a point and a rotation (2-D version).
int extrude_rotate (DoubleMat &pos, const DoubleVec2 &P0, const DoubleVec2 &C, double Rz, unsigned NE,
UIntVec &indices);
```

```
// Creates an arc mesh from a node and a rotation with specific sizes (3-D version).
int extrude_rotate (DoubleMat &pos, unsigned N0, const DoubleVec3 &C, const DoubleVec3 &R, double h0,
double h1, bool force_even, UIntVec &indices);

// Creates an arc mesh from a node and a rotation with specific sizes (2-D version).
int extrude_rotate (DoubleMat &pos, unsigned N0, const DoubleVec2 &C, double Rz, double h0, double h1,
bool force_even, UIntVec &indices);

// Creates an arc mesh from a point and a rotation with specific sizes (3D version).
int extrude_rotate (DoubleMat &pos, const DoubleVec3 &P0, const DoubleVec3 &C, const DoubleVec3 &R,
double h0, double h1, bool force_even, UIntVec &indices);

// Creates an arc mesh from a point and a rotation with specific sizes (2-D version).
int extrude_rotate (DoubleMat &pos, const DoubleVec2 &P0, const DoubleVec2 &C, double Rz, double h0,
double h1, bool force_even, UIntVec &indices);
```

CAD meshing on parametric curves.

```
// Meshes a parametric 3-D curve (isotropic version).
template <class Curve>
int mesh_curve_param (const Curve &C, DoubleMat &pos, const UIntVec &nodes, DoubleVec &pos1D, DoubleVec
&H, UIntMat &connectE, double target_h, bool force_even, unsigned min_n, unsigned max_n, double max_
chordal_error, double min_h, unsigned chordal_control_type, double max_gradation, unsigned high_order_
type=0, unsigned high_order_mode=2, unsigned max_bgm_remeshings=4, bool update_metrics_flag=true);

// Meshes a parametric 3-D curve (anisotropic version).
template <class Curve>
int mesh_curve_param (const Curve &C, DoubleMat &pos, const UIntVec &nodes, DoubleVec &pos1D, DoubleMat
&metrics, UIntMat &connectE, double target_h, bool force_even, unsigned min_n, unsigned max_n, double
max_chordal_error, double min_h, unsigned chordal_control_type, double max_gradation, unsigned high_
order_type=0, unsigned high_order_mode=1, unsigned max_bgm_remeshings=4, bool update_metrics_
flag=true);
```

Generates meshes along lines with internal specific sizes

```
// Creates a mesh along a line defined by several geometric points and isotropic metrics.
int mesh_line (DoubleMat &pos, const UIntVec &geo_nodes, const DoubleVec &metricsG, bool force_even,
unsigned min_n, unsigned max_n, double max_chordal_error, double min_h, UIntVec &indices, DoubleVec
&Us, DoubleVec &metrics);

// Creates a mesh along a line defined by several geometric points and isotropic metrics.
int mesh_line (DoubleMat &pos, const UIntVec &geo_nodes, const DoubleVec &UG, const DoubleVec
&metricsG, bool force_even, unsigned min_n, unsigned max_n, double max_chordal_error, double min_h,
UIntVec &indices, DoubleVec &Us, DoubleVec &metrics);

// Creates a mesh along a line defined by several geometric points and anisotropic metrics.
int mesh_line (DoubleMat &pos, const UIntVec &geo_nodes, const DoubleMat &metricsG, bool force_even,
unsigned min_n, unsigned max_n, double max_chordal_error, double min_h, UIntVec &indices, DoubleVec
&Us, DoubleMat &metrics);

// Creates a mesh along a line defined by several geometric points, with an anisotropic metric given at
some specific soft nodes.
int mesh_line (DoubleMat &pos, const UIntVec &geo_nodes, const DoubleVec &UG, const DoubleMat
&metricsG, bool force_even, unsigned min_n, unsigned max_n, double max_chordal_error, double min_h,
UIntVec &indices, DoubleVec &Us, DoubleMat &metrics);
```

Mesh splines

```
// Computes a point's coordinates on a spline (2-D version).
int spline_point (const DoubleMat &pos, unsigned ia, unsigned ib, unsigned ic, unsigned id, double s,
DoubleVec2 &Ps);

// Computes a point's coordinates on a spline (3-D version).
int spline_point (const DoubleMat &pos, unsigned ia, unsigned ib, unsigned ic, unsigned id, double s,
DoubleVec3 &Ps);

// Creates a mesh on a poly-spline with specific isotropic sizes defined at the spline points.
int mesh_spline (DoubleMat &pos, const UIntVec &spline_nodes, const DoubleVec &spline_metrics, bool
force_even, unsigned min_n, unsigned max_n, double max_chordal_err, double min_h, double target_metric,
double max_gradation, UIntVec &indices, DoubleVec &Us, DoubleVec &metrics, double &G_distance);

// Creates a mesh on a poly-spline with specific anisotropic sizes.
int mesh_spline (DoubleMat &pos, const UIntVec &spline_nodes, const DoubleMat &spline_metrics, bool
force_even, unsigned min_n, unsigned max_n, double max_chordal_err, double min_h, double target_metric,
double max_gradation, UIntVec &indices, DoubleVec &Us, DoubleMat &metrics, double &G_distance);
```

Mesh straight lines and broken lines

```
// Creates a straight mesh between two nodes.
int mesh_straight (DoubleMat &pos, unsigned N0, unsigned N1, unsigned NE, UIntVec &indices);
```

```
// Creates a straight mesh between three nodes.
int mesh_straight (DoubleMat &pos, unsigned N0, unsigned N1, unsigned N2, unsigned NE, UIntVec
&indices);

// Creates a straight mesh between two nodes with specific sizes.
int mesh_straight (DoubleMat &pos, unsigned N0, unsigned N1, double h0, double h1, bool force_even,
UIntVec &indices);

// Creates a broken-line mesh between some imposed nodes (hard nodes).
int mesh_straight (DoubleMat &pos, const UIntVec &hard_nodes, unsigned NE, bool force_even, UIntVec
&indices);

// Creates a broken-line mesh between some imposed nodes (hard nodes) with specific (isotropic) sizes.
int mesh_straight (DoubleMat &pos, const UIntVec &hard_nodes, const DoubleVec &sizes, bool force_even,
UIntVec &indices);

// Creates a broken-line mesh between some imposed nodes (hard nodes) with specific anisotropic
metrics.
int mesh_straight (DoubleMat &pos, const UIntVec &hard_nodes, const DoubleMat &metrics, bool force_
even, UIntVec &indices);
```

Remeshing of lines

```
// Remeshes lines, connected or not, with possibly user-defined hard nodes and colors. Similar to CM2
SurfRemesh T3/Q4 but for edge meshes.
int remesh_lines (DoubleMat& pos, UIntMat& connectE, double target_h, const DoubleVec& metrics,
UIntVec& colors, UIntVec& hard_nodes, const UIntVec& hard_edges, bool force_even = false, double angle_
tol = 20., double max_chordal_error = -0.02, double max_gradation = 0.5, double min_h = 0., unsigned
min_n = 1, unsigned max_n = UINT_MAX);

// Remeshes lines, connected or not, with possibly user-defined colors. Simplified version. Same as
above without the metrics, hard_nodes and hard_edges arrays.
int remesh_lines (DoubleMat& pos, UIntMat& connectE, double target_h, UIntVec& colors, bool force_even
= false, double angle_tol = 20., double max_chordal_error = -0.02, double max_gradation = 0.5, double
min_h = 0., unsigned min_n = 1, unsigned max_n = UINT_MAX);

/// Remeshes lines, connected or not. Simplified version. Same as above without the metrics, colors,
hard_nodes and hard_edges arrays.
int remesh_lines (DoubleMat& pos, UIntMat& connectE, double target_h, bool force_even = false, double
angle_tol = 20., double max_chordal_error = -0.02, double max_gradation = 0.5, double min_h = 0.,
unsigned min_n = 1, unsigned max_n = UINT_MAX);
```

Reorientation of E2 elements

```
// Changes the orientation of the edges in a connectivity matrix (considered as edges).
int flip (UIntMat &connectE);

// Changes the orientation of the edges so that all elements have similar orientation.
int mesh_reorient (UIntMat &connect, UIntMat &neighbors, unsigned ref_element, int code);
```

Normals and curvatures computation routine along E2 meshes

```
// The averaged nodal normals along a line mesh (2-D only).
int normals (const DoubleMat &pos, const UIntMat &connectE, double w, bool normalize_flag, DoubleMat
&D);

// Approximate nodal curvatures (inverse of radius) along a line mesh.
int curvatures (const DoubleMat &pos, const UIntVec &nodes, DoubleVec &invR);

// Approximative nodal curvatures (inverse of radius) at each node along a line mesh.
int parametric_curvatures (const DoubleVec &pos1D, const UIntMat &connectE, const DoubleMat &local_
bases, DoubleVec &curvatures);
```

Miscellaneous routines for E2 meshes

```
// Length of a line mesh.
int length (const DoubleMat &pos, const UIntVec &nodes, double &L);

// Length of a line mesh.
int length (const DoubleMat &pos, const UIntMat &connectE, double &L);

// Interpolates nodes onto a polyline.
int interpolate_coordinates (DoubleMat &pos, const UIntVec &nodes0, const DoubleVec &U0, const UIntVec
&nodes1, const DoubleVec &U1);

// Sorts nodes by increasing curvilinear coordinates.
int sort (DoubleVec &Us, UIntVec &nodes);

// Subdivides a 1-D line.
int subdivide (DoubleVec &pos, DoubleVec &metrics, const UIntVec &nodes, bool force_up, bool force_
even, unsigned min_n, unsigned max_n, UIntVec &indices);
```

Anisotropic metrics generation routines along E2 meshes

```
// Generates 2-D anisotropic metrics for a given set of edges.
int metrics_gen_aniso2d (const DoubleMat &pos, const UIntMat &connectE, double hn, DoubleMat &M);

// Generates 2-D anisotropic metrics for a given set of nodes.
int metrics_gen_aniso2d (const DoubleMat &normals, const UIntVec &node_IDs, double hn, double ht,
DoubleMat &M);

// Generates 3-D anisotropic metrics for a given line mesh.
int metrics_gen_aniso3d (const DoubleMat &pos, const UIntMat &connectE, double hn, DoubleMat &M);

// Generates 3-D anisotropic metrics for a given line mesh (different sizes along normal and binormal).
int metrics_gen_aniso3d (const DoubleMat &pos, const UIntMat &connectE, double hn, double hb, DoubleMat
&M);
```

Interpolation of metrics on E2 meshes

```
// Interpolates a field of scalar metrics on a line mesh.
int interpolate_metrics (const DoubleVec &Us, const DoubleVec &Smetrics, const DoubleVec &Ug, DoubleVec
&Gmetrics);

// Interpolates a field of tensorial metrics on a line mesh.
int interpolate_metrics (const DoubleVec &Us, const DoubleMat &Smetrics, const DoubleVec &Ug, DoubleMat
&Gmetrics);
```

cm2::meshtools2d functions (97)

Boolean operations

```
// Computes differences, intersection and union between (usually) two 2-D domains defined by their boundary edges.
template <class IntersectorE2, class MesherT3>
int boolean_ops (IntersectorE2& intersector, MesherT3& mesher2D, typename IntersectorE2::data_type& data, UIntMat& connectMU);
```

Transformations / conversion

```
// Converts T3 and Q4 elements into high-order elements (T6 and over, Q8 and over), creating new high-order nodes.
int convert_into_high_order (DoubleMat &pos, const UIntMat &connectL, unsigned Ne, unsigned Nqi, unsigned Nti, UIntMat &connectQ, const UIntMat &connectE);

// Converts in-place T3 and Q4 elements into high-order elements (T6 and over, Q8 and over), creating new high-order nodes.
int convert_into_high_order (DoubleMat &pos, UIntMat &connectM, unsigned Ne, unsigned Nqi, unsigned Nti, const UIntMat &connectE);

// Converts in-place T3 and Q4 elements into T6 and Q9 elements, creating new high-order nodes.
int convert_into_quadratic (DoubleMat &pos, UIntMat &connectM, const UIntMat &connectE3);

// Converts in-place T3 and Q4 elements into T6 and Q9 elements, creating new high-order nodes.
nt convert_into_quadratic (DoubleMat &pos, UIntMat &connectM);

// Converts in-place high-order face elements (T6, Q8 or Q9) into linear elements (T3, Q4).
int convert_into_linear (UIntMat &connectM);

// Splits T6 and Q9 elements into linear elements (T3 and Q4).
int split_into_linear (const UIntMat &connectQ, UIntMat &connectL);

// Splits quads into triangles (2 triangles each).
int split_Q4_into_T3 (const UIntMat &connectM, UIntMat &connectT);

// Splits quads into triangles (2 triangles each).
int split_Q4_into_T3 (const DoubleMat &pos, const UIntMat &connectM, UIntMat &connectT);

// Splits bad quads into triangles (2 triangles each).
int split_Q4_into_T3 (const DoubleMat &pos, const UIntMat &connectM0, double min_Q4_shape_quality, double min_Q4_angle_quality, double min_Q4_warp_quality, int split_criterion, UIntVec &good_IDs, UIntVec &bad_IDs, UIntMat &connectM1);

// Transforms a mixed linear 2-D mesh into a all-quad linear mesh by splitting each triangle into three sub-quads and each quad into four sub-quads.
int split_into_Q4 (DoubleMat &pos, const UIntMat &connectM, UIntMat &connectQ4, const UIntMat &connectE3);
```

Extrude translate into T3 or Q4 mesh

```
// Extrudes (sweeps) with translation a E2 mesh into a structured Q4 mesh (2-D version).
int extrude_translate_Q4 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec2 &T, unsigned N1, UIntMat &connectQ4);

// Extrudes (sweeps) with translation a E2 mesh into a structured Q4 mesh (3-D version).
int extrude_translate_Q4 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &T, unsigned N1, UIntMat &connectQ4);

// Extrudes (sweeps) with translation a E2 mesh into a structured Q4 mesh with specific sizes at the beginning and at the end of the translation (2-D version).
int extrude_translate_Q4 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec2 &T, double h0, double h1, bool force_even, UIntMat &connectQ4);

// Extrudes (sweeps) with translation a E2 mesh into a structured Q4 mesh with specific sizes at the beginning and at the end of the translation (3-D version).
int extrude_translate_Q4 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &T, double h0, double h1, bool force_even, UIntMat &connectQ4);

// Extrudes (sweeps) with translation a E2 mesh into a structured T3 mesh (2-D version).
int extrude_translate_T3 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec2 &T, unsigned N1, int pattern, UIntMat &connectT3);

// Extrudes (sweeps) with translation a E2 mesh into a structured T3 mesh (3-D version).
int extrude_translate_T3 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &T, unsigned N1, int pattern, UIntMat &connectT3);
```

```
// Extrudes (sweeps) with translation a E2 mesh into a structured T3 mesh with specific sizes at the beginning and at the end of the translation (2-D version).
int extrude_translate_T3 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec2 &T, double h0, double h1, int pattern, bool force_even, UIntMat &connectT3);

// Extrudes (sweeps) with translation a E2 mesh into a structured T3 mesh with specific sizes at the beginning and at the end of the translation (3-D version).
int extrude_translate_T3 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &T, double h0, double h1, int pattern, bool force_even, UIntMat &connectT3);
```

Extrude rotate into T3 or Q4 mesh

```
// Extrudes (sweeps) with rotation a E2 mesh into a structured T3 mesh.
int extrude_rotate_T3 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &C, const DoubleVec3 &R, unsigned N1, int pattern, UIntMat &connectT3);

// Extrudes (sweeps) with rotation a E2 mesh into a structured T3 mesh.
int extrude_rotate_T3 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec2 &C, double Rz, unsigned N1, int pattern, UIntMat &connectT3);

// Extrudes (sweeps) with rotation a E2 mesh into a structured T3 mesh with specific sizes at the beginning and at the end of the rotation.
int extrude_rotate_T3 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &C, const DoubleVec3 &R, double h0, double h1, int pattern, bool force_even, UIntMat &connectT3);

// Extrudes (sweeps) with rotation a E2 mesh into a structured T3 mesh with specific sizes at the beginning and at the end of the rotation.
int extrude_rotate_T3 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec2 &C, double Rz, double h0, double h1, int pattern, bool force_even, UIntMat &connectT3);

// Extrudes (sweeps) with rotation a E2 mesh into a structured Q4 mesh.
int extrude_rotate_Q4 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &C, const DoubleVec3 &R, unsigned N1, UIntMat &connectQ4);

// Extrudes (sweeps) with rotation a E2 mesh into a structured Q4 mesh.
int extrude_rotate_Q4 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec2 &C, double Rz, unsigned N1, UIntMat &connectQ4);

// Extrudes (sweeps) with rotation a E2 mesh into a structured Q4 mesh with specific sizes at the beginning and at the end of the rotation.
int extrude_rotate_Q4 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &C, const DoubleVec3 &R, double h0, double h1, bool force_even, UIntMat &connectQ4);

// Extrudes (sweeps) with rotation a E2 mesh into a structured Q4 mesh with specific sizes at the beginning and at the end of the rotation.
int extrude_rotate_Q4 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec2 &C, double Rz, double h0, double h1, bool force_even, UIntMat &connectQ4);
```

Extrudes with both rotation and translation into a T3 or Q4 mesh

```
// Extrudes (sweeps) with rotation and translation a E2 mesh into a structured T3 mesh.
int extrude_spiral_T3 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &C, const DoubleVec3 &R, const DoubleVec3 &T, unsigned N1, int pattern, UIntMat &connectT3);

// Extrudes (sweeps) a E2 mesh with rotation and translation into a structured T3 mesh.
int extrude_spiral_T3 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &C, const DoubleVec3 &R, const DoubleVec3 &T, double h0, double h1, int pattern, bool force_even, UIntMat &connectT3);

// Extrudes (sweeps) with rotation and translation a E2 mesh into a structured Q4 mesh.
int extrude_spiral_Q4 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &C, const DoubleVec3 &R, const DoubleVec3 &T, unsigned N1, UIntMat &connectQ4);

// Extrudes (sweeps) with rotation and translation a E2 mesh into a structured Q4 mesh.
int extrude_spiral_Q4 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &C, const DoubleVec3 &R, const DoubleVec3 &T, double h0, double h1, bool force_even, UIntMat &connectQ4);
```

Extrudes normal into a T3 or Q4 mesh

```
// Extrudes (sweeps) a E2 mesh along specific directions into a structured T3 mesh.
int extrude_normal_T3 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &D, unsigned N1, int pattern, UIntMat &connectT3);

// Extrudes (sweeps) a E2 mesh along the normal directions into a structured T3 mesh.
int extrude_normal_T3 (DoubleMat &pos, const UIntMat &connectE2, double T, unsigned N1, int pattern, UIntMat &connectT3);

// Extrudes (sweeps) a E2 mesh along the normal directions into a structured T3 mesh.
int extrude_normal_T3 (DoubleMat &pos, const UIntMat &connectE2, double T, double h0, double h1, bool force_even, int pattern, UIntMat &connectT3);

// Extrudes (sweeps) a E2 mesh along specific directions into a structured Q4 mesh.
int extrude_normal_Q4 (DoubleMat &pos, const UIntMat &connectE2, const DoubleVec3 &D, unsigned N1, UIntMat &connectT3);
```

```
// Extrudes (sweeps) a E2 mesh along the normal directions into a structured Q4 mesh.
int extrude_normal_Q4 (DoubleMat &pos, const UIntMat &connectE2, double T, unsigned N1, UIntMat
&connectT3);

// Extrudes (sweeps) a E2 mesh along the normal directions into a structured Q4 mesh.
int extrude_normal_Q4 (DoubleMat &pos, const UIntMat &connectE2, double T, double h0, double h1, bool
force_even, UIntMat &connectT3);
```

Fields interpolation on 2-D meshes

```
// Interpolates a scalar field (doubles) defined on the nodes of a 2-D mesh.
int interpolate (const DoubleMat &pos, const UIntMat &connectM, const UIntMat &neighbors, const UIntVec
&ancestors, DoubleVec &field, const UIntVec &nodes, cm2::element_type FE_type);

// Interpolates a vectorial field (doubles) defined on the nodes of a 2-D mesh.
int interpolate (const DoubleMat &pos, const UIntMat &connectM, const UIntMat &neighbors, const UIntVec
&ancestors, DoubleMat &field, const UIntVec &nodes, cm2::element_type FE_type);
```

Structured T3 or Q4 mesh (regular grid)

```
// Makes structured T3 mesh (regular grid).
int mesh_struct_T3 (DoubleMat &pos, const UIntVec &indices, unsigned N1, int pattern, UIntMat
&connectT3);

// Makes structured Q4 mesh (regular grid).
int mesh_struct_Q4 (DoubleMat &pos, const UIntVec &indices, unsigned N1, UIntMat &connectQ4);

// Computes metrics inside a regular grid by double interpolation (isotropic version).
int metrics_struct (const UIntVec &indices, unsigned N1, DoubleVec &metrics);

// Computes metrics inside a regular grid by double interpolation (anisotropic version).
int metrics_struct (const UIntVec &indices, unsigned N1, DoubleMat &metrics);
```

Pseudo-structured disk (quarter or full, T3 or Q4)

```
// Makes a pseudo-structured mesh on a disk quarter (upper-right quarter) with Q4 elements.
int mesh_disk_UR_Q4 (DoubleMat& pos, const DoubleVec2& C, double R, unsigned N, UIntMat& connectM,
UIntMat& connectE);

// Makes a pseudo-sstructured mesh on a disk quarter (upper-right quarter) with T3 elements.
int mesh_disk_UR_T3 (DoubleMat& pos, const DoubleVec2& C, double R, unsigned N, int pattern, UIntMat&
connectM, UIntMat& connectE);

// Makes a pseudo-structured mesh on a full disk with Q4 elements.
int mesh_disk_Q4 (DoubleMat& pos, const DoubleVec2& C, double R, unsigned N, UIntMat& connectM,
UIntMat& connectE);

// Makes a pseudo-structured mesh on a full disk with T3 elements.
int mesh_disk_T3 (DoubleMat& pos, const DoubleVec2& C, double R, unsigned N, int pattern, UIntMat&
connectM, UIntMat& connectE);
```

Pseudo-structured sphere (1/8 or full, T3 or Q4)

```
// Makes a pseudo-structured shell mesh on a 1/8 sphere ("positive" quadrant) with T3 elements.
int mesh_sphere_UR_T3 (DoubleMat& pos, const DoubleVec3& C, double R, unsigned N, int pattern, UIntMat&
connectM, UIntMat& connectE);

// Makes a pseudo-structured shell mesh on a 1/8 sphere ("positive" quadrant) with Q4 elements.
int mesh_sphere_UR_Q4 (DoubleMat& pos, const DoubleVec3& C, double R, unsigned N, UIntMat& connectM,
UIntMat& connectE);

// Makes a pseudo-structured shell mesh on a full sphere with T3 elements.
int mesh_sphere_T3 (DoubleMat& pos, const DoubleVec3& C, double R, unsigned N, int pattern, UIntMat&
connectM, UIntMat& connectE, UIntMat& connectG);

// Makes a pseudo-structured shell mesh on a full sphere with T3 elements.
int mesh_sphere_T3 (DoubleMat& pos, const DoubleVec3& C, double R, unsigned N, int pattern, UIntMat&
connectM);

// Makes a pseudo-structured shell mesh on a full sphere with Q4 elements.
int mesh_sphere_Q4 (DoubleMat& pos, const DoubleVec3& C, double R, unsigned N, UIntMat& connectM,
UIntMat& connectE, UIntMat& connectG);

// Makes a pseudo-structured shell mesh on a full sphere with Q4 elements.
int mesh_sphere_Q4 (DoubleMat& pos, const DoubleVec3& C, double R, unsigned N, UIntMat& connectM);
```

Structured surface meshes on parallelepipeds (regular grids)

```
/// Makes a structured surface mesh on a parallelepiped with Q4 elements.
int mesh_parallelepiped_Q4 (DoubleMat& pos, const matrix_fixed<double, 3, 8>& P07, unsigned N0,
unsigned N1, unsigned N2, UIntMat& connectM);

/// Makes a structured surface mesh on a XYZ-aligned parallelepiped with Q4 elements.
int mesh_parallelepiped_Q4 (DoubleMat& pos, const DoubleVec3& P0, const DoubleVec3& P6, unsigned Nx,
unsigned Ny, unsigned Nz, UIntMat& connectM);

/// Makes a structured surface mesh on a parallelepiped with T3 elements.
int mesh_parallelepiped_T3 (DoubleMat& pos, const matrix_fixed<double, 3, 8>& P07, unsigned N0,
unsigned N1, unsigned N2, int pattern, UIntMat& connectM);

/// Makes a structured surface mesh on a XYZ-aligned parallelepiped with T3 elements.
int mesh_parallelepiped_T3 (DoubleMat& pos, const DoubleVec3& P0, const DoubleVec3& P6, unsigned Nx,
unsigned Ny, unsigned Nz, int pattern, UIntMat& connectM);
```

CAD meshing on parametric surfaces.

```
// Meshes a parametric 3-D surface using a 2-D anisotropic mesher and mappings.
template <class Surface, class AnisoMesher, class AuxMesher>
int mesh_surface_param (const Surface &S, AnisoMesher &mesher2D, typename AnisoMesher::data_type
&data3D, AuxMesher &aux_mesher, double max_chordal_error, double min_h, unsigned chordal_control_type,
unsigned high_order_type=0, unsigned high_order_mode=2, double max_chordal_error_ratio=0.10, bool dry_
run_flag=false, unsigned max_bgm_remeshings=4, bool recompute_Qs_flag=true, bool compute_area_
flag=true);
```

Offsets

```
// Offsets a surface mesh in specific directions. Spherical offset.
int offset (DoubleMat& pos, const UIntMat& connectM, const DoubleMat& D, double h);

// Offsets a surface mesh in directions normal to the surface. Spherical offset.
int offset (DoubleMat& pos, const UIntMat& connectM, double h);

// Duplicates and offsets a surface mesh in specific directions. Spherical offset.
int copy_with_offset (DoubleMat& pos, const UIntMat& connectM0, const DoubleMat& D, double h, UIntMat&
connectM1);

// Duplicates and offsets a surface mesh in directions normal to the surface. Spherical offset.
int copy_with_offset (DoubleMat& pos, const UIntMat& connectM0, double h, UIntMat& connectM1);

// Offsets a surface mesh in specific directions. Anisotropic offset.
int offset (DoubleMat& pos, const UIntMat& connectM, const DoubleMat& D, const DoubleSym3& M);

// Offsets a surface mesh in directions normal to the surface. Anisotropic  offset.
int offset (DoubleMat& pos, const UIntMat& connectM, const DoubleSym3& M);

// Duplicates and offsets a surface mesh in specific directions. Anisotropic  offset.
int copy_with_offset (DoubleMat& pos, const UIntMat& connectM0, const DoubleMat& D, const DoubleSym3&
M, UIntMat& connectM1);

// Duplicates and offsets a surface mesh in directions normal to the surface. Anisotropic  offset.
int copy_with_offset (DoubleMat& pos, const UIntMat& connectM0, const DoubleSym3& M, UIntMat&
connectM1);
```

Reorientation of 2-D elements

```
// Changes the orientation of the faces in a 2-D mesh (T3, Q4, mixed).
int flip (UIntMat &connect);

// Changes the orientation of the faces in a 2-D mesh (T3, Q4, mixed, T6, Q8 or Q9).
int flip (UIntMat &connect, cm2::element_type FE_type);

// Changes the orientation of the triangles in a 2-D mesh (T3, Q4 or  mixed) so that all elements have
similar orientation as a reference element.
int mesh_reorient (UIntMat &connectM, UIntMat &neighbors, unsigned ref_element, int code);
```

Special transformation for degenerated quadrangles

```
// Transforms the degenerated quads with two consecutive identical node IDs into CM2-compliant
triangles stored in a mixed mesh.
int convert_degenerated_Q4 (UIntMat &connect);

// Transforms the degenerated quads with two consecutive identical node IDs into CM2-compliant
triangles stored in a mixed mesh.
int convert_degenerated_Q (UIntMat &connect, UIntVec &modified_IDs);
```

Area computation for T3 and Q4 meshes (planar or not)

```
// The area of a triangle or quadrangle mesh (planar or not).
int area (const DoubleMat &pos, const UIntMat &connectM, double &s);
```

Voronoi cells for T3 meshes (planar or not)

```
// The circumcenters (Voronoi points) of triangles (2D or 3-D).
int circumcenters (const DoubleMat &pos, const UIntMat &connect, DoubleMat &PC);

// Computes the connected faces around nodes (triangles or quads).
int get_connected_elements (const UIntMat &connect, const UIntMat &neighbors, const UIntVec &ancestors,
UIntVec &xadj, UIntVec &adjncy);
```

Angle computation in T3 and Q4 meshes (planar or not)

```
// The angles (min and max) at nodes of each element in a triangle or quadrangle mesh (planar or not).
int angles (const DoubleMat &pos, const UIntMat &connectM, bool normalized, DoubleVec &min_angles,
DoubleVec &max_angles);
```

Normal computations on surface meshes (triangles and quadrangles)

```
// The element directors/normals on a 3-D surface (at centre of elements).
int normals (const DoubleMat &pos, const UIntMat &connectM, bool normalize_flag, DoubleMat &D);

// The averaged nodal normals on a 3-D surface mesh (triangles and/or quadrangles).
int normals (const DoubleMat &pos, const UIntMat &connectM, double w, bool normalize_flag, DoubleMat
&D);
```

Curvature computations on surface meshes (triangles only)

```
// Computes the two approximative principal curvatures (directions and inverse of radii) at each node
of a triangle 3-D surface mesh.
int principal_curvatures (const DoubleMat &pos, const UIntMat &connectM, const UIntMat &neighbors,
double R, double angle_max, unsigned max_nefs_in_disks, const UIntMat &boundaries, DoubleMat
&curvatures);

// Approximative nodal curvature tensor (2x2 tensors) of a mesh.
int parametric_curvatures (const DoubleMat &pos2D, const UIntMat &connectM, const DoubleMat &local_
bases, DoubleMat &curvatures);
```

Anisotropic metrics generation routines on 2-D triangle meshes

```
// Generates a set of 3-D anisotropic metrics for a given 3-D surface triangle mesh.
int metrics_gen_aniso3d (const DoubleMat &pos, const UIntMat &connectT3, double hn, DoubleMat &M);
```

Interpolation of isotropic metrics on 2-D meshes

```
// Interpolates a field of isotropic metrics defined on a planar mesh.
int interpolate_metrics (const DoubleMat &pos, const UIntMat &connectM, const UIntMat &neighbors, const
UIntVec &ancestors, DoubleVec &metrics, bool invalid_metrics_only, cm2::element_type FE_type);

// Interpolates a field of isotropic metrics defined on a planar mesh.
int interpolate_metrics (const DoubleMat &pos, const UIntMat &connectM, const UIntMat &neighbors, const
UIntVec &ancestors, DoubleVec &metrics, const UIntVec &nodes, bool invalid_metrics_only, cm2::element_
type FE_type);
```

Interpolation of anisotropic metrics on 2-D meshes

```
// Interpolates a field of 2-D or 3-D anisotropic metrics defined on a planar mesh.
int interpolate_metrics (const DoubleMat &pos, const UIntMat &connectM, const UIntMat &neighbors, const
UIntVec &ancestors, DoubleMat &metrics, bool invalid_metrics_only, cm2::element_type FE_type);

// Interpolates a field of 2-D or 3-D anisotropic metrics defined on a planar mesh.
int interpolate_metrics (const DoubleMat &pos, const UIntMat &connectM, const UIntMat &neighbors, const
UIntVec &ancestors, DoubleMat &metrics, const UIntVec &nodes, bool invalid_metrics_only, cm2::element_
type FE_type);
```

Element localization in 2-D and surface meshes

```
// Finds an element containig a point in a plane mesh (all-triangle, all-quad or mixed).
int get_element_containing_point (const DoubleMat &pos, const UIntMat &connectM, const UIntMat
&neighbors, const UIntVec &ancestors, const meshtools::node_localizer& localizer,
element_type FE_type, const DoubleVec2& Pi, unsigned& Ki,  DoubleVec2& Qi, double tol = 1E-6);
```

```
// Finds an element containig a point in a surface mesh (all-triangle, all-quad or mixed).
int get_element_containing_point (const DoubleMat &pos, const UIntMat &connectM, const UIntMat
&neighbors, const UIntVec &ancestors, const meshtools::node_localizer& localizer,
element_type FE_type, const DoubleVec2& Pi, unsigned& Ki, DoubleVec3& Qi, double tol = 1E-6);

// Finds the elements containig a set of points in a plane or surface mesh (all-triangle, all-quad or
mixed).
int get_elements_containing_points (const DoubleMat &pos, const UIntMat &connectM, const UIntMat
&neighbors, const UIntVec &ancestors, const meshtools::node_localizer& localizer,
element_type FE_type, const DoubleMat& Ps, UIntVec& Ks, DoubleMat& Qs, double tol = 1E-6);

// Finds the elements containig a set of nodes in a plane or surface mesh (all-triangle, all-quad or
mixed).
int get_elements_containing_nodes (const DoubleMat &pos, const UIntMat &connectM,
element_type FE_type, const UIntVec& nodes, UIntVec& Ks, DoubleMat& Qs, double tol = 1E-6);
```

cm2::meshtools3d functions (48)

Boolean operations

```
// Computes differences, intersection and union between two solids defined by their boundary faces.
template <class IntersectorT3, class MesherTH4>
int boolean_ops (IntersectorT3& intersector, MesherTH4& mesher3D, typename IntersectorT3::data_type&
data, UIntMat& connectMU);
```

Transformations / conversion

```
// Converts TETRA4, PYRAMID5, WEDGE6 and HEXA8 elements into higher degree elements (TETRA10,
PYRAMID14, WEDGE18 and HEXA20 and over), creating new high-order nodes.
int convert_into_high_order (DoubleMat &pos, const UIntMat &connectL, unsigned Ne, unsigned Nfq,
unsigned Nft, unsigned Nhi, unsigned Nwi, unsigned Npi, unsigned Nti, UIntMat &connectQ, const UIntMat
&connectB, const UIntMat &connectE);

// Converts in-place TETRA4, PYRAMID5, WEDGE6 and HEXA8 elements into higher degree elements (TETRA10,
PYRAMID14, WEDGE18 and HEXA20 and over), creating new high-order nodes.
int convert_into_high_order (DoubleMat &pos, UIntMat &connectM, unsigned Ne, unsigned Nfq, unsigned
Nft, unsigned Nhi, unsigned Nwi, unsigned Npi, unsigned Nti, const UIntMat &connectB, const UIntMat
&connectE);

// Converts in-place TETRA4, PYRAMID5, WEDGE6 and HEXA8 elements into TETRA10 elements, PYRAMID14,
WEDGE18 and HEXA27, creating new high-order nodes.
int convert_into_quadratic (DoubleMat &pos, UIntMat &connectM, const UIntMat &connectB, const UIntMat
&connectE3);

// Converts in-place TETRA4, PYRAMID5, WEDGE6 and HEXA8 elements into TETRA10 elements, PYRAMID14,
WEDGE18 and HEXA27, creating new high-order nodes.
int convert_into_quadratic (DoubleMat &pos, UIntMat &connectM);

// Converts in-place high-order solid elements (TETRA10, PYRAMID13, PYRAMID14, WEDGE15, WEDGE18, HEXA20
or HEXA27) into linear elements (TETRA4, PYRAMID5, WEDGE6, HEXA8).
int convert_into_linear (UIntMat &connectM);

// Transforms a mixed 3-D mesh (without any pyramid!) into a all-hex mesh by splitting each tetrahedron
into four sub-hexes, each wedge into six sub-hexes and each hexahedron into eight sub-hexes.
int split_into_H8 (DoubleMat &pos, const UIntMat &connectM, UIntMat &connectH8, const UIntMat
&connectBQ, const UIntMat &connectE3);
```

Extrude translate a 2-D mesh into a 3-D mesh

```
// Extrudes (sweeps) with translation a 2-D mesh into a structured 3-D mesh.
int extrude_translate (DoubleMat &pos, const UIntMat &connect2D, const DoubleVec3 &T, unsigned N1,
UIntMat &connect3D);

// Extrudes (sweeps) with translation a 2-D mesh into a structured 3-D mesh, with specific sizes.
int extrude_translate (DoubleMat &pos, const UIntMat &connect2D, const DoubleVec3 &T, double h0, double
h1, bool force_even, UIntMat &connect3D);
```

Extrude rotate a 2-D mesh into a 3-D mesh

```
// Extrudes (sweeps) with rotation a 2-D mesh into a structured 3-D mesh.
int extrude_rotate (DoubleMat &pos, const UIntMat &connect2D, const DoubleVec3 &C, const DoubleVec3 &R,
unsigned N1, UIntMat &connect3D);

// Extrudes (sweeps) with rotation a 2-D mesh into a structured 3-D mesh, with specific sizes.
int extrude_rotate (DoubleMat &pos, const UIntMat &connect2D, const DoubleVec3 &C, const DoubleVec3 &R,
double h0, double h1, bool force_even, UIntMat &connect3D);
```

Extrudes with both rotation and translation a 2-D mesh into a 3-D mesh

```
// Extrudes (sweeps) with rotation and translation a 2-D mesh into a structured 3-D mesh.
int extrude_spiral (DoubleMat &pos, const UIntMat &connect2D, const DoubleVec3 &C, const DoubleVec3 &R,
const DoubleVec3 &T, unsigned N1, UIntMat &connect3D);

// Extrudes (sweeps) with rotation a 2-D mesh into a structured 3-D mesh, with specific sizes.
int extrude_spiral (DoubleMat &pos, const UIntMat &connect2D, const DoubleVec3 &C, const DoubleVec3 &R,
const DoubleVec3 &T, double h0, double h1, bool force_even, UIntMat &connect3D);
```

Extrudes a 2-D mesh along specific directions into a 3-D mesh

```
// Extrudes (sweeps) a 2-D mesh along specific directions into a structured 3-D mesh.
int extrude_normal (DoubleMat& pos, const UIntMat& connectBase, const DoubleMat& D, unsigned N1,
UIntMat& connect3D, const UIntMat& connectE, UIntMat& connectSide, UIntMat& connectTop);

// Extrudes (sweeps) a 2-D mesh along a specific line into a structured 3-D mesh. Spherical expansions.
int extrude_normal (DoubleMat& pos, const UIntMat& connectBase, const DoubleMat& D, unsigned N1,
UIntMat& connect3D);
```

```
// Extrudes (sweeps) a 2-D mesh along the node normals into a structured 3-D mesh.
int extrude_normal (DoubleMat& pos, const UIntMat& connectBase, double T, unsigned N1, UIntMat&
connect3D, const UIntMat& connectE, UIntMat& connectSide, UIntMat& connectTop);

// Extrudes (sweeps) a 2-D mesh along the node normals into a structured 3-D mesh.
int extrude_normal (DoubleMat& pos, const UIntMat& connectBase, double T, unsigned N1, UIntMat&
connect3D);

// Extrudes (sweeps) a 2-D mesh along the node normals into a structured 3-D mesh.
int extrude_normal (DoubleMat& pos, const UIntMat& connectBase, double T, double h0, double h1, bool
force_even, UIntMat& connect3D, const UIntMat& connectE, UIntMat& connectSide, UIntMat& connectTop);

// Extrudes (sweeps) a 2-D mesh along the node normals into a structured 3-D mesh.
int extrude_normal (DoubleMat& pos, const UIntMat& connectBase, double T, double h0, double h1, bool
force_even, unsigned N1, UIntMat& connect3D);
```

Extrudes a 2-D mesh along a line into a 3-D mesh

```
// Extrudes (sweeps) a 2-D mesh along a specific line into a structured 3-D mesh. Spherical expansions.
int extrude_line (DoubleMat& pos, const UIntMat& connectBase, const UIntVec& line_nodes, const
DoubleVec3& D0, const DoubleVec& expansion_H, UIntMat& connect3D, const UIntMat& connectE, UIntMat&
connectSide, UIntMat& connectTop);

// Extrudes (sweeps) a 2-D mesh along a specific line into a structured 3-D mesh. Spherical expansions.
int extrude_line (DoubleMat& pos, const UIntMat& connectBase, const UIntVec& line_nodes, const
DoubleVec3& D0, const DoubleVec& expansion_H, UIntMat& connect3D);

// Extrudes (sweeps) a 2-D mesh along a specific line into a structured 3-D mesh. Anisotropic
expansions.
int extrude_line (DoubleMat& pos, const UIntMat& connectBase, const UIntVec& line_nodes, const
DoubleVec3& D0, const DoubleMa& expansion_M, UIntMat& connect3D, const UIntMat& connectE, UIntMat&
connectSide, UIntMat& connectTop);

// Extrudes (sweeps) a 2-D mesh along a specific line into a structured 3-D mesh. Anisotropic
expansions.
int extrude_line (DoubleMat& pos, const UIntMat& connectBase, const UIntVec& line_nodes, const
DoubleVec3& D0, const DoubleMat& expansion_M, UIntMat& connect3D);
```

Interpolation of fields on 3-D tetrahedral meshes

```
// Interpolates a scalar field (doubles) defined on the nodes of a tetrahedral mesh.
int interpolate (const DoubleMat &pos, const UIntMat &connectTH4, const UIntMat &neighbors, const
UIntVec &ancestors, DoubleVec &field, const UIntVec &nodes);

// Interpolates a vectorial field (doubles) defined on the nodes of a tetrahedral mesh.
int interpolate (const DoubleMat &pos, const UIntMat &connectTH4, const UIntMat &neighbors, const
UIntVec &ancestors, DoubleMat &field, const UIntVec &nodes);
```

Reorientation of 3-D elements

```
// Changes the orientation of tetrahedrons (TH4).
int flip_TH4 (UIntMat &connectM);

// Changes the orientation of wedges (W6).
int flip_W6 (UIntMat &connectM);

// Changes the orientation of hexahedrons (H8).
int flip_H8 (UIntMat &connectM);

// Changes the orientation of the solid elements in a 3-D mesh (TH4, P5, WE6, H8, mixed, TH10, P14,
W18, H27, mixed).
int flip (UIntMat &connect, cm2::element_type FE_type);
```

Special transformations of 3-D meshes

```
// Duplicates the nodes of a surface mesh embedded inside a solid mesh and change accordingly the
connectivity of the solids over/under the surface.
int duplicate_surface_nodes (DoubleMat &pos, UIntMat &connectM, const UIntMat &neighbors, const UIntVec
&ancestors, const UIntMat &connectB, const UIntVec &except_nodes, bool upper_nodes_flag, bool lower_
nodes_flag, cm2::element_type FE_type);

// Splits the input solid elements having a Q4 face in the input list.
int split_solid_Q4 (DoubleMat &pos, const UIntMat &connectM, const UIntMat &connectQ4, const UIntVec
&diagonals, double min_qx, UIntVec &failed_indices, UIntMat &connectOut);
```

Volume computation

```
// Computes the volume of a 3-D mesh.
int volume (const DoubleMat &pos, const UIntMat &connectM, double &v);
```

Voronoi cells for tetrahedron meshes

```
// Computes the circumcenters (Voronoi points) of tetrahedron.
int circumcenters (const DoubleMat &pos, const UIntMat &connect, DoubleMat &PC);
```

Angle computation

```
// Computes the solid angles (min and max) at nodes of each element in a 3-D mesh.
int solid_angles (const DoubleMat &pos, const UIntMat &connectM, bool normalized, DoubleVec &min_
angles, DoubleVec &max_angles);

// Computes the dihedral angles (min and max) at edges of each element in a 3-D mesh.
int dihedral_angles (const DoubleMat &pos, const UIntMat &connectM, bool normalized, DoubleVec &min_
angles, DoubleVec &max_angles);
```

Interpolation of isotropic metrics on 3-D meshes

```
// Interpolates a field of isotropic metrics defined on a tetrahedral mesh.
int interpolate_metrics (const DoubleMat &pos, const UIntMat &connectTH4, const UIntMat &neighbors,
const UIntVec &ancestors, DoubleVec &metrics, bool invalid_metrics_only);

// Interpolates a field of isotropic metrics defined on a tetrahedral mesh.
int interpolate_metrics (const DoubleMat &pos, const UIntMat &connectTH4, const UIntMat &neighbors,
const UIntVec &ancestors, DoubleVec &metrics, const UIntVec &nodes, bool invalid_metrics_only);
```

Interpolation of anisotropic metrics on 3-D meshes

```
// Interpolates a field of 3-D anisotropic metrics defined on a tetrahedral mesh.
int interpolate_metrics (const DoubleMat &pos, const UIntMat &connectTH4, const UIntMat &neighbors,
const UIntVec &ancestors, DoubleMat &metrics, bool invalid_metrics_only);

// Interpolates a field of 3-D anisotropic metrics defined on a tetrahedral mesh.
int interpolate_metrics (const DoubleMat &pos, const UIntMat &connectTH4, const UIntMat &neighbors,
const UIntVec &ancestors, DoubleMat &metrics, const UIntVec &nodes, bool invalid_metrics_only);
```

Element localization in 3-D meshes

```
// Finds an element containig a point in a solid mesh (tetrahedrons only).
int get_element_containing_point (const DoubleMat &pos, const UIntMat &connectM, const UIntMat
&neighbors, const UIntVec &ancestors, const meshtools::node_localizer& localizer,
element_type FE_type, const DoubleVec3& Pi, unsigned& Ki, DoubleVec3& Qi, double tol = 1E-6);

// Finds the elements containig a set of points in a solid mesh (tetrahedrons only).
int get_elements_containing_points (const DoubleMat &pos, const UIntMat &connectM, const UIntMat
&neighbors, const UIntVec &ancestors, const meshtools::node_localizer& localizer,
element_type FE_type, const DoubleMat& Ps, UIntVec& Ks, DoubleMat& Qs, double tol = 1E-6);

// Finds the elements containig a set of nodes in a solid mesh (tetrahedrons only).
int get_elements_containing_nodes (const DoubleMat &pos, const UIntMat &connectM,
element_type FE_type, const UIntVec& nodes, UIntVec& Ks, DoubleMat& Qs, double tol = 1E-6);
```

COMPUTING
OBJECTS

https://www.computing-objects.com

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