LMGC90v2_Pre : catalogue of LMGC90's preprocessor functionnalities

A. Martin, M. Bagnéris, F. Dubois, R. Mozul

Laboratoire de Mécanique et Génie Civil

February 7, 2012

Domains

Granulometry

Need to compile Pre extension... (cmake option BUILD_PRE: True by default)
List of functions generating a granulometry according to different methods:

1. granulo_Monodisperse
2. granulo_Random
3. granulo_Uniform
4. granulo_TwoSizesNumber
5. granulo_TwoSizesVolume
6. granulo_ReadFromFile

Granulometry
granulo_Monodisperse

Generates a list of radii following a monodisperese distribution.

$$radii = granulo\_Monodisperse(nb\_particles, radius)$$

where:

- nb_particles (input integer), number of particles
- radius (input double), radius of the particles
- radii (returned double array), generated radii list

Granulometry
granulo_Random

Used in every examples of the prepro_grains directory.
Generates a random list of radii between bounds.

$$radii = granulo\_Random(nb\_particles, min\_radius, max\_radius)$$

where:

- nb_particles (input integer), number of particles
- min_radius (input double), minimum radius of the particles
- max_radius (input double), maximum radius of the particles
- radii (returned double array), generated radii list

Granulometry
granulo_Uniform

Generates a list of radii following a uniform distribution between bounds.

$$radii = granulo\_Uniform(nb\_particles, min\_radius, max\_radius)$$

where:

- nb_particles (input integer), number of particles
- min_radius (input double), minimum radius of the particles
- max_radius (input double), maximum radius of the particles
- radii (returned double array), generated radii list

Granulometry
granulo_TwoSizesNumber

Generates a list of radii of two sizes specifiying a ratio on numbers of particles.

$$radii = granulo\_TwoSizesNumber(nb\_particles, radius1, radius2, ratio)$$

where:

- nb_particles (input integer), number of particles
- radius1 (input double), first radius of the particles
- radius2 (input double), second radius of the particles
- ratio (input doube), ratio of particles of first radius
- radii (returned double array), generated radii list

In the end, in the radii array, there are about $ratio * nb\_particles$ particles of the first radius

Granulometry
granulo_TwoSizesVolume

Generates a list of radii of two sizes specifiying a ratio on volume of particles.

$$radii = granulo\_TwoSizesVolume(nb\_particles, radius1, radius2, ratio)$$

where:

- nb_particles (input integer), number of particles
- radius1 (input double), first radius of the particles
- radius2 (input double), second radius of the particles
- ratio (input doube), ratio of particles of first radius
- radii (returned double array), generated radii list

In the end, in the radii array, there is a number of the particles of the first radius so that their volume is *ratio*$*$the total volume of the particles.

Granulometry
granulo_ReadFromFile

Generates a list of radii from a file.

$$radii = granulo\_ReadFromFile(nb\_particles, file\_name)$$

where:

- nb_particles (input integer), number of particles
- file_name (input string), name of the file to read
- radii (returned double array), generated radii list

Deposit

Need to compile Pre extension... (cmake option BUILD_PRE: True by default)
List of functions deposing a list of radius in a container or on a lattice

1. Deposit in container 2D
   1.1 depositInBox2D
   1.2 depositInDisk2D
   1.3 depositInCouette2D
   1.4 depositInDrum2D
2. Deposit in container 3D
   2.1 depositInBox3D
   2.2 depositInCylinder3D
   2.3 depositInSphere3D
3. Deposit on lattices
   3.1 squareLattice2D
   3.2 triangularLattice2D
   3.3 cubicLattice3D

Warnings:

- the input granulometry may be changed on output if the number of remaining particles is less than the input number of particles
- to avoid interprenetration between particles and container, these function use a shrink based on the size of the particles

Deposit in container 2D

depositInBox2D

Deposit under gravity circular particles in a box center on 0 along x-axis.

$$[nb\_remaining\_particles, coor] = depositInBox2D(radii, lx, ly, \\ deposited\_radii = None, \\ deposited\_coor = None)$$

where:

- radii (input double array), list of radii
- lx (input double), length of the box in which to deposit
- ly (input double), height of the box in which to deposit
- deposited_radii (optional input double array), radii of particles supposed to be in the container before the deposit
- deposited_coor (optional input coordinates array), coordinates of deposited_radii particles
- nb_remaining_particles (returned integer), number of deposited particles in the container
- coor (returned coordinates array), coordinates of deposited particles

Deposit in container 2D
depositInDisk2D

Deposit under gravity of circular particles in a disk centered on 0.

$$[nb\_remaining\_particles, coor] = depositInDisk2D(radii, r, \\ deposited\_radii = None, \\ deposited\_coor = None)$$

where:

- radii (input double array), list of radii
- r (input double), radius of the container
- deposited_radii (optional input double array), radii of particles supposed to be in the container before the deposit
- deposited_coor (optional input coordinates array), coordinates of deposited_radii particles
- nb_remaining_particles (returned integer), number of deposited particles in the container
- coor (returned coordinates array), coordinates of deposited particles

Deposit in container 2D

depositInCouette2D

Deposit under gravity of circular particles in a container designed for a Couette shear.

$$[nb\_remaining\_particles, coor] = depositInCouette2D(radii, rint, rext, \\ deposited\_radii = None, \\ deposited\_coor = None)$$

where:

- radii (input double array), list of radii
- rint (input double), internal radius of the ring occupied by particles
- rext (input double), external radius of the ring occupied by particles
- deposited_radii (optional input double array), radii of particles supposed to be in the container before the deposit
- deposited_coor (optional input coordinates array), coordinates of deposited_radii particles
- nb_remaining_particles (returned integer), number of deposited particles in the container
- coor (returned coordinates array), coordinates of deposited particles

Deposit in container 2D
depositInDrum2D

Deposit under gravity of circular particles in a half drum.

$$[nb\_remaining\_particles, coor] = depositInDrum2D(radii, r, \\ deposited\_radii = None, \\ deposited\_coor = None)$$

where:

- radii (input double array), list of radii
- r (input double), radius of the drum
- deposited_radii (optional input double array), radii of particles supposed to be in the container before the deposit
- deposited_coor (optional input coordinates array), coordinates of deposited_radii particles
- nb_remaining_particles (returned integer), number of deposited particles in the container
- coor (returned coordinates array), coordinates of deposited particles

Deposit in container 3D
depositInBox3D

Deposit under gravity spherical particles in a box centered on 0 along x and y axis.

$$[nb\_remaining\_particles, coor] = depositInBox3D(radii, lx, ly, lz, \\ deposited\_radii = None, \\ deposited\_coor = None)$$

where:

- radii (input double array), list of radii
- lx (input double), width of the box along x-axis in which to deposit
- ly (input double), width of the box along y-axis in which to deposit
- lz (input double), height of the box in which to deposit
- deposited_radii (optional input double array), radii of particles supposed to be in the container before the deposit
- deposited_coor (optional input coordinates array), coordinates of deposited_radii particles
- nb_remaining_particles (returned integer), number of deposited particles in the container

Deposit in container 3D

depositInCylinder3D

Deposit under gravity of spherical particles in a cylinder with bottom at 0.

$$[nb\_remaining\_particles, coor] = depositInCylinder3D(radii, R, lz, \\ deposited\_radii = None, \\ deposited\_coor = None)$$

where:

- radii (input double array), list of radii
- R (input double), radius of the cylinder
- lz (input double), heigth of the cylinder
- deposited_radii (optional input double array), radii of particles supposed to be in the container before the deposit
- deposited_coor (optional input coordinates array), coordinates of deposited_radii particles
- nb_remaining_particles (returned integer), number of deposited particles in the container
- coor (returned coordinates array), coordinates of deposited particles

Deposit in container 3D

depositInSphere3D

Deposit under gravity of spherical particles in a sphere.

$$[nb\_remaining\_particles, coor] = depositInSphere3D(radii, R, center, \\ deposited\_radii = None, \\ deposited\_coor = None)$$

where:

- radii (input double array), list of radii
- R (input double), radius of the sphere
- center (input double array), center of the sphere
- deposited_radii (optional input double array), radii of particles supposed to be in the container before the deposit
- deposited_coor (optional input coordinates array), coordinates of deposited_radii particles
- nb_remaining_particles (returned integer), number of deposited particles in the container
- coor (returned coordinates array), coordinates of deposited particles

Deposit on lattice
squareLattice2D

Generate a list of coordinates on a square lattice

$$coor = squareLattice2D(nb\_ele, nb\_layer, l, x = 0.0, y = 0.0)$$

where:

- nb_ele (input integer), number of particles on the first layer
- nb_layer (input integer), number of layers
- l (input double), length of the lattice element
- x (optional input double), x coordinate of the lower left corner of the bounding box
- y (optional input double), y coordinate of the lower left corner of the bounding box
- coor (returned double array), coordinates of lattice $[x_1, y_1, x_2, y_2, ...x_n, y_n]$

Deposit on lattice
triangularLattice2D

Generate a list of coordinates on a triangular lattice

$$coor = triangularLattice2D(nb\_ele, nb\_layer, l, x = 0.0, y = 0.0,\\ orientation =' up')$$

where:

- nb_ele (input integer), number of particles on the first layer
- nb_layer (input integer), number of layers
- l (input double), length of the lattice element
- x (optional input double), x coordinate of the lower left corner of the bounding box
- y (optional input double), y coordinate of the lower left corner of the bounding box
- orientation (optional input 'up' or 'down'), orientation of the first lower triangular
- coor (returned double array), coordinates of lattice $[x_1, y_1, x_2, y_2, \ldots x_n, y_n]$

Deposit on lattice
cubicLattice3D

Generate a list of coordinates on a cubic lattice

$coor = cubicLattice3D(nb\_ele\_x, nb\_ele\_y, nb\_layer, l, x = 0.0, y = 0.0, z = 0.0)$

where:

- nb_ele_x (input integer), number of particles on the first layer along x-axis
- nb_ele_y (input integer), number of particles on the first layer along y-axis
- nb_layer (input integer), number of layers
- l (input double), length of the lattice element
- x (optional input double), x coordinate of the lower left corner of the bounding box
- y (optional input double), y coordinate of the lower left corner of the bounding box
- z (optional input double), z coordinate of the lower left corner of the bounding box
- orientation (optional input 'up' or 'down'), orientation of the first lower triangular
- coor (returned double array), coordinates of lattice $[x_1, y_1, z_1 \ldots x_n, y_n, z_n]$

Particles generation

1. rigidDisk
2. rigidSpher
3. rigidCluster
4. rigidPolygon
5. rigidDiscreteDisk
6. deformableParticles2D

Particles generation
rigidDisk

Create an avatar of a rigid disk.

$$body = rigidDisk(r, center, model, material,$$
$$theta = 0., color = 'BLEUx', number = None)$$

where:

- r (input double), radius of the desired particle
- center (input double array), coordinates of the center of the desired particle
- model (input model object), rigid model for the particle
- material (input material object), material of the particle
- theta (optional input double), rotation angle in the inertia frame
- color (optional input string), color of the disk contactor (5 characters string)
- number (optional input integer) index of the avatar
- body (returned avatar object), the avatar

Particles generation
rigidSphere

Create an avatar of a rigid sphere.

$$body = rigidSphere(r, center, model, material,\\ color = 'BLEUx', number = None)$$

where:

- r (input double), radius of the desired particle
- center (input double array), coordinates of the center of the desired particle
- model (input model object), rigid model for the particle
- material (input material object), material of the particle
- color (optional input string), color of the disk contactor (5 characters string)
- number (optional input integer) index of the avatar
- body (returned avatar object), the avatar

Particles generation

rigidCluster

Create an avatar of a cluster of rigid disks.

$$body = rigidCluster(r, center, nb\_disk, model, material, \\ theta = 0., color = 'BLEUx', number = None)$$

where:

- r (input double), radius of the bounding disk
- center (input double array), coordinates of the center of the desired particle
- nb_disk (input integer), number of disks of the cluster
- model (input model object), rigid model for the particle
- material (input material object), material of the particle
- theta (optional input double), rotation angle in the inertia frame
- color (optional input string), color of the disk contactor (5 characters string)
- number (optional input integer) index of the avatar
- body (returned avatar object), the avatar

Particles generation
rigidPolygon

Create an avatar of a rigid cluster of disks.

$$body = rigidPolygon(r, center, nb\_vertex, model, material,$$
$$theta = 0., color = 'BLEUx', number = None)$$

where:

- r (input double), bounding radius of the particle
- center (input double array), coordinates of the center of the desired particle
- nb_vertex (input integer), number of vertex of the polygon
- model (input model object), rigid model for the particle
- material (input material object), material of the particle
- theta (optional input double), rotation angle in the inertia frame
- color (optional input string), color of the disk contactor (5 characters string)
- number (optional input integer) index of the avatar
- body (returned avatar object), the avatar

Particles generation
deformableParticles2D

Create an avatar of a deformable disk or pentagon.

$$body = deformableParticle2D(r, center, type\_part, model, material, \\ theta = 0., color = 'BLEUx', number = None)$$

where:

- r (input double), bounding radius of the particle
- center (input double array), coordinates of the center of inertia of the desired particle
- type_part (input 'Disk' or 'pent'), the type of particle to generate
- model (input model object), model of the particle
- material (input material object), material of the particle
- theta (optional input double), rotation angle in the inertia frame
- color (optional input string), color of the contactor (5 characters string) on the group 'skin'
- number (optional input integer) index of the avatar
- body (returned avatar object), the avatar

Bricks generation

1. brick2D
   1.1 constructor
   1.2 rigidBrick
   1.3 deformableBrick
   1.4 explodedDeformableBrick
2. brick3D
   2.1 constructor
   2.2 rigidBrick

the *brick2D* object
constructor

Create a brick2D object

$$my\_brick = brick2D(name, lx, ly)$$

where:

- name (input string), name of the brick
- lx (input double), length of the brick
- ly (input double), height of the brick
- my_brick (returned brick2D object), the brick2D object

the *brick2D* object
constructor

Create a rigid avatar from a brick2D object

$$
\begin{aligned}
body = my\_brick.rigidBrick(&center, model, material, \\
&color = 'BLEUx', number = None)
\end{aligned}
$$

where:

- center (input double array), coordinates of the center of inertia of the avatar
- model (input model object), model of the avatar
- material (input material object), material of avatar
- color (optional input string), color of the POLYG contactor
- number (optional input integer), index of the avatar
- body (returned avatar object), the avatar object

the *brick2D* object
deformableBrick

Create a deformable avatar from a brick2D object

$$\begin{aligned} & body = my\_brick.deformableBrick(\\ & \quad center, material, model, type = \text{'4T3'}, nb\_elem\_x = 1, nb\_elem\_y = 1, \\ & \quad apabh = [], apabv = [], apabhc = 0.25, apabvc = 0.25, \\ & \quad colors = [\text{'HORIx'}, \text{'VERTx'}, \text{'HORIx'}, \text{'VERTx'}, number = None) \end{aligned}$$

where:

- center (input double array) coordinates of the center of inertia of the avatar
- material (input material object), material of avatar
- model (input model object), model of the avatar,
- type (optional input '4T3', '2T3', 'Q4' or 'Q8'), type of element to mesh the brick
- nb_elem_x (optional input integer), number of elements following *x*-axis
- nb_elem_y (optional input integer), number of elements following *y*-axis
- apabh (optional input double array), curvilign abscisses used to put candidate points on horizontal lines, size of the array is nb_elem_x

the *brick2D* object
explodedDeformableBrick

Create a list of deformable avatars from a brick2D object

$$
\begin{aligned}
&body = my\_brick.deformableBrick(\\
&\qquad center, material, model, type = '4T3', nb\_elem\_x = 1, nb\_elem\_y = 1, \\
&\qquad apabh = [], apabv = [], apabhc = 0.25, apabvc = 0.25, \\
&\qquad colors = ['HORIx', 'VERTx', 'HORIx', 'VERTx', shift = 0)
\end{aligned}
$$

where:

- center (input double array) coordinates of the center of inertia of the brick
- material (input material object), material of the avatars
- model (input model object), model of the avatars
- type (optional input '4T3', '2T3', 'Q4' or 'Q8'), type of element to mesh the brick
- nb_elem_x (optional input integer), number of elements following *x*-axis
- nb_elem_y (optional input integer), number of elements following *y*-axis
- apabh (optional input double), curvilign abscisses used to put candidate points on horizontal lines
- apabv (optional input double), curvilign abscisses used to put candidate

the *brick3D*
constructor

Create a brick3D object

$$my\_brick = brick3D(name, lx, ly, lz)$$

where:

- name (input string), name of the brick
- lx (input double), length of the brick
- ly (input double), depth of the brick
- lz (input double), height of the brick
- my_brick (returned brick3D object), the brick3D object

the *brick3D*
rigidBrick

Create a rigid avatar from a brick3D object

$$body = my\_brick.rigidBrick(center, model, material, color = 'BLEUx')$$

where:

- center (input double array), coordinates of the center of inertia of the avatar
- model (input model object), model of the avatar
- material (input material object), material of avatar
- color (optional input string), color of the POLYR contactor
- body (returned avatar object), the avatar object

Wall generation

1. *paneresse objects*
   1.1 *paneresse_simple*
   1.2 *paneresse_double*
2. *methods*
   2.1 *setNumberOfRows*
   2.2 *set/compute/JointThicknessBetweenRows*
   2.3 *set/compute/evaluate/Height*
   2.4 *setFirstRowBy/Length/NumberOfBricks*
   2.5 *buildRigidWall*

paneresse objects

paneresse_simple

Create a wall in 'paneresse simple' disposition.

$$my\_wall = paneresse\_simple(brick\_ref, disposition)$$

where:

- brick_ref (input brick object), a brick object describing the kind of brick used to build the wall
- disposition (input string 'paneresse', 'boutisse' or 'chant'), disposition of the brick in the wall
- my_wall (returned wall object), the object allowing to generate a list of avatar

paneresse objects

paneresse_double

Create a wall in 'paneresse double' disposition.

$$my\_wall = paneresse\_double(brick\_ref, disposition)$$

where:

- brick_ref (input brick object), a brick object describing the kind of brick used to build the wall
- disposition (input string 'paneresse' or 'chant'), disposition of the brick in the wall
- my_wall (returned wall object), the object allowing to generate a list of avatar

Methods of paneresse objects

setNumberOfRows

Set the number of rows of a *paneresse* object.

$$my\_wall.setNumberOfRows(nb\_rows, tol = 1.e05)$$

where:

- my_wall is a paneresse object
- nb_rows (input integer), the number of rows desired in the wall
- tol (optional input double), the tolerance used for comparison of real numbers

Methods of paneresse objects

setJointThicknessBetweenRows

Set the joint thickness between rows of a *paneresse*.

$$my\_wall.setJointThicknessBetweenRows(joint\_thickness)$$

where:

- my_wall is a paneresse object
- joint_thickness (input double), thickness of joint between rows

Method of paneresse objects

computeJointThickness

Compute the joint thickness of a wall, if the number or rows and the height is set.

$$my\_wall.computeJointThickness()$$

where:

- my_wall is a paneresse object

Method of paneresse objects

setHeight

Set the height of the wall.

$$my\_wall.setHeigth(height)$$

where:

- my_wall is a paneresse object
- height (input double), desired height of the wall

Method of paneresse objects

computeHeight

Compute the height of the wall from the number of rows and joint thickness.

$$my\_wall.setHeigth(height)$$

where:

- my_wall is a paneresse object
- height (input double), desired height of the wall

Method of paneresse objects

setFirstRowByNumberOfBricks

Set the first row of the wall by giving the type of the first brick of this row, the number of bricks in this row and the joint thickness.

$$my\_wall.setFirstRowByNumberOfBricks(first\_brick\_type,\ nb\_bricks, joint\_thickness)$$

where:

- my_wall is a paneresse object
- first_brick_type (input string '1' or '1/2'), describe if the brick begining the first row is a whole brick or a half of a brick
- nb_bricks (input double), the given number of bricks; this number can be fractional
- joint_thickness (input double), the given joint thickness for the first row

Method of paneresse objects

setFirstRowByLength

Set the first row of the wall by giving the type of the first brick of this row, the the length of this row and the joint thickness.

$$my\_wall.setFirstRowByLength(first\_brick\_type, length, joint\_thickness)$$

where:

- my_wall is a paneresse object
- first_brick_type (input string '1' or '1/2'), describe if the brick begining the first row is a whole brick or a half of a brick
- length (input double), the length of first row
- joint_thickness (input double), the given joint thickness for the first row

Method of paneresse objects

buildRigidWall

Generate the list of rigid avatars corresponding to the wall

$$my\_wall.buildRigidWall(origin, model, material,$$
$$colors = ['BLEUx', 'REDxx'], rtol = 1e-5)$$

where:

- my_wall is a paneresse object
- origin (input double array), location of origin of the wall
- model (input model object), model of the bricks of the wall
- material (input material object), material of the bricks of the wall
- colors (input list of two strings), color of contactors
- rtol (optional input double), relative tolerance used in floatting number comparaisons

Method of paneresse objects

buildRigidWallWithoutHalfBricks

Generate the list of rigid avatars corresponding to the wall without the half bricks for the 'harpage'. Methode usable only 'paneresse_simple' object.

$$my\_wall.buildRigidWall(origin, model, material, \\ colors = [\text{'}BLEUx\text{'}, \text{'}REDxx\text{'}], rtol = 1e-5)$$

where:

- my_wall is a paneresse object
- origin (input double array), location of origin of the wall
- model (input model object), model of the bricks of the wall
- material (input material object), material of the bricks of the wall
- colors (input list of two strings), color of contactors
- rtol (optional input double), relative tolerance used in floatting number comparaisons

Meshes

1. generation
   1.1 lecture
   1.2 buildMesh2D
   1.3 buildMeshH8
2. *mesh* objects
   2.1 constructor
   2.2 addNode
   2.3 addBulk
   2.4 buildMeshedAvatar
   2.5 separateMeshes
3. manipulation of *mesh* objects
   3.1 extractFreeSurface
   3.2 reorientSurfacicElement

Meshes generation

lecture

Read a mesh in a file. Supported file format are *gmsh* and *sysweld*

$$my\_mesh = lecture(name, dim, keep\_elements = [], scale\_factor = None)$$

where:

- name (input string), name of file to read
- dim (input integer), dimension
- keep_elements (optional string list), to keep only a subset of element types
- scale_factor (optional double), to rescale the read mesh
- my_mesh (returned mesh object), the read mesh

Meshes generation
buildMesh2D

Mesh a rectangle with a group for each boundary line (*'left'*, *'down'*, *'right'*, *'up'*)

$$my\_mesh = buildMesh2D(type\_mesh, x0, y0, lx, ly, nb\_elem\_x, nb\_elem\_y,$$
$$vertices = [], number = None)$$

where:

- type_mesh (input string '2T3', '4T3', 'Q4', 'Q8'), the type of element to use to mesh the rectangular
- x0 (input double), left corner position
- y0 (input double), lower corner position
- lx (input double), dimension of the rectangle along x-axis
- ly (input double), dimension of the rectangle along y-axis
- nb_elem_x (input integer), number of elements along x-axis
- nb_elem_y (input integer), number of elements along y-axis
- vertices (optional double array), a list of x,y-coordinates following a suitable Q4 mesh node ordering
- number (optional input integer), number of avatar

Meshes generation
buildMeshH8

Mesh a parallelepiped using hexaedric element. Group for each boundary line (*'left'*, *'down'*, *'right'*, *'up'*, *front*, *rear*) are also defined.

$$my\_mesh = buildMeshH8(x0, y0, z0, lx, ly, lz, nb\_elem\_x, nb\_elem\_y, nb\_elem\_z,$$
$$surfacic\_mesh\_type = 'Q4')$$

where:

- x0 (input double), left corner position
- y0 (input double), lower corner position
- z0 (input double), rear corner position
- lx (input double), dimension of the rectangle along *x*-axis
- ly (input double), dimension of the rectangle along *y*-axis
- lz (input double), dimension of the rectangle along *z*-axis
- nb_elem_x (input integer), number of elements along *x*-axis
- nb_elem_y (input integer), number of elements along *y*-axis
- nb_elem_z (input integer), number of elements along *z*-axis
- surfacic_mesh_type (optional string 'Q4' or '2T3'), the element type to use on surfaces

mesh objects
Constructor

Create an empty mesh object

$$my\_mesh = mesh(dimension)$$

where:

- dimension (input integer), the dimension of the mesh
- my_mesh (returned mesh object), the new mesh empty mesh object

mesh objects
addNode

Add a node to a mesh.

$$body = my\_mesh.addNode(noeud)$$

where:

- my_mesh, is the mesh object in which to add the node
- noeud (input node object), the node to add to my_mesh

mesh objects
addBulk

Add an element to a mesh.

$$body = my\_mesh.addBulk(ele)$$

where:

- my_mesh, is the mesh object in which to add the node
- ele (input bulk object), the bulk to add to my_mesh

mesh objects
buildMeshedAvatar

Building a meshed avatar. Contactors have to be added afterward to the body.

$$body = my\_mesh.buildMeshedAvatar(model, material)$$

where:

- my_mesh, is the mesh object to build the avatar from
- model (input model object), the model of the elements
- material (input material object), the material of the elements
- body (returned avatar object), the generated avatar

mesh objects
separateMeshes

When reading a file, only one mesh object is returned, even if there are several meshes in the file. This function allow to get back the mesh objects corresponding to the submeshes (in a Python dictionnary).

$$meshes = my\_mesh.separateMeshes(dim, entity\_type = \text{'}geometricalEntity\text{'}, keep\_all\_elements = True)$$

where:

- my_mesh, is the mesh object to build the avatar from
- dim (input integer), dimension depending on the type of mesh (surfacic or volumic)
- entity_type (optional input string 'geometricalEntity' or 'physicalEntity'), select according which type to separate the meshes
- keep_all_elements (optional input boolean), if all elements are to be kept or just those of the input dimension (dim)
- meshes (returned dictionnary), a dictionnary of mesh objects where keys are the entity_type names

manipulation of *mesh* objects

extractFreeSurface

Create a surfacic mesh corresponding to the free surface of a volumic mesh. The elements of the volumic mesh must be tetrahdra.

$$surfacic\_mesh = extractFreeSurface(volumic\_mesh)$$

where:

- volumic_mesh (input mesh object), a mesh object in 3D
- surfacic_mesh (returned mesh object), a mesh object in 2D

manipulation of *mesh* objects

reorientSurfacicElements

Reorient the surfacic elements of surfacic elements of a 3D mesh using the orientation of its volumic elements. The elements of the volumic mesh must be tetrahdra.

$$reorientSurfacicElements(volumic\_mesh)$$

where:

- volumic_mesh (input mesh object), the 3D mesh object to reorient

Meshed bodies generation

1. 2D avatars from mesh
   1.1 rigidFromMesh2D
   1.2 rigidsFromMesh2D
2. 2D avatars from meshed avatar
   2.1 explodeMeshedAvatar2D
3. 3D avatars from mesh
   3.1 volumicMeshToRigid3D
   3.2 surfacicMeshToRigid3D
   3.3 surfacicMeshesToRigid3D

2D avatars from mesh

rigidFromMesh2D

Create a rigid avatar using a mesh to define its geometric boundary.

$$body = rigidFromMesh2D(surfacic_mesh, model, material, color = 'BLEUx',\\ reverse = False)$$

where:

- surfacic_mesh (input mesh object), the 2D mesh object to use to generate the rigid avatar
- model (input model object), rigid model for the particle
- material (input material object), material of the particle
- color (optional input string), color of the POLYG contactors (5 characters string)
- reverse (optional input boolean), to specify if the elements need to be reversed
- body (returned avatar object), the rigid avatar with each element of surfacic_mesh being a POLYG contactor

2D avatars from mesh

rigidsFromMesh2D

Create an avatar containers of rigid avatars from the elements of a mesh.

$$bodies = rigidFromMesh2D(surfacic_mesh, model, material, color = 'BLEUx', reverse = False)$$

where:

- surfacic_mesh (input mesh object), the 2D mesh object to use to generate the rigid avatar
- model (input model object), rigid model for the particle
- material (input material object), material of the particle
- color (optional input string), color of the POLYG contactors (5 characters string)
- reverse (optional input boolean), to specify if the elements need to be reversed
- bodies (returned avatar container), the rigid avatars corresponding to each element of surfacic_mesh with a POLYG contactor

2D avatars from meshed avatar
expledMeshedAvatar2D

Create an avatar container of meshed avatars each corresponding to one element of a 2D avatar meshed with elements of the first order.

$$bodies = explodeMeshedAvatar2D(body, nbPoints = 2, color = 'BLEUx') :$$

where:

- body (input avatar object), a 2D avatar of a meshed body
- nbPoints (optional input integer), number of points to put on candidat contactor line
- color (optional input string), color of the contactors (5 characters string)
- bodies (returned avatar container), the meshed avatars each corresponding to an element of the input meshed avatar *body*

3D avatars from mesh
volumicMeshToRigid3D

Create a 3D rigid avatar from a mesh object by extracting its surfacic mesh and computing its mass and inertia. The elements of the mesh must be tetrahedra.

$$\mathit{body} = \mathit{volumicMeshToRigid3D}(\mathit{volumic\_mesh}, \mathit{model}, \mathit{material}, \mathit{color} = \text{'}\mathit{BLEUx}\text{'})$$

where:

- volumic_mesh (input mesh object), a volumic mesh object
- model (input model object), a rigid model
- material (input material object), the material of the avatar
- color (optional input string), the color of the POLYR contactor (5 characters string)
- body (returned avatar object), a mesh rigid object with a POLYR contactor corresponding to the surfacic mesh of the input volumic_mesh

3D avatars from mesh

surfacicMeshToRigid3D

Create a 3D rigid avatar from a surfacic mesh object in 3D by computing its mass and inertia. The elements of the mesh must be triangles.

$$\begin{aligned} body = surfacicMeshToRigid3D(&surfacic\_mesh, model, material, \\ &color = 'BLEUx') \end{aligned}$$

where:

- surfacic_mesh (input mesh object), a surfacic mesh object
- model (input model object), a rigid model
- material (input material object), the material of the avatar
- color (optional input string), the color of the POLYR contactor (5 characters string)
- body (returned avatar object), a mesh rigid object with a POLYR contactor corresponding to the input surfacic_mesh

3D avatars from mesh

surfacicMeshesToRigid3D

Create a 3D rigid avatar from a list of surfacic mesh object in 3D. The elements of the meshes must be triangles.

$$body = surfacicMeshesToRigid3D(surfacic\_meshes, model, material, color = 'BLEUx')$$

where:

- surfacic_mesh (input list of mesh objects), a list of surfacic mesh objects
- model (input model object), a rigid model
- material (input material object), the material of the build avatar
- color (optional input string), the color of the POLYR contactor (5 characters string)
- body (returned avatar object), a mesh rigid object with POLYF contactors corresponding to the input surfacic_meshes

Cluster of rigids wall generation

1. roughWall
2. fineWall
3. smoothWall
4. granuloRoughWall
5. roughWall3D
6. granuloRoughWall3D

Cluster or Rigids wall generation

roughWall

Create a 2D rigid rough wall made of a cluster of disks or polygons

$$body = roughWall(center, l, r, model, material, theta = 0., color = 'WALLx', nb\_vertex = 0, number = None)$$

where:

- center (input double array), coordinates of the inertia center of the wall
- l (input double), minimal length of the wall
- r (input double), bouding radius of the particles to use to generate the wall
- model (input model object), rigid model for the wall
- material (input material object), material of the wall
- theta (optional input double), rotation angle in the inertia frame
- color (optional input string), color of the contactors (5 characters string)
- nb_vertex (optional input integer), the number of vertices of the polygons to use to generate the wall, if inferior to 3 disks are used
- number (optional input integer) index of the avatar
- body (returned avatar object), the rough wall avatar

Cluster or Rigids wall generation

fineWall

Create a 2D rigid slightly rough wall made of a cluster of disks or polygons

$$body = fineWall(center, l, r, model, material, theta = 0., color = 'WALLx', \\ nb\_vertex = 0, number = None)$$

where:

- center (input double array), coordinates of the inertia center of the wall
- l (input double), minimal length of the wall
- r (input double), bouding radius of the particles to use to generate the wall
- model (input model object), rigid model for the wall
- material (input material object), material of the wall
- theta (optional input double), rotation angle in the inertia frame
- color (optional input string), color of the contactors (5 characters string)
- nb_vertex (optional input integer), the number of vertices of the polygons to use to generate the wall, if inferior to 3 disks are used
- number (optional input integer) index of the avatar
- body (returned avatar object), the sligthly rough wall avatar

Cluster or Rigids wall generation

smoothWall

Create a 2D rigid smooth rough wall made of a cluster of bricks (polygons)

$$body = smoothWall(center, l, h, nb\_polyg, model, material, \\ theta = 0., color = 'WALLx', number = None)$$

where:

- center (input double array), coordinates of the inertia center of the wall
- l (input double), length of the wall
- h (input double), height of the wall
- nb_polyg (input integer), number of bricks in the wall
- model (input model object), rigid model for the wall
- material (input material object), material of the wall
- theta (optional input double), rotation angle in the inertia frame
- color (optional input string), color of the contactors (5 characters string)
- number (optional input integer) index of the avatar
- body (returned avatar object), the sligthly rough wall avatar

Cluster or Rigids wall generation

granuloRoughWall

Create a 2D rigid rough wall made of a cluster of disks or polygons using a random granulometry to choose the bounding radius of each element of the cluster.

$$body = granuloRoughWall(center, l, rmin, rmax, model, material,$$
$$theta = 0., color =' WALLx', nb\_vertex = 0)$$

where:

- center (input double array), coordinates of the inertia center of the wall
- l (input double), minimal length of the wall
- rmin (input double), minimum radius use to generate the granulometry
- rmax (input double), maximum radius use to generate the granulometry
- model (input model object), rigid model for the wall
- material (input material object), material of the wall
- theta (optional input double), rotation angle in the inertia frame
- color (optional input string), color of the contactors (5 characters string)
- nb_vertex (optional input integer), the number of vertices of the polygons to use to generate the wall, if inferior to 3 disks are used

Cluster or Rigids wall generation

roughWall3D

Create a 3D rigid rough wall made of a cluster of spheres

$$body = roughWall3D(center, lx, ly, r, model, material, color = 'WALLx')$$

where:

- center (input double array), coordinates of the inertia center of the wall
- lx (input double), minimal length of the wall along $x-$axis
- ly (input double), minimal length of the wall along $y-$axis
- r (input double), radius of the spheres to use to generate the wall
- model (input model object), rigid model for the wall
- material (input material object), material of the wall
- color (optional input string), color of the SPHER contactors (5 characters string)
- body (returned avatar object), the rough wall avatar

Cluster or Rigids wall generation
granuloRoughWall3D

Create a 3D rigid rough wall made of a cluster of spheres using a random granulometry to choose the bounding radius of each sphere of the cluster.

$$body = granuloRoughWall(center, lx, ly, rmin, rmax, model, material, \\ color =' WALLx')$$

where:

- center (input double array), coordinates of the inertia center of the wall
- lx (input double), minimal length of the wall along $x-$axis
- ly (input double), minimal length of the wall along $y-$axis
- rmin (input double), minimum radius use to generate the granulometry
- rmax (input double), maximum radius use to generate the granulometry
- model (input model object), rigid model for the wall
- material (input material object), material of the wall
- color (optional input string), color of the contactors (5 characters string)
- body (returned avatar object), the sligthly rough wall avatar

Extrusion

1. extrudeRigid
2. extrudeRigids

Extrusion

extrudeRigid

Create a 3D rigid body by extruding an input 2D rigid avatar. The extruded avatar inherits the material and the contactor color of the input avatar. Extrusion is made along z-axis except for *JONCx* contactor which are placed in xOz plane. Possible contactors type extrusion are *'POLYG'*, *'DISKx'*, *'xKSID'* and *'JONCx'*

$$body = extrudeRigid(body2D, model3D, depth, factor = 1.e0,$$
$$extrudedDisk = 'Sphere', number = None)$$

where:

- body2D (input avatar object), must be a 2D rigid
- model3D (input model object), 3D rigid model for the new avatar
- depth (input double), depth of the extrusion
- factor (optional input double), homotetic factor
- extrudedDisk (optional input string 'Sphere' or 'Cylinder'), specifies how to extrude a disk
- number (optional input integer) index of the avatar
- body (returned avatar object), the new 3D rigid avatar

Extrusion
extrudeRigids

Create a container of 3D rigid bodies by extruding a 2D rigid avatar container. The extruded avatars inherits the material and the contactor color of the input avatars. Extrusion is made along *z*-axis except for *JONCx* contactor which are placed in *xOz* plane. Possible contactors type extrusion are *'POLYG'*, *'DISKx'*, *'xKSID'* and *'JONCx'*

$$body = extrudeRigids(bodies2D, model3D, depth, factor = 1.e0, extrudedDisk = \text{'}Sphere\text{'})$$

where:

- body2D (input avatar container), must be an avatar container of 2D rigid bodies
- model3D (input model object), 3D rigid model for the new avatar
- depth (input double), depth of the extrusion
- factor (optional input double), homotetic factor
- extrudedDisk (optional input string 'Sphere' or 'Cylinder'), specifies how to extrude a disk
- bodies (returned avatar container), the new 3D rigid avatars

Extrusion

extrudePolygon