LMGC90v2_Pre : le pré-processeur de LMGC90

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

Laboratoire de Mécanique et Génie Civil

Formation LMGC90 - janvier 2013

Sommaire

Pré-ambule : pourquoi un pre-processeur ?

- la construction d'échantillon pour les simulations par éléments discrets implique :
  - manipulation de grandes collections d'objets,
  - la géométrie des objets peut être simple (sphère) ou complexe (surface maillée), résulter d'assemblage (cluster), *etc.*
  - dans certains cas la collection doit vérifier des contraintes statistiques (*e.g.* granulométrie imposée, forme d'objets, *etc.*),
  - dans certains cas la position des objets doit vérifier des contraintes statistiques (*e.g.* densité, anisotropie),

  $\Rightarrow$ nécessité de générer automatiquement les collections d'objets.

Pré-ambule : pourquoi un pre-processeur ?

- les informations contenues dans les fichiers d'entrée de LMGC90 s'appuient sur la discrétisation des objets :
  - structures de données très codifiées et fichiers formatés (*cf.* BODIES.DAT, DRV_DOF.DAT, DOF.INI, *etc.*),
  - description fine s'appuyant sur des choix abscons d'indexation, *e.g.* C.L. données au nœud pour un corps maillé par éléments finis et non en un lieu géométrique,

$\Rightarrow$ nécessité de manipuler des informations plus "naturelles".

Pré-ambule : pourquoi un pre-processeur ?

- mutualisation et pérennisation des outils
  - structure de données générique et simple à utiliser,
  - procèdures d'écriture des fichiers partagées,
  - permettre de créer des outils "one-shot" sans avoir à faire de développements accessoires conséquents,

  $\Rightarrow$ besoin d'un environnement de développement.

Mise en œuvre

- implémentation en Python :
  - langage interprété et orienté objet,
  - type dictionnaire : gestion des options (*e.g.* contacteurs) et analyse de cohérence
  - possibilité de *wrapping* des pré-processeurs existants (SWIG).

Mise en œuvre

- implémentation en Python :
  - langage interprété et orienté objet,
  - type dictionnaire : gestion des options (*e.g.* contacteurs) et analyse de cohérence
  - possibilité de *wrapping* des pré-processeurs existants (SWIG).
    - fonctionnalités des anciens pré-processeurs disponibles par ce biais

Mise en œuvre

- implémentation en Python :
  - langage interprété et orienté objet,
  - type dictionnaire : gestion des options (*e.g.* contacteurs) et analyse de cohérence
  - possibilité de *wrapping* des pré-processeurs existants (SWIG).
    - fonctionnalités des anciens pré-processeurs disponibles par ce biais
    - pourquoi pas les vôtres ?

Base de données

- Idée générale du remplissage de la base de données :
  - créer un nouvel objet,
  - définir l'ensemble des caractéristiques de l'objet,
  - ajouter l'objet dans un conteneur dédié.
- les cinq types d'objet dont nous avons besoin sont :

Base de données

- Idée générale du remplissage de la base de données :
  - créer un nouvel objet,
  - définir l'ensemble des caractéristiques de l'objet,
  - ajouter l'objet dans un conteneur dédié.
- les cinq types d'objet dont nous avons besoin sont :
  - *material* (paramètres matériaux),

```
1 #create rigid materials
2 mats = materials()
3 tdur = material(name='TDURx', type='RIGID',
4                 density=1000.)
5 pdur = material(name='MOUxx', type='RIGID',
6                 density=100.)
7 mats.addMaterial(tdur)
8 mats.addMaterial(pdur)
```

Base de données

- Idée générale du remplissage de la base de données :
  - créer un nouvel objet,
  - définir l'ensemble des caractéristiques de l'objet,
  - ajouter l'objet dans un conteneur dédié.
- les cinq types d'objet dont nous avons besoin sont :
  - *material* (paramètres matériaux),
  - *model* (modèle physique + loi de comportement),

```
1 #create a rigid model
2 mods = models()
3 mod = model(name='rigid', type='MECAx', element='Rxx2D',
4               dimension=2)
5 mods.addModel(mod)
```

Base de données

- Idée générale du remplissage de la base de données :
  - créer un nouvel objet,
  - définir l'ensemble des caractéristiques de l'objet,
  - ajouter l'objet dans un conteneur dédié.
- les cinq types d'objet dont nous avons besoin sont :
  - *material* (paramètres matériaux),
  - *model* (modèle physique + loi de comportement),
  - *tact_behav* (loi d'interaction),

```
1 #create a frictional contact behaviour
2 tacts = tact_behavs()
3 b = tact_behav(name='iqsc0', type='IQS_CLB', fric=0.3)
4 tacts.addBehav(b)
```

Base de données

- Idée générale du remplissage de la base de données :
  - créer un nouvel objet,
  - définir l'ensemble des caractéristiques de l'objet,
  - ajouter l'objet dans un conteneur dédié.
- les cinq types d'objet dont nous avons besoin sont :
  - *material* (paramètres matériaux),
  - *model* (modèle physique + loi de comportement),
  - *tact_behav* (loi d'interaction),
  - *see_type* (table de visibilité),

```
1 #create visibility table
2 svs = see_tables()
3 sv1 = see_table( \
4     CorpsCandidat='RBDY2', candidat='DISKx', \
5     colorCandidat='BLEUx', \
6     CorpsAntagoniste='RBDY2', antagoniste='JONCx', \
7     colorAntagoniste='VERTx', \
8     behav=b, alert=0.1)
9 svs.addSeeTable(sv1)
```

Base de données

- Idée générale du remplissage de la base de données :
  - créer un nouvel objet,
  - définir l'ensemble des caractéristiques de l'objet,
  - ajouter l'objet dans un conteneur dédié.
- les cinq types d'objet dont nous avons besoin sont :
  - *material* (paramètres matériaux),
  - *model* (modèle physique + loi de comportement),
  - *tact_behav* (loi d'interaction),
  - *see_type* (table de visibilité),
  - *avatar* (objet physique),

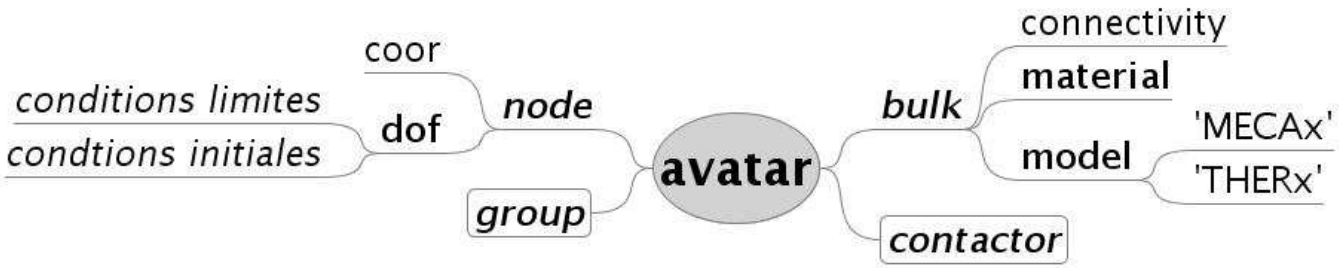
```
1 #create rigid avatars
2 bodies = avatars()
3 disk = avatar(type='RBDY2', dimension=2)
4 wall = avatar(type='RBDY2', dimension=2)
5 bodies.addAvatar(disk)
6 bodies.addAvatar(wall)
```

contrairement aux autres objets, un *avatar* n'est pas complet dès sa création.

Avatar : Représentation discrète d'un objet physique

- discrétisation de la géométrie :
  - ensemble de nœuds (*node*) décrits par leur coordonnées,
  - ensemble d'éléments (*bulk*) s'appuyant sur ces noeuds (tables de connectivité),
- modèle physique, loi de comportement (*model*) et paramètres (*material*) du matériau, portés par un groupe d'éléments,
- conditions initiales et aux limites (*dof*), portées par les nœuds d'appui d'un groupe d'éléments,
- description des zones géométriques susceptibles d'entrer en contact (*contactor*), portée par un groupe d'éléments.

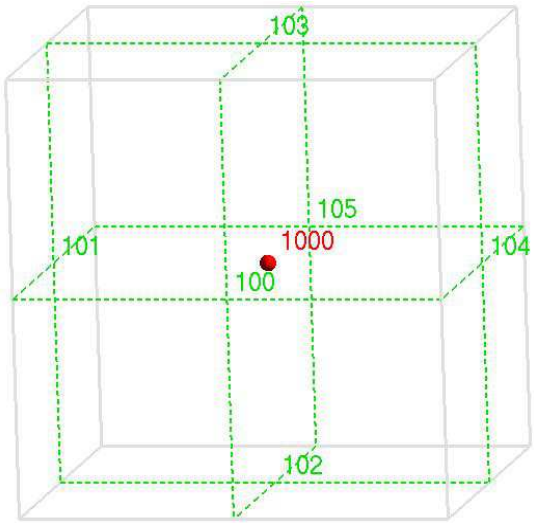
Avatar : Description géométrique




```
1 #adding a node
2 no = node(type='NO2xx', number=1,
3             coor=numpy.array([0.,0.]))
4 disk.addNode( no )
5
6 #adding a rigid bulk
7 disk.addBulk( rigid2d() )
```

Avatar : Affectation de propriétés

- représentation continue : propriétés affectées à des entités géométriques (*i.e.* sommet, ligne, surface, volume),
- représentation discrète ⇒ propriétés affectées à des groupes d'éléments (*group*) définis à la construction de la géométrie,

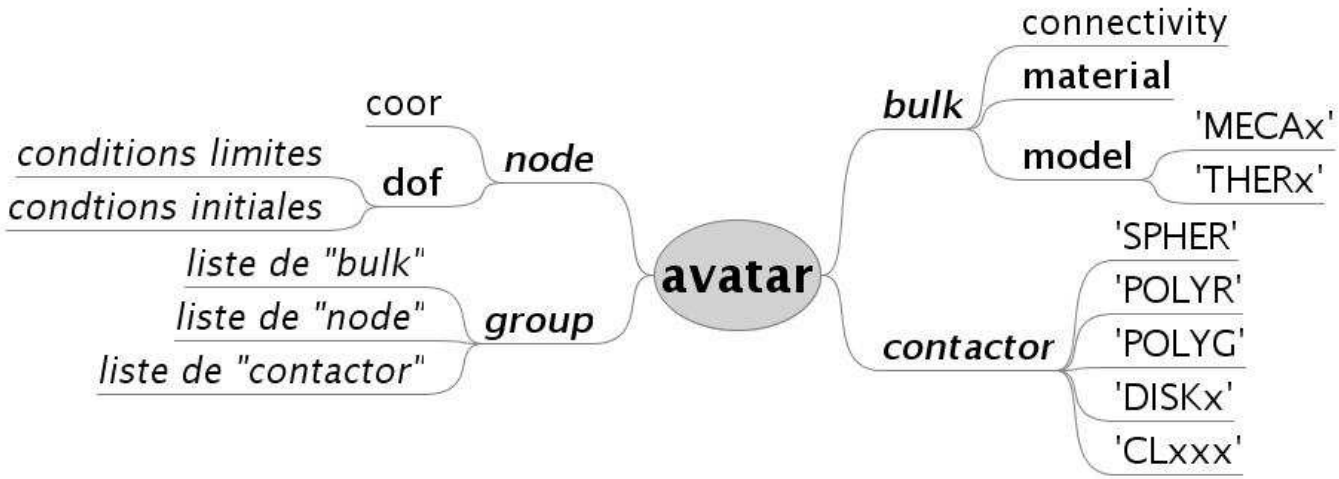


exemple :

- associer le comportement volumique (matériau, modèle) aux éléments volumiques d'indice "1000",
- encastrer les nœuds portés par les éléments surfaciques d'indice "102",
- associer un contacteur aux éléments surfaciques d'indice "103".

- groupe "all" : ensemble de tous les éléments.

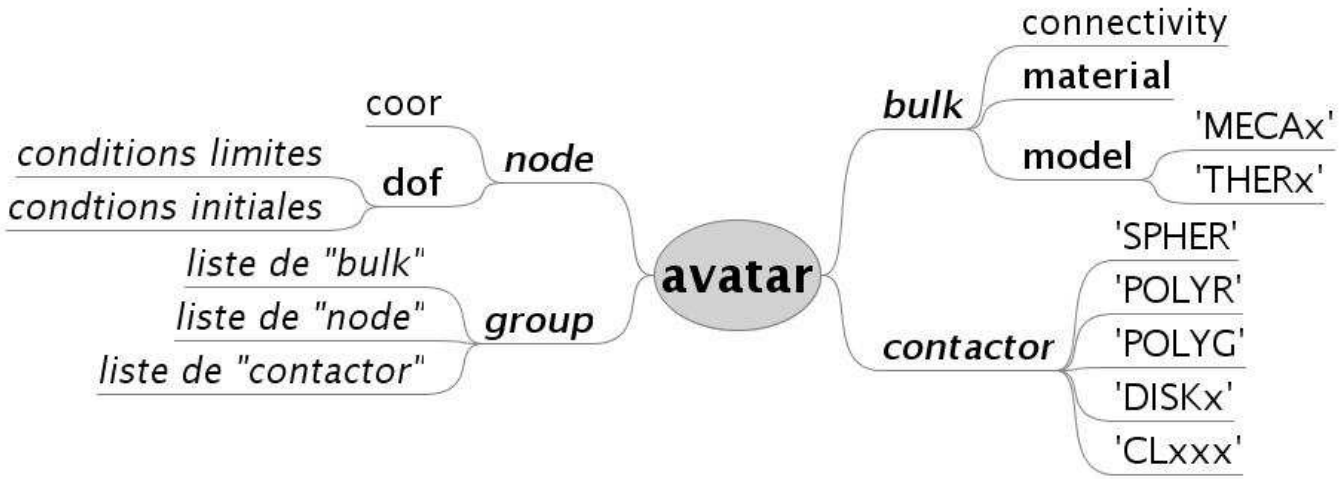
Avatar : Définition des groupes



- définition des groupes de l'avatar (groupes portés par les éléments + "all")

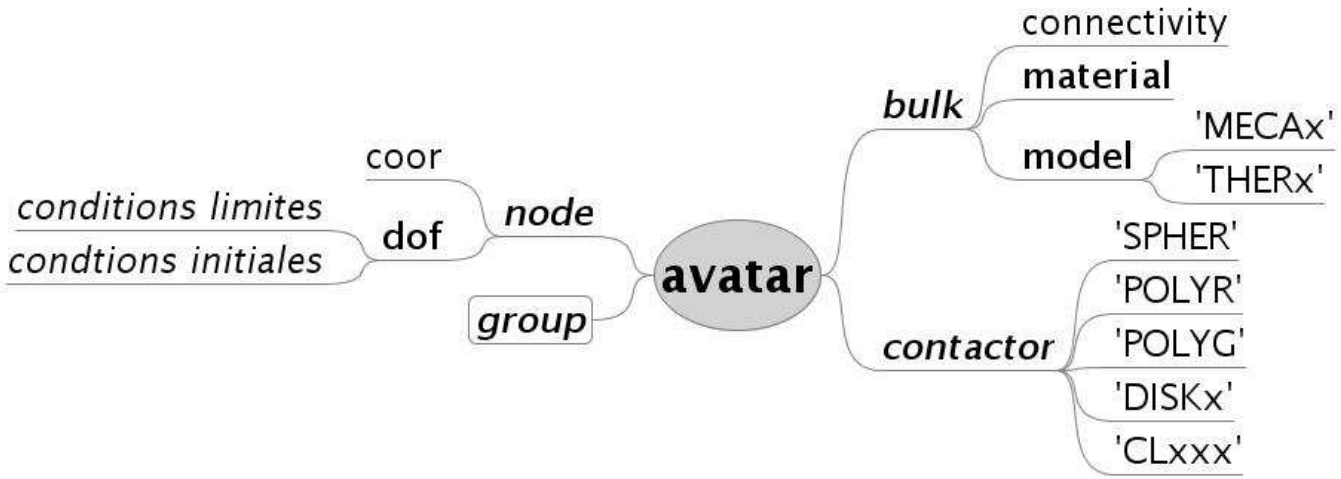
```
1 #defining group
2 disk.defineGroups()
```

Avatar : Définition des modèles et des matériaux




```
1 #model setting
2 disk.defineModel(group='all', model=mod)
3
4 #material setting
5 disk.defineMaterial(group='all', material=pdur)
```

Avatar : Ajout des contacteurs




```
1 #adding a contactor
2 disk.addContactors(group='all',
3                    type='DISKx', color='VERTx',
4                    byrd=0.05)
```

Avatar : Conditions aux limites et conditions initiales

- C.L. et C.I. appliquées aux degrés de liberté de nœuds supports d'un groupe d'éléments.

Avatar : Conditions aux limites et conditions initiales

- C.L. et C.I. appliquées aux degrés de liberté de nœuds supports d'un groupe d'éléments.
- nœud $\neq$ support géométrique !

Avatar : Conditions aux limites et conditions initiales

- C.L. et C.I. appliquées aux degrés de liberté de nœuds supports d'un groupe d'éléments.
- nœud $\neq$ support géométrique !
- $\Rightarrow$ On ne peut appliquer des C.L. ou des C.I. aux nœuds d'un groupe qu'après avoir affecté un modèle aux éléments de ce groupe.

Avatar : Conditions aux limites et conditions initiales

- C.L. et C.I. appliquées aux degrés de liberté de nœuds supports d'un groupe d'éléments.
- nœud $\neq$ support géométrique !
- $\Rightarrow$ On ne peut appliquer des C.L. ou des C.I. aux nœuds d'un groupe qu'après avoir affecté un modèle aux éléments de ce groupe.

```
1 #set an initial velocity
2 disk.imposeInitValue(group='all', \
3                       component=1, value=3.0)
4
5 #set a boundary condition
6 wall.imposeDrivenDof(group='all', \
7                       component=[1,2,3], dofty='vlocy')
```

Avatar : Harmonie entre représentation rigide et déformable

- "le corps rigide est un élément fini comme les autres".
  - connectivité : un nœud, le centre d'inertie,
  - stratégie élémentaire : résolution des équations de Newton-Euler,
  - degrés de liberté du nœud : $\left(\overrightarrow{v}_G, \overrightarrow{\omega}\right)$.

Avatar : Harmonie entre représentation rigide et déformable

- "le corps rigide est un élément fini comme les autres".
  - connectivité : un nœud, le centre d'inertie,
  - stratégie élémentaire : résolution des équations de Newton-Euler,
  - degrés de liberté du nœud : $\left(\overrightarrow{v}_G, \overrightarrow{\omega}\right)$.
- mais, besoin d'informations géométriques supplémentaires :
  - son volume V,
  - son inertie $\mathbb{I}$,
  - un repère pour définir les rotations.

Avatar : Harmonie entre représentation rigide et déformable

- "le corps rigide est un élément fini comme les autres".
  - connectivité : un nœud, le centre d'inertie,
  - stratégie élémentaire : résolution des équations de Newton-Euler,
  - degrés de liberté du nœud : $\left(\overrightarrow{v}_G, \overrightarrow{\omega}\right)$.
- mais, besoin d'informations géométriques supplémentaires :
  - son volume V,
  - son inertie $\mathbb{I}$,
  - un repère pour définir les rotations.
- calcul de ces informations à partir de la géométrie des contacteurs

```
1 # compute area and inertia of the disk
2 disk.computeRigidProperties()
3 # compute area and inertia of the wall
4 wall.computeRigidProperties()
```

Avatar : repositionnement des objets

- translation

```
1 # import math module
2 import math
3 # translate the disk
4 disk.translate(dx=0., dy=1.75, dz=0.)
```

- rotation (axe de rotation + angle)

```
1 # rotate the wall
2 wall.rotate(type='axis', alpha=0.5*math.pi
3             axis=[0., 0., 1.],
4             center=wall.nodes[1].coor)
```

- rotation (angles d'Euler)

```
1 # rotate the wall
2 wall.rotate(type='Euler',
3             psi=0.5*math.pi, theta=0., phi=0.
4             center=wall.nodes[1].coor)
```

Relation entre les objets

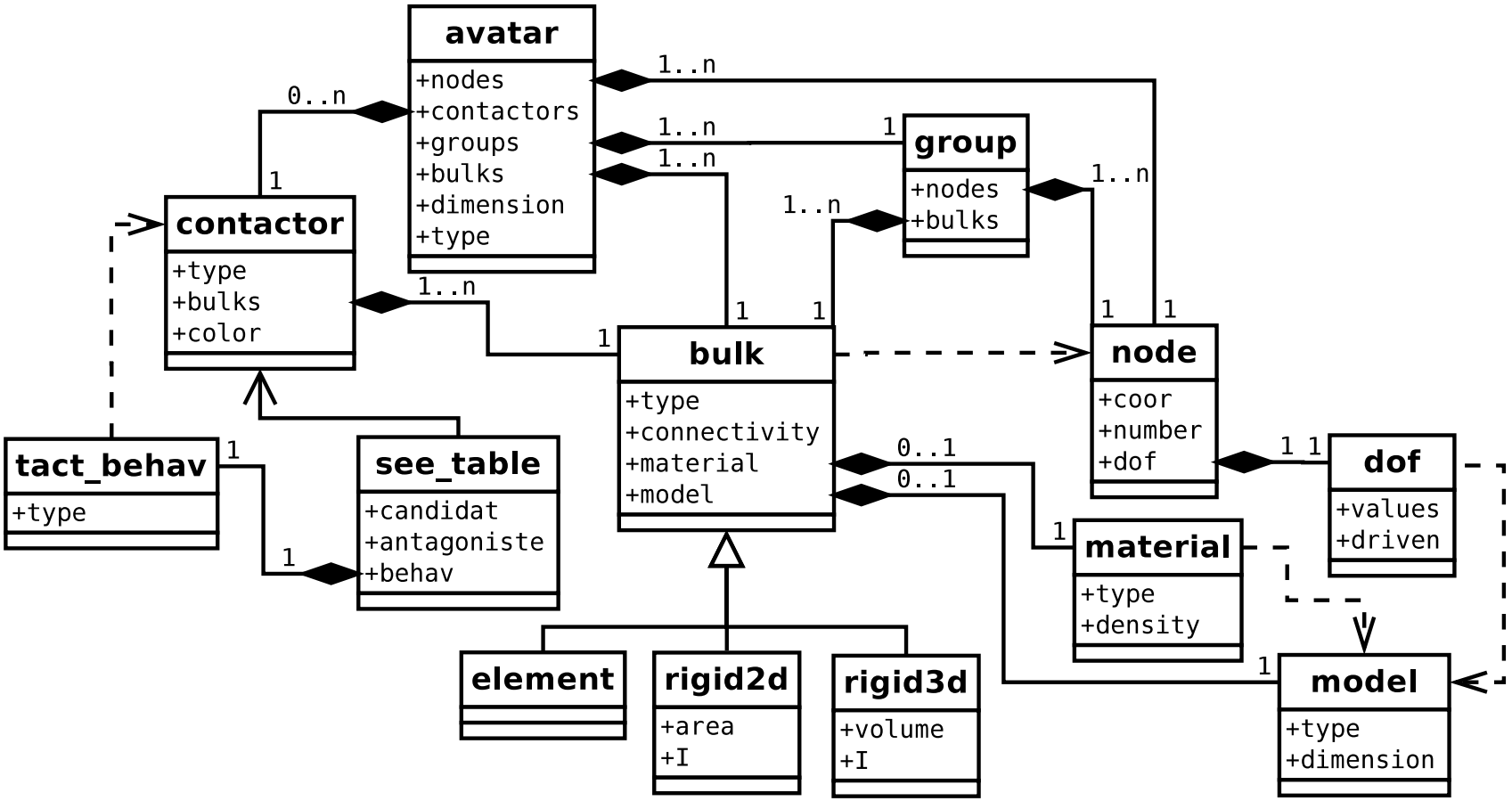
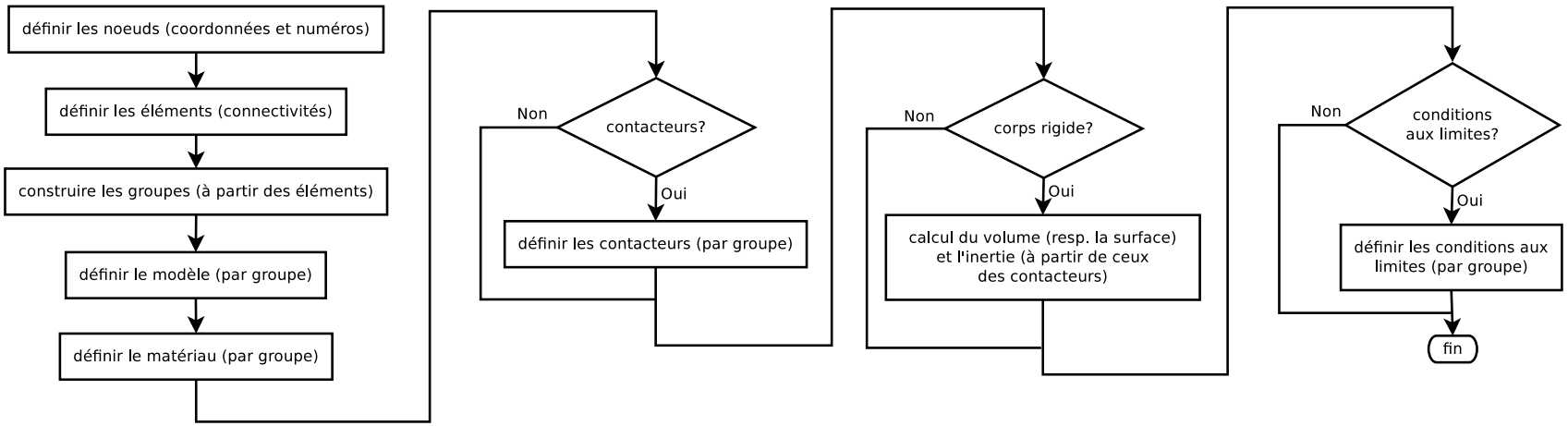


Schéma de construction d'un échantillon

- définition de la dimension considérée (*i.e.* 2D ou 3D),
- définition des conteneurs (modèles, matériaux, avatars, *etc*),
- définition des modèles et des matériaux,
- définition de chaque avatar :



- définition des lois d'interaction et des tables de visibilité,
- remplissage des conteneurs,
- écriture des fichiers.

Schéma de construction d'un échantillon

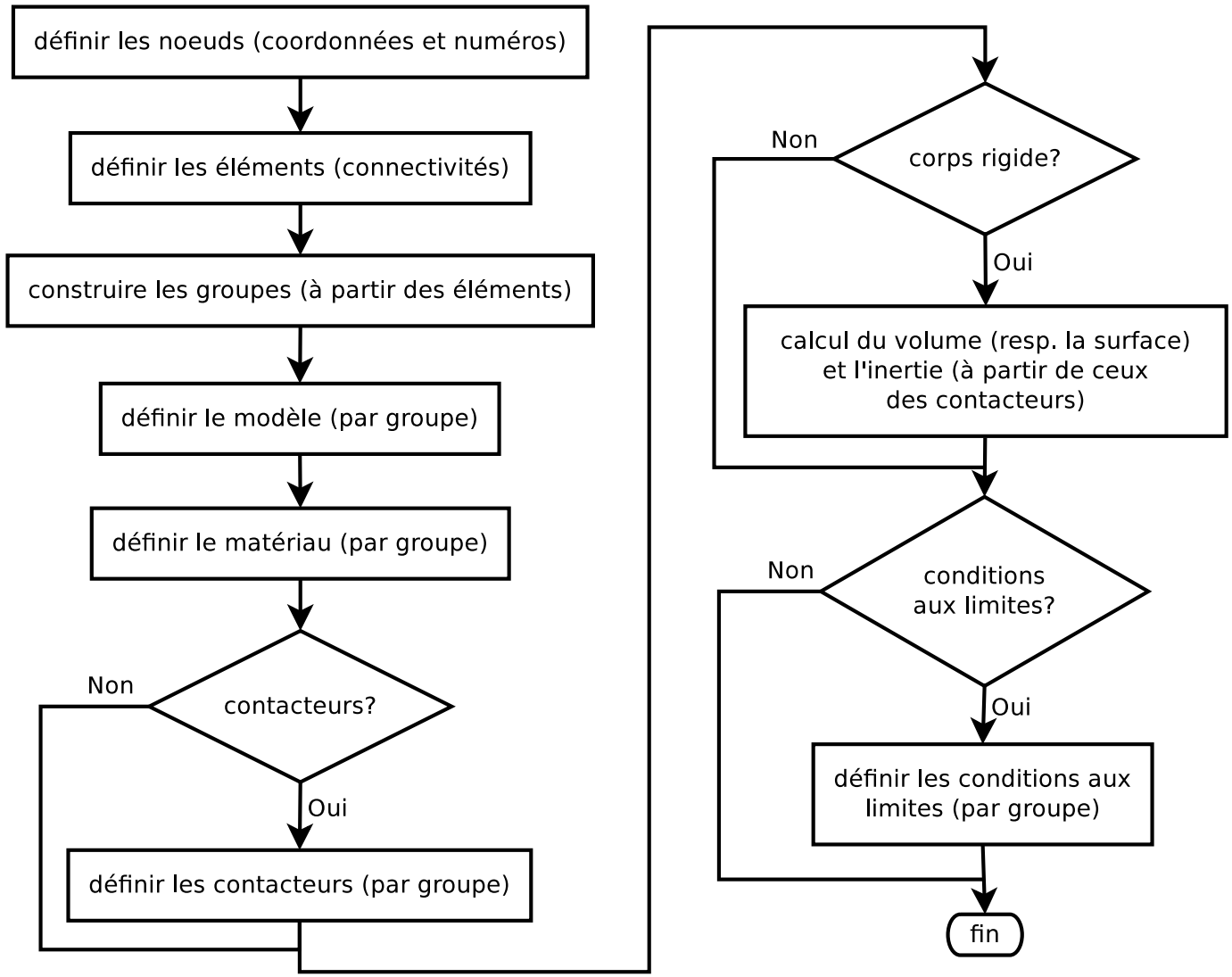
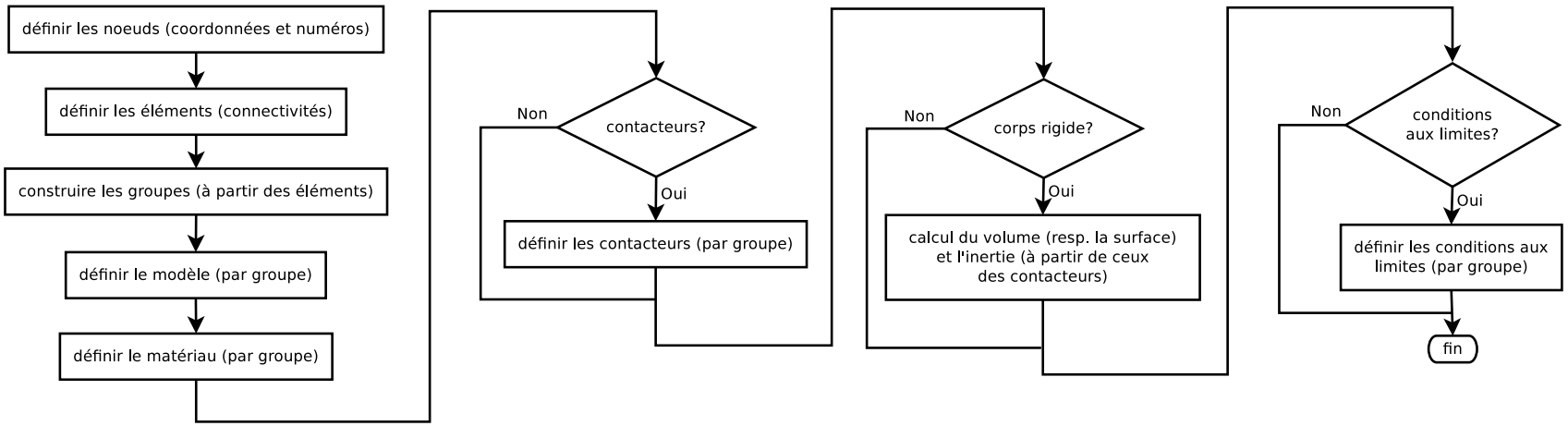


Schéma de construction d'un échantillon

- définition de la dimension considérée (*i.e.* 2D ou 3D),
- définition des conteneurs (modèles, matériaux, avatars, *etc*),
- définition des modèles et des matériaux,
- définition de chaque avatar :



- définition des lois d'interaction et des tables de visibilité,
- remplissage des conteneurs,
- écriture des fichiers.