Rapport de Projet Industriel de Fin d'Etude

Modélisation numérique de la morphogenèse épithéliale

Max SONZOGNI

Tuteurs

Patrick CAÑADAS

Julien AVERSENG

Laboratoire de Mécanique et de Génie Civil

MI

08/02/2019

Table des matières

Remerciements

Je tiens à remercier mes tuteurs de projet, Patrick Cañadas et Julien Averseng, pour m'avoir permis de travailler sur ce sujet. N'ayant étudié que très peu de biologie, j'ai pu découvrir de nombreux aspects du développement cellulaire qui m'étaient inconnus jusqu'alors. Les idées et conseils qu'ils ont pu me fournir tout le long du projet m'ont été d'une grande aide pour établir un modèle de simulation valable et facilement compréhensible.

Je veux également remercier Rémy Mozul, sans qui ce projet serait resté bloqué à l'étape initiale. Son expertise du code LMGC90 m'a permis de construire un modèle bien plus structuré que ce que j'aurais été en mesure de réaliser seul.

1 Introduction

Un épithélium est un tissu cellulaire formé par des cellules rangées de manière compactée (Figure 1), avec souvent très peu de marge de manœuvre pour se déplacer entre elles, appelées cellules épithéliales. La morphogenèse épithéliale constitue donc l'ensemble des étapes du développement d'un épithélium.

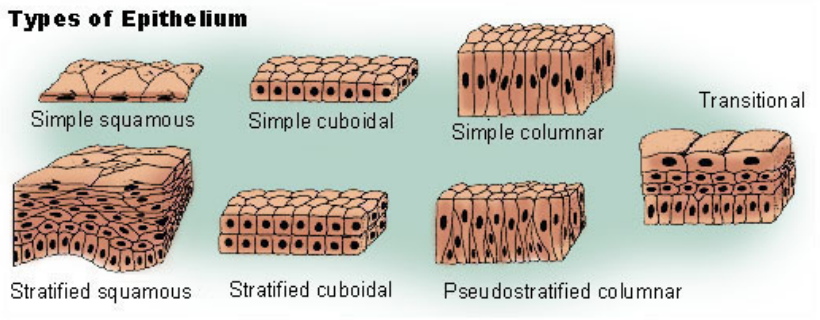


Figure 1: Différents formes d'épithéliums (source : Wikipedia)

Il existe deux types d'épithéliums : les prolifératifs et les non-prolifératifs. Les épithéliums prolifératifs se forment principalement par multiplication cellulaire : une fois que les cellules atteignent un certain volume, elles peuvent se diviser en de nouvelles cellules plus petites avec les mêmes caractéristiques, qui vont alors grandir puis se multiplier, et ce de manière quasi-infinie. Ces cellules peuvent aussi subir une mort cellulaire programmée, ou '*suicide cellulaire*', appelée l'apoptose : elles commencent à perdre en volume pour petit à petit finir par disparaître.
De l'autre côté, les épithéliums non-prolifératifs ne se forment que par dépôts successifs de cellules par une source extérieure, processus que l'on appelle accrétion. Ces cellules peuvent grandir individuellement, mais contrairement aux cas prolifératifs, elles ne peuvent pas se multiplier et ne semblent pas subir d'apoptose.
L'objectif est donc d'étudier chacun des deux types de morphogenèse épithéliale et de simuler les deux modes de développement.

Dans la plupart des simulations numériques, seul un aspect monocouche des épithéliums est étudié. Cela permet principalement d'avoir des systèmes moins complexes à gérer, car c'est surtout la forme externe que prennent ces tissus qui est utilisée par les biologistes pour caractériser leur topologie. Passer sur un modèle multicouches ou en amas 3D se rapprocherait plus de la réalité, mais demanderait aussi beaucoup plus de ressources et de temps de calcul.

De précédents travaux réalisés au LMGC ont tenté de reproduire le développement de ces tissus : Une thèse écrite en 2012 par Yoann Chélin [1] y décrivait les différentes étapes permettant la création d'un modèle numérique de morphogenèse épithéliale. Cette thèse a ensuite abouti sur plusieurs articles dont un publié en 2013 pour le Colloque National en Calcul des Structures [2].

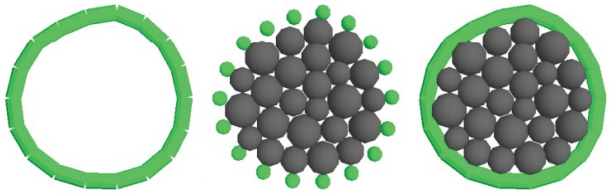


Figure 2: Modèle de cellule développé par Yoann Chélin sous ToyGL

Une approche granulaire de la simulation a été tentée à l'aide du code ToyGL, développé par l'équipe Conception et Structures du LMGC[1]. Une cellule est modélisée par 2 types d'éléments (Figure 2): des

[1] `http://www.lmgc.univ-montp2.fr/~averseng/JA/ToyGL.html`

grains internes de plusieurs tailles différentes qui modélisent le cytoplasme, et des grains membranaires, tous de même taille. Ces derniers sont liés par des câbles visco-élastiques, et forment ainsi la membrane externe de la cellule.
Ce modèle comportait toutes les étapes de la vie d'une cellule épithéliale : croissance, division et apoptose. Il était donc possible de créer des épithéliums à partir de différentes configurations initiales et même de les appliquer sur diverses surfaces, comme visible dans la Figure 3, et ce pour des modèles prolifératifs ou non.

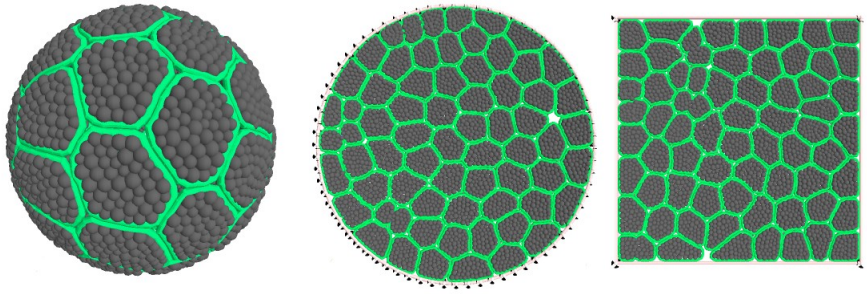


Figure 3: Modèle multicellulaire développé par Yoann Chélin sous ToyGL appliqué sur différentes formes géométriques (sphère, cercle, carré)

Cependant, cette thèse avait pour but premier de créer une simulation numérique avec une vision plutôt biologique de la chose. Pour étudier comment la mécanique influe sur la morphogenèse des tissus épithéliaux, il fallait changer de code de calcul et passer sur un code plus spécialisé dans les études en milieu granulaires.

C'est lors d'un précédent PIFE, réalisé en 2017 par Franck Richard [3], qu'ont été posées les bases de ce modèle sous LMGC90[2]. La structure granulaire est récupérée, avec des grains internes cytoplasmiques et d'autres grains externes pour la membrane. Les éléments câbles sont supprimés (car difficiles à mettre en place dans LMGC90) et sont remplacés à la place par des liaisons de type ressort visqueux.

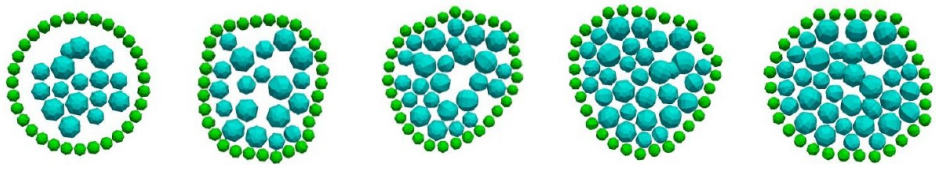
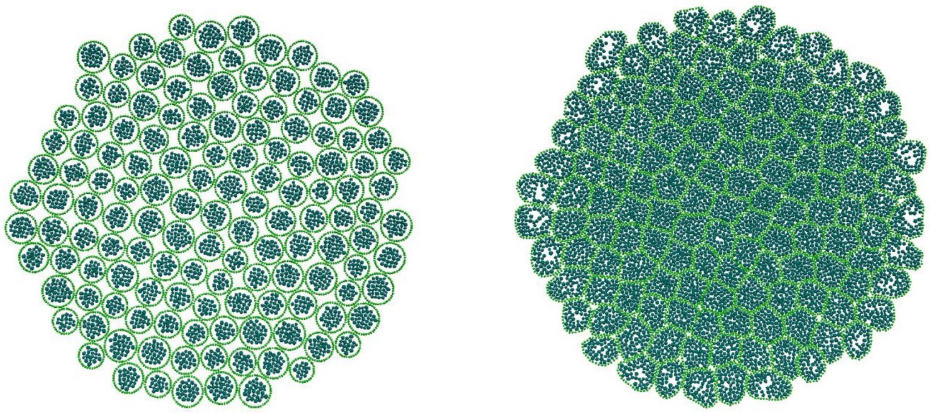


Figure 4: Modèle de croissance développé par Franck Richard sous LMGC90 pour une seule cellule

Le modèle qui a été réalisé ne prenait en compte que la morphogenèse de tissus non-prolifératifs. Ainsi, quelques tissus épithéliaux ont pu être réalisés en partant d'une multitude de petites cellules et en les faisant grandir (c.f. Figure 5). Le but premier de ce PIFE était de valider les observations faites à partir de cas réels et dans la simulation de Yoann Chélin concernant la répartition du nombre de voisins qu'avait chaque cellule dans un épithélium pour un modèle en accrétion. Les simulations avaient alors confirmé ces résultats, comme la Figure 6 le démontre.
Cependant, ce programme avait quelques difficultés concernant la croissance cellulaire : en effet, quand une cellule seule était étudiée, sa croissance était rendue infinie, et il arrivait forcément à un moment que l'espace entre les sphères de paroi soit trop grand et donc les sphères de cytoplasme pouvaient s'échapper de la cellule.

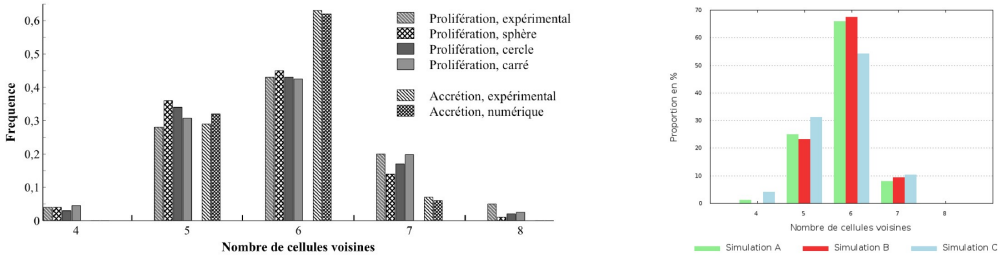
L'objectif serait donc dans un premier temps de corriger ces erreurs dans la croissance des cellules, puis ensuite d'y rajouter les étapes de division et d'apoptose pour pouvoir simuler la morphogenèse d'épithéliums prolifératifs.

[2] `https://git-xen.lmgc.univ-montp2.fr/lmgc90/lmgc90_user/wikis/home`



(a) Etat initial (b) Etat final

Figure 5: Modèle de croissance développé par Franck Richard sous LMGC90 avec 136 cellules



(a) Mesures de Yoann Chélin (prolifération et accrétion) (b) Mesures de Franck Richard (accrétion)

Figure 6: Proportions de cellules voisines dans un épithélium

2 Description du programme

Comme décrit précédemment, le premier programme réalisé par Franck Richard dans le cadre de son PIFE avait été créé pour réaliser des tissus par accrétion, c'est à dire en partant d'un nombre fini de cellules. Seul l'ajout de sphères de cytoplasme à l'intérieur des cellules était nécessaire, car il était supposé qu'elles n'atteindraient pas un volume suffisamment grand pour devoir se diviser ou augmenter le nombre de sphères sur leur paroi. Ce dernier point posait problème, comme décrit précédemment, et devait donc être repris pour pouvoir créer un modèle prolifératif.

De plus, l'accès aux différentes caractéristiques de chacune des sphères posait également problème. Le choix a donc été fait de mettre en place des listes qui vont récupérer différents attributs concernant chacun des corps. Il y en a pour l'instant 3 qui sont implémentées (voir Figure 7):

- `list_cell` = [`num`, [`sph_paroi`], `phase`, `cdg`, [`iter_div`, `id_div1`, `id_div2`]] la liste des cellules avec `num` son numéro (1,2,3 ...), `sph_paroi` la liste des numéros des sphères de paroi qui composent sa membrane (à noter qu'elles sont rangées de sorte à parcourir la membrane dans le sens anti-horaire), `phase` la phase dans laquelle se trouve la cellule (0 = croissance, 1 = division, 2 = apoptose) et `cdg` son centre de masse. La liste en cinquième position contient les informations nécessaires lors de la division de la cellule, à savoir l'itération de division actuelle et l'identifiant des deux sphères choisies comme points de repère (si `phase` $\neq 1$, ils sont tous égaux à -1).

- `list_sph_paroi` : [`num`, `nom`, `visibilite`] la liste de toutes les sphères de paroi
avec `num` le numéro de la sphère (1,2,3 ...), `nom` la retranscription de son numéro sous la forme d'une chaîne de 5 caractères (1 $\rightarrow$ "00001"), et la variable `visibilite` qui permet de savoir si la sphère est visible ou non (1 si visible, -1 sinon)

- `list_sph_cyto` : [`num`, `visibilite`] la liste de toutes les sphères de cytoplasme

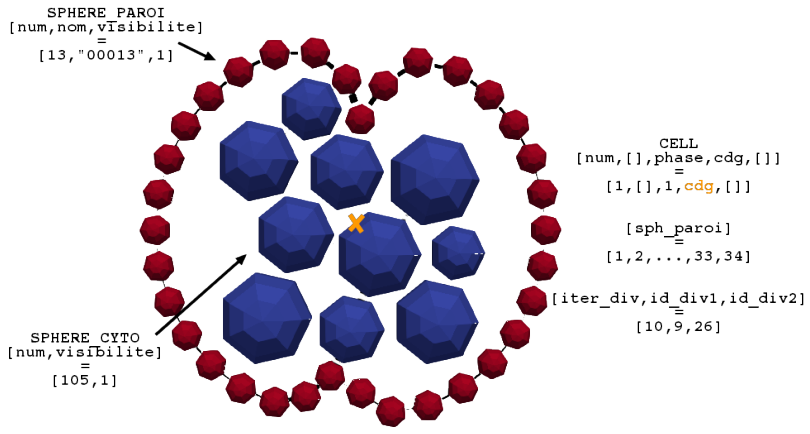


Figure 7: Listes utilisées pour une cellule initiale à 34 sphères de paroi lors de sa division

Avec l'aide de Rémy Mozul, le code a été repris et restructuré. Il a été choisi de le scinder en 2 programmes distincts : un pour gérer la structure initiale de la cellule (génération et positionnement des sphères, création des contacts), et un autre qui s'occupe d'effectuer tous les calculs.

2.1 Fichier de génération

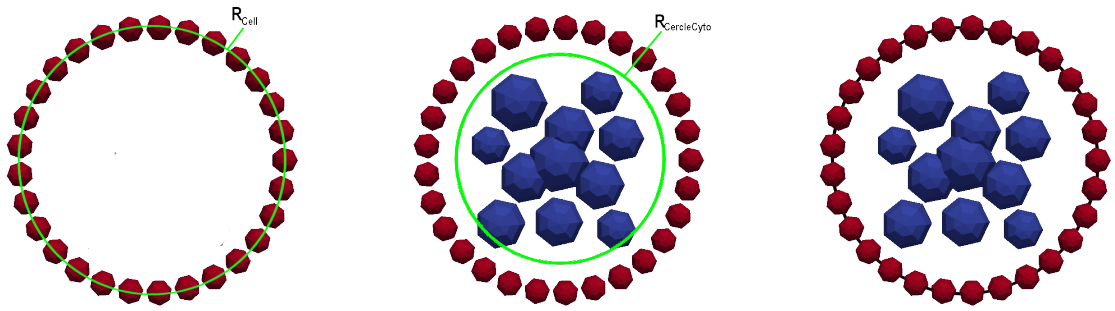
Le fichier de génération contenu dans l'Annexe A permet de mettre en place tous les éléments qui seront utilisés pendant le calcul et de leur donner une configuration initiale.

Dans un premier temps, l'utilisateur spécifie quelles seront les dimensions et les caractériques intrinsèques à chacun des matériaux au travers de diverses variables.
Les valeurs utilisées dans les simulations suivantes ont été reprises du Tableau 3-3 (p.51) de la thèse de Yoann Chélin. Certaines caractéristiques, comme par exemple celles concernant les éléments de type câbles, ont été retirées, pour ne garder au final que celles du tableau 1.

Cellule	Rayon initial = 50 μm
Grain de paroi	Rayon = 4 μm
Grain de cytoplasme	Rayon = 6-12 μm
Liaison paroi-paroi	Rigidité = 10 N.m^{-1} Viscosité dynamique = 1 N.m^{-1}.s Frottement dynamique = 0.1 Longueur effective < $4R_{Paroi}$
Contact cytoplasme-cytoplasme	Rigidité = 1 N.m^{-1} Frottement dynamique = 0.1 Distance d'activation : < $10^{-2}R_{CytoMin}$
Contact cytoplasme-paroi	Rigidité = 0.1 N.m^{-1} Frottement dynamique = 0.1 Distance d'activation : < $10^{-2}R_{CytoMin}$

Tableau 1: Récapitulatif des différents paramètres utilisés dans les simulations

Le nombre total de sphères de membrane et de sphères de cytoplasme doit être spécifié manuellement au travers de 2 variables, respectivement `nb_paroi` et `nb_cyto`. Ceci permettra alors d'avoir un critère d'arrêt, car le programme s'arrêtera quand toutes les sphères seront en cours d'utilisation et qu'il faudra en rajouter une nouvelle. Il est donc important d'en choisir un nombre assez important, mais il faut aussi faire attention à ce que la surcharge d'objets ne ralentisse pas trop le calcul.



(a) Mise en place de la membrane (b) Ajout du cytoplasme (c) Création des liaisons de paroi

Figure 8: Déroulement du fichier de génération

L'utilisateur peut choisir si les sphères de paroi seront espacées ou non lors de leur positionnement sur la périphérie de la cellule initiale. Il suffit pour ça de spécifier une valeur dans la variable `gap_ini`, qui dans notre cas sera de 1 μm.

Connaissant le rayon initial de la cellule, la taille des sphères de paroi et l'espace entre chacune d'entre elles au départ, un nombre suffisant de sphères est créé, puis elles sont positionnées sur le cercle délimitant la surface extérieure de la cellule (voir Figure 8a). Elles sont alors ajoutées à la liste des sphères de paroi avec un indice de visibilité de 1. Un nouveau set de sphères de paroi est ensuite crée et elles sont placées au centre du cercle. Elles sont alors enregistrées dans la liste des sphères de paroi avec cette fois-ci un indice de visibilité de -1.

En plus d'être définies comme des sphères (contacteur `SPHER`), les sphères de paroi possèdent un contacteur de type point 3D (`PTPT3`). Ces contacteurs permettent de déterminer quels types de liaisons il existe quand différents corps sont en contact. Les lois d'interactions seront précisées par la suite.

Ensuite ce sont les sphères de cytoplasme qui sont générées. A partir des différent rayons, un nouveau cercle est défini, dont le rayon est égal à :

$$R_{CercleCyto} = R_{Cell} - R_{Paroi} - R_{CytoMax}$$

Les fonctions `granuloRandom` et `depositInDisk2D` de LMGC90 permettent alors de placer des sphères de cytoplasme de manière aléatoire dans ce cercle en connaissant les rayons minimaux et maximaux de ces sphères (voir Figure 8b). Le numéro de ces sphères est alors inscrit dans la liste des sphères de cytoplasme, avec un indice de visibilité de 1. Tout comme pour les sphères de paroi précédemment, de nouvelles sphères de cytoplasme sont créées, installées au centre de la cellule et leur numéro est ensuite reporté dans la liste des sphères de cytoplasme, avec ici un indice de visibilité de -1.

Les sphères de cytoplasme possèdent elles 2 contacteurs de type `SPHER` qui seront nécessaires pour la mise en place des lois d'interactions.

Pour savoir comment ces sphères devront réagir entre elles, il va de soi de devoir définir leurs lois d'interactions. Il y en a 3 types qui sont utilisés dans ce programme (Figure 9):

Le modèle de Kelvin-Voigt (`VOIGT_ROD`) est utilisé pour relier entre elles les sphères de paroi. Il est similaire à un ressort avec de l'écoulement visqueux, ce qui permet d'avoir des liaisons fixes entre les sphères en contact jusqu'à une certaine distance critique (voir Tableau 1). Par contre, une fois cette distance dépassée, le lien entre les sphères est rompu et la cellule se '*brise*'.
Cependant, la relation `VOIGT_ROD` n'est possible qu'entre des éléments de type `PTPT3`. C'est pourquoi il a été nécessaire de rajouter ce type de contacteur aux sphères de paroi lors de leur création.

Les contacts réalisés par les sphères de cytoplasme sont de simples répulsions élastiques avec frottement (`ELASTIC_REPELL_CLB`). Ceci permet d'avoir de la répulsion entre les sphères et ainsi éviter les problèmes d'interpénétration. Le processus de repoussement s'active dès que les surfaces extérieures des sphères sont à une distance inférieure à celle spécifiée dans le Tableau 1.

En plus des deux précédentes relations, il existe entre chacune des sphères du frottement dynamique, défini par une loi de Coulomb (`IQS_CLB`), et dont le coefficient est identique pour chacun des corps en contact (ici $\mu = 0.1$).

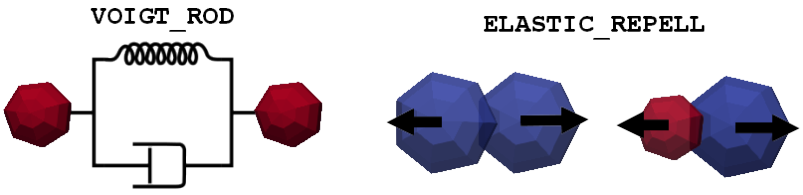


Figure 9: Types de relations entre les sphères

Pour maintenir la membrane de la cellule en place, il est nécessaire de créer manuellement les relations entre chacune des sphères de paroi. Autrement, dès qu'une sphère de paroi serait suffisamment proche d'une autre, une nouvelle liaison de type `VOIGT_ROD` se formerait et il y aurait comme un ressort entre elles.

Ainsi, chacune des sphères est liée avec ses voisins directs (Figure 8c), puis chaque liaison est stockée dans un fichier nommé `PTPT3_NETWORK.DAT`. Les lignes de ce fichier sont de la forme suivante :

```
nom1 nom2 0 0 0 dist
```

avec `nom1` et `nom2` les noms (chaînes de caractères) des deux sphères liées et `dist` la distance à vide du ressort entre elles. A noter que, retranscrit en nombres, `nom1` doit toujours être inférieur à `nom2`, et ce pour des raisons de référencement à l'intérieur de LMGC90.
De plus, il faut préciser qu'avec une installation classique de LMGC90, l'utilisation d'un fichier de liaisons entre `PTPT3` externe ne prendra pas en compte la variable de distance à vide `dist`, car elle sera recalculée automatiquement à partir de l'état actuel par le code à chaque fois que le fichier sera appelé. Il a donc été nécessaire de modifier le code source de certains fichiers internes pour ignorer cette étape (contacter Rémy Mozul pour plus de détails).

Finalement, le programme va génèrer les fichiers contenant toutes les informations sur ce qui vient d'être construit, aussi bien les corps que les caractéristiques des contacts et des liaisons entre eux, et va les placer dans le dossier `DATBOX`.

2.2 Fichier de calcul

Le fichier de calcul varie suivant le modèle de morphogenèse désiré, et un exemple peut être trouvé dans l'Annexe B. Il commence toujours par la récupération de variables passées par l'intermédiaire du fichier de génération. Ensuite, ce sont de nouvelles variables permettant de contrôler les différents éléments du calcul qui sont spécifiées par l'utilisateur, tel que le temps maximal de simulation `tmax`.

Les fichiers écrits dans la programme de génération sont lus une première fois par le programme de calcul, qui va alors charger dans la mémoire de l'ordinateur tous les éléments créés auparavant. La liste des cellules est ainsi initialisée avec une seule entrée, celle de la cellule de départ. On lui accorde le numéro 1, la liste des sphères de sa paroi est spécifiée et son centre de masse est centré sur l'origine. Les sphères de paroi et de cytoplasme inutilisées au départ sont rendues invisibles, et le fichier de relations sur les sphères de paroi est appelé, au travers de la commande `PTPT3_LoadNetwork()`. La boucle de calcul principale peut alors démarrer.

Tous les `iter_ajout` pas de temps, une nouvelle sphère de cytoplasme est ajoutée dans les cellules. Le programme va alors parcourir la liste des sphères de cytoplasme et en trouver une dont l'indice de visibilité sera de -1. Elle sera alors positionnée au centre de masse de la cellule et la valeur de son indice de visibilité sera alors remplacée par 1.

A chaque nouveau pas de temps, le programme va déterminer quelles sont les forces qui sont en jeu, qu'elles soient externes aux corps ou internes. La vitesse de chacun d'entre eux va aussi être prise en compte, puis chaque contact est identifié. Toutes ces informations vont alors être utilisées pour calculer la solution, au travers de la méthode 'Non Smooth Contact Dynamics' [4]. Cette méthode permet de trouver la solution d'un système mécanique où les corps sont nombreux et supposés rigides, et donc où les effets de frottement et de friction sont importants (notamment en milieu granulaire).

La solution ainsi déterminée donne alors le nouvel état du système au temps suivant, qui est alors retranscrit dans des fichiers de données qui seront relus pour effectuer le prochain calcul. Des fichiers de visualisation (`.vtk`) sont aussi écrits à chaque pas de temps pour permettre de mieux observer l'évolution du système.

Une fois tous les calculs effectués pour la configuration actuelle, le centre de gravité de chaque cellule est mis à jour. Pour l'instant il représente la position moyenne de toutes les sphères de paroi, et correspond donc au point '*central*' de la cellule. Cette définition pourrait être améliorée en y incorporant la contribution des sphères de cytoplasme contenue à l'intérieur de la cellule et en prenant en compte leurs variations de taille.

Les étapes présentées ci-dessus sont celles nécessaires pour réaliser la simulation d'un épithélium non-prolifératif ; l'agrandissement de la membrane et la division cellulaire ne sont donc pas nécessaires pour ce type de cas-ci. Ces étapes étant assez complexes à mettre en oeuvre, elles seront toutes les deux détaillées dans leurs chapitres respectifs.

3 Croissance de la membrane

Le premier problème que posait le programme de départ était que la croissance de la cellule ne pouvait pas être infinie. En effet, plus le nombre de sphères de cytoplasme augmentait à l'intérieur de la cellule, plus le volume de la cellule était important, et donc plus l'espace entre chaque sphère de paroi était grand. Il arrivait donc forcément un moment où cet espacement soit permettait de laisser passer les sphères de cytoplasme à travers la membrane, soit était suffisament large pour qu'il dépasse la distance limite de liaison et donc que la membrane se brise.

Dans la thèse de Yoann Chélin, l'ajout de sphères de cytoplasme est fait à intervalles réguliers. Les éléments câbles représentant la paroi se tendaient alors petit à petit au fur et à mesure que le nombre de sphères internes augmentait. Ces éléments étaient pilotés grâce à leur longueur, mais aussi avec la tension interne qu'ils subissaient.
Ainsi, si la tension d'un câble était inférieure à une valeur donnée, sa longueur libre était diminuée. Inversement, si sa tension était supérieure à une valeur donnée, sa longueur libre était augmentée. Si la longueur d'un élément était trop grande, il se scindait en deux éléments de même longueur et de même tension. De l'autre côté, si deux éléments successifs avaient une longueur trop faible, ils fusionnaient pour n'en former plus qu'un seul.

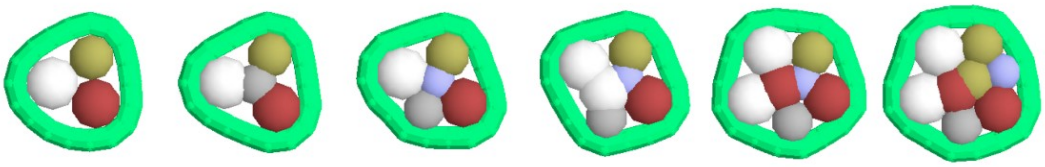
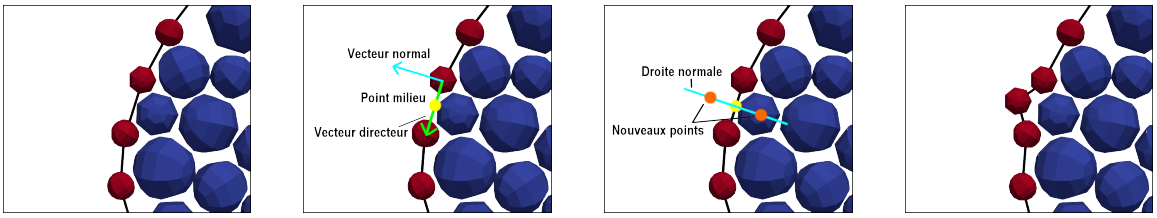


Figure 10: Modèle de croissance développé par Yoann Chélin sous ToyGL

L'utilisation de LMGC90 étant différente de celle de ToyGL, du fait de l'absence d'élément câbles, il a fallu se restraindre à piloter les relations de paroi par leur seule longueur. Le processus de croissance ainsi implémenté est donc décrit ci-dessous.

Dans un premier temps, la liste des sphères de paroi de la cellule va être parcourue et, pour chaque sphère de paroi, la distance qui la sépare de ses voisines directes va être calculée. Si cette distance est supérieure à une valeur critique, spécifiée par la variable `dist_max`, l'identifiant du couple de sphères dans la liste des sphères de sa cellule est alors enregistré dans une autre liste, appelée `list_ajout`.

Pour chaque entrée de cette liste, une sphère de paroi qui était invisible à ce moment de l'exécution du code est alors récupérée, puis rendue visible pour ensuite être placée près du couple de sphères correspondant. Il existe différentes méthodes pour rajouter cette nouvelle sphère entre les anciennes: elle peut soit être directement placée à mi-distance entre les deux, soit être positionnée de façon à garantir la croissance de la cellule. Cette dernière méthode fonctionne comme suit :



(a) Détection d'un espacement critique

(b) Calcul des vecteurs et du point milieu

(c) Positionnement des nouveaux points

(d) Ajout de la nouvelle sphère sur la paroi

Figure 11: Etapes de l'ajout d'une sphère sur la membrane pendant la phase de croissance

Une fois que les deux sphères entre lesquelles la nouvelle doit être rajoutée sont identifiées (Figure 11a), leur position est enregistrée et le vecteur allant de la première vers la seconde est créé. Notez que ce vecteur ira toujours dans le sens anti-horaire, du fait que dans la liste des sphères de chaque cellule elles sont rangées dans cet ordre-là. A partir de ce vecteur, ses deux vecteurs normaux sont créés (Figure 11b), car il n'est pas possible de déterminer clairement lequel pointe vers l'intérieur ou vers l'extérieur de la cellule. A partir de là, la position au milieu des deux sphères est repérée, et deux nouveaux points sont établis en déplaçant le point précédent par les deux vecteurs normaux (Figure 11c). La distance entre ces nouveaux points et le centre de masse de la cellule est alors calculée : le point le plus éloigné du centre de masse est alors celui qui aura été déplacé suivant le vecteur normal sortant, c'est donc à cet endroit que sera placée la nouvelle sphère (Figure 11d).

Le fait que ce nouveau '*morceau*' de membrane soit situé à l'extérieur de la cellule garantit le fait que la cellule aura toujours une forme convexe lors de sa croissance, évitant ainsi des problèmes de recoupement éventuels entre les liaisons existantes.

La distance sur cette normale peut être contrôlée au travers de la variable `dist_norm`. Si `dist_norm`$>$ 0, alors le nouveau point sera situé à `dist_norm` fois la longueur de la distance entre les sphères sur le vecteur normal sortant. Le même résultat est obtenu si `dist_norm` < 0, car que la valeur entrée soit positive ou négative, deux points sont créés sur le vecteur normal, et c'est celui qui sera le plus éloigné du centre de masse qui sera conservé. Par contre, si `dist_norm` $= 0$, le point sera alors placé pile entre les deux sphères, ce qui reviendrait à utiliser la méthode basique de placement à mi-chemin.

Pour chacun des couples de sphères identifiés, la relation correspondante est identifiée dans le fichier `PTPT3_NETWORK.DAT`, puis '*notée*' comme étant à supprimer. Les nouvelles relations reliant la nouvelle sphère aux anciennes sont créées et restranscrites dans le fichier de relations. Ce fichier est finalement réécrit et la détection des nouveaux contacts est exécutée.
La longueur à vide qui est spécifiée dans les nouvelles relations est la même que celle que des autres sphères de paroi. Cela permet de ne pas les différencier par rapport aux liaisons '*originelles*'.

L'ajout d'éléments de paroi n'est pas suffisant à lui seul pour simuler entièrement un épithélium prolifératif : il est nécessaire de rajouter une étape où les cellules vont se multiplier, pour à partir d'une seule cellule obtenir tout un tapis de cellules.

4 Division cellulaire

Il a été démontré en 2009 que la division cellulaire modifiait la topologie de l'épithélium et joue donc un rôle important dans sa morphogenèse [5]. Il est ainsi tout à fait logique de vouloir étudier l'impact de la division cellulaire sur la formation des épithéliums au travers de simulations numériques.

Dans le monde du vivant, la division cellulaire se déroule en 3 phases principales : la croissance, la réplication de l'ADN, puis la séparation (c.f. Figure 12). C'est cette dernière étape, appelée la mitose, qui va être modélisée ici.

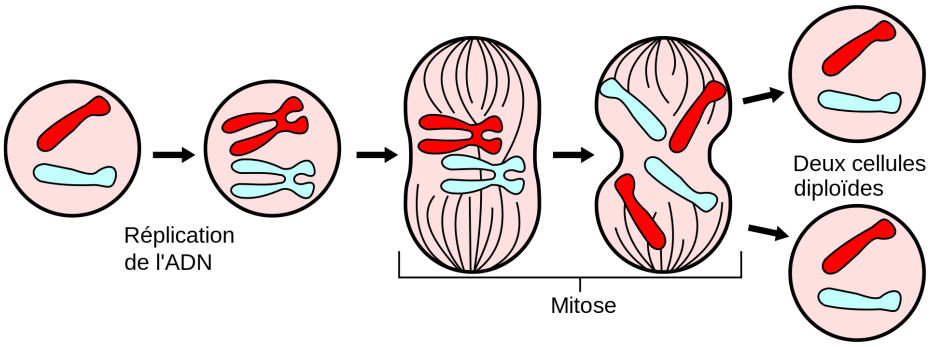


Figure 12: Principe de la division cellulaire (source : Wikipedia)

Ainsi, à partir d'une cellule mère, deux cellules filles sont créées, possédant les mêmes caractérisques que leur mère. Réalisée plusieurs fois de suite, ceci permet d'avoir une grande quantité de cellules identiques en partant d'un nombre initial restreint.

Dans sa thèse, Yoann Chélin s'est basé sur des observations faites sur des cellules animales et végétales, en collaboration avec des biologistes. Il a pu alors établir un processus de division proche de la réalité (Figure 13).

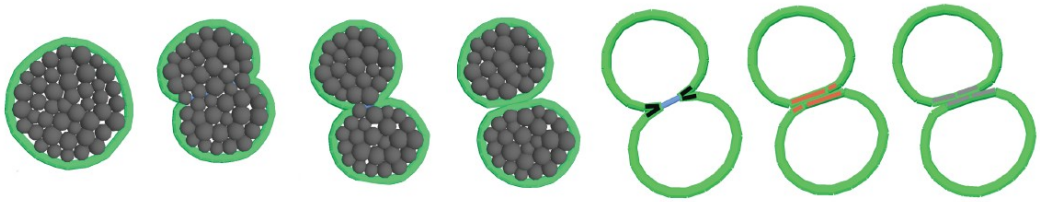
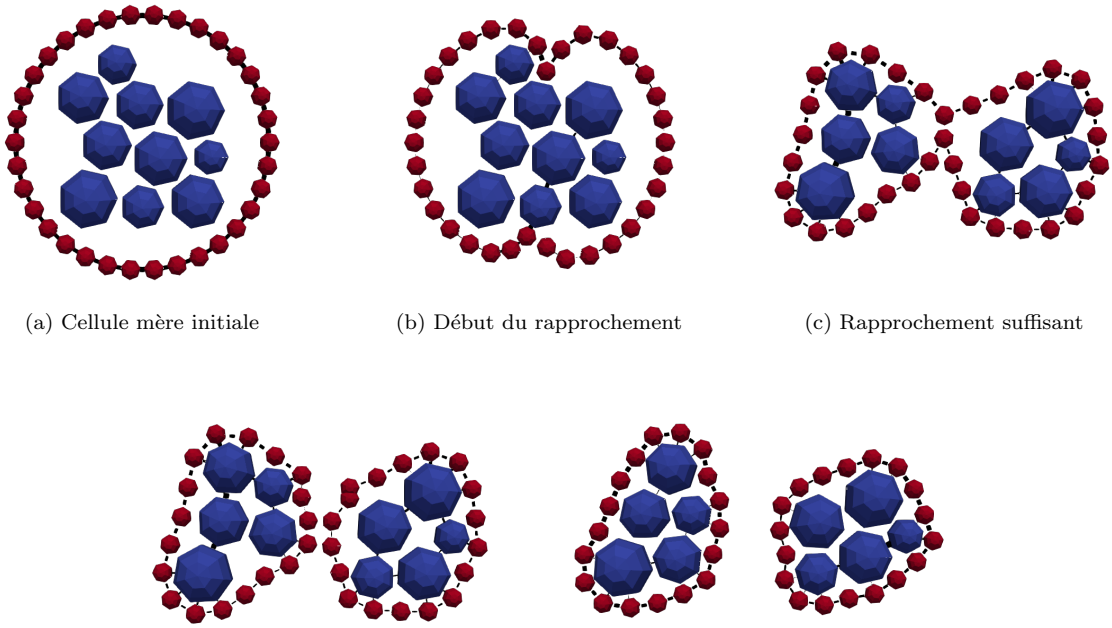


Figure 13: Etapes de division du modèle développé par Yoann Chélin sous ToyGL

Une fois la cellule à diviser identifiée, un élément de type câble, d'un autre matériau que celui utilisé pour la membrane, est tiré entre deux grains de membrane de chaque côté de la cellule. La longueur de ce câble est alors progressivement diminuée jusqu'à atteindre une valeur seuil. Les deux grains et les relations qu'ils effectuaient sont alors supprimés, puis remplacés par deux nouveaux grains, un sur chaque '*nouvelle*' cellule, et les liaisons permettant de refermer ces cellules sont construites.

Pour l'instant, le début de la division peut démarrer dès que la cellule atteint un certain nombre de sphères de paroi, nombre spécifié par la variable `paroi_max`. La probabilité que la cellule entre alors en phase de division est déterminée par la variable `proba_div` qui, en reprenant les valeurs données dans la thèse, peut varier entre 0 et 0.3.
Si ce critère est rempli, la variable `phase` de la cellule est passée à 1 et une sphère est alors choisie au hasard sur la paroi. Son opposée est déterminée en prenant en compte le nombre de sphères sur la paroi et en choisissant celle qui est décalée '*à l'opposé*' de la liste des sphères de paroi. Ainsi, la division coupera la cellule mère en deux cellules filles qui auront le même nombre de sphères de membrane à une unité près.

Pour rapprocher les deux côtés de la cellule, il n'est pas possible avec LMGC90 de créer de câble comme dans ToyGL. Une force est alors appliquée sur chacune des deux sphères tout le long du rapprochement et est renouvelée à chaque pas de temps. Elle est dirigée de sorte à ce que les deux sphères se dirigent l'une vers l'autre, et ce grâce au vecteur directeur `vectD`. L'intensité de cette force est proportionnelle à une valeur spécifiée par l'utilisateur, mais aussi au temps passé pour effectuer la division ; la force va alors augmenter en intensité au fur et à mesure que la division dure, évitant ainsi de se retrouver dans une situation fixe. Ce type de cas est assez fréquent dans les épithéliums, du fait de la promiscuité dont font l'objet les cellules.



(a) Cellule mère initiale

(b) Début du rapprochement

(c) Rapprochement suffisant

(d) Ajout des nouvelles sphères et disparition des anciennes

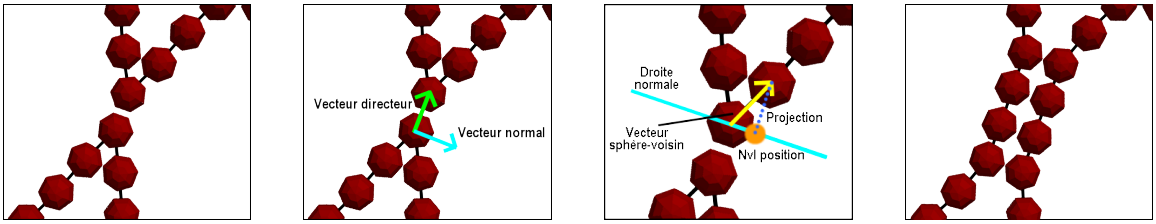
(e) Séparation en deux cellules filles

Figure 14: Division cellulaire obtenue avec le nouveau programme pour une cellule non pleine

Une fois que les deux sphères sont assez proches (Figure 14c), le calcul passe à une nouvelle étape. A côté de chacune des sphères qui ont été rapprochées, deux nouvelles sphères de paroi invisibles jusqu'alors sont rajoutées. Pour déterminer à quel endroit elles seront placées, plusieurs calculs sont nécessaires:
En premier lieu, la normale au vecteur directeur allant de l'une des sphères rapprochées à l'autre est construite (`vectN`), et la droite correspondante à ce vecteur est placée sur une de ces sphères. Ensuite, le vecteur allant de l'une de ces sphères vers une de ces deux voisines directes est établi (`vect12`). La position de la nouvelle sphère correspond alors au point représentant la projection sur la droite de ce vecteur (Figures 15c et 15d).

De nouvelles relations sont établies entre les sphères ajoutées et les voisines de celles qui ont été rapprochées. Les anciennes relations entre les sphères rapprochées et leurs voisines sont alors marquées comme étant à supprimer. Le numéro de chaque nouvelle sphère est ensuite stocké dans une liste temporaire (`list_ajout`) avec la valeur de sa projection (positive ou négative suivant le positionnement). Le processus est ainsi répété quatre fois au total.

Une fois chaque sphère mise en place et ses relations établies (Figure 14d), la liste `list_ajout` est triée par ordre croissant des valeurs des projections. Ainsi, les deux valeurs négatives, qui sont donc du même côté de la droite normale à la direction de rapprochement, sont clairement définies et permettent ainsi de facilement déterminer quelles sphères doivent être reliées pour terminer de construire la paroi



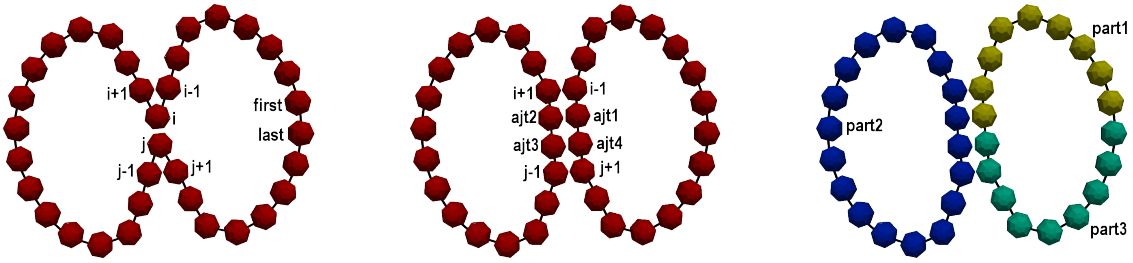
(a) Rapprochement des sphères (b) Création des vecteurs (c) Détermination de la nouvelle position (d) Ajout des nouvelles sphères

Figure 15: Etapes de la création et du positionnement des nouvelles sphères lors de la division cellulaire

de chaque cellule.

Tout comme pour la croissance, le fichier `PTPT3_NETWORK.DAT` est réécrit et la détection des contacts relancée. Les sphères servant à faire le rapprochement n'étant alors plus d'aucune utilité, elles sont finalement rendues invisibles.

Le cycle de division étant terminé, deux nouvelles cellules sont alors créées : la liste des cellules `list_cell` doit donc être mise à jour. La liste des sphères de paroi de la précédente cellule est alors découpée en trois parties (Figure 16c): une allant de la première sphère à une première sphère ajoutée ; une allant d'une deuxième sphère ajoutée à une troisième sphère ajoutée ; et une allant de la quatrième sphère ajoutée à la dernière sphère de la liste. Ainsi, une coupure est effectuée à chaque fois entre deux sphères ayant été ajoutées. Cela permet d'avoir une sphère dont la paroi est définie par la première et troisième parties, et une autre composée des sphères de la deuxième partie.



(a) Liste initiale (b) Ajout des nouvelles sphères (c) Découpage en 3 sous-listes

Figure 16: Formation des nouvelles listes de sphères de paroi pendant la division

Une nouvelle entrée est alors ajoutée à la liste, avec un nouveau numéro et une des deux nouvelles listes, tandis que l'ancienne cellule voit ses caractéristiques mises à jour. Elles sont toutes les deux redéfinies comme étant en phase de croissance, avec leur variable `phase` égale à 0.

Il faut cependant faire attention à ce que ce volume de la cellule ne soit pas trop élevé avant la division pour éviter que l'espacement entre les sphères de paroi ne dépasse pas la valeur critique pendant le processus. La forme que prend la cellule pendant la division n'est plus une forme convexe, et il y a des risques que sa structure soit compromise si une nouvelle sphère est ajoutée par la méthode de la normale sortante. C'est pourquoi il faut choisir la variable `paroi_max` telle que le nombre maximal de sphères sur la paroi ne soit pas trop grand. Dans la thèse de Yoann Chélin, ce nombre est de 46 ; il a été imposé comme valant 50 dans les simulations effectuées ici.

5 Conclusion

Le problème du programme précédent concernant l'agrandissement de la membrane cytoplasmique a été réglé, et la croissance cellulaire peut donc être simulée sans problèmes. Les critères rajoutés permettent de contrôler l'évolution de cette croissance, que ce soit sur la taille maximale que peut avoir la cellule ou sa vitesse de croissance.

L'ajout de la phase de division représente un grand pas dans la création du modèle. Il est désormais possible de générer une multitude de cellules à partir d'un nombre fini de cellules originelles. Avec le caractère aléatoire inclus dans cette phase, le nombre d'épithéliums différents créables à partir d'une seule cellule est quasi-infini.

Cependant, le travail effectué n'est pas suffisant pour réaliser une simulation complète d'un épithélium prolifératif. En effet, la phase d'apoptose doit encore être ajoutée, et avec elle s'ajoutent de nouveaux paramètres de contrôle, qui affecteront les sphères de cytoplasme cette fois-ci. Par exemple, il faudrait déterminer à l'intérieur de quelle cellule se trouve chaque sphère de cytoplasme active. Pour la division cellulaire, cela impliquerait de devoir chercher quelles sphères sont contenues à l'intérieur des cellules nouvellement créées, chose qui n'est possible avec la configuration actuelle qu'en regardant la position de chacune des sphères de cytoplasme visibles. Cette méthode serait très coûteuse en temps de calcul si un grand nombre de sphères sont actives.

De plus, certains aspects du processus de calcul, comme le choix des sphères au début de la phase de division ou le calcul du centre de masse des cellules, se sont pas totalement optimisés. Ces points pourront être revus dans un futur projet, qui aura alors pour but de les améliorer et de rajouter l'apoptose au modèle, afin de finalement avoir une simulation d'épithéliums prolifératifs sous LMGC90.

Bibliographie

[1] Yoann Chélin. *Développement d'un modèle mécanique et numérique de morphogenèse de tissus épithéliaux*. PhD thesis, 2012. Thèse de doctorat dirigée par Maurin, Bernard Mécanique et Génie civil Montpellier 2 2012.

[2] Yoann Chélin, Julien Averseng, Patrick Cañadas, and Bernard Maurin. Modèle granulaire de morphogenèse tissulaire. In *CSMA 2013*, pages 8 p., Clé USB, Giens, France, May 2013.

[3] Franck Richard. Modélisation et simulation numérique de la morphogenèse épithéliale. Rapport de Projet Industriel de Fin d'Etude, Polytech Montpellier, 2017.

[4] Michel Jean. The non-smooth contact dynamics method. *Computer Methods in Applied Mechanics and Engineering*, 177(3-4):235–257, 1999.

[5] W.T. Gibson and M.C. Gibson. Cell Topology, Geometry, and Morphogenesis in Proliferating Epithelia. *Current Topics in Developmental Biology*, 89:87–114, 2009.

Annexes

Annexe A Fichier de génération

```
#DEBUT FICHIER
import os
import math
import numpy as np
import shutil
from pylmgc90 import pre

if not os.path.isdir('DATBOX'):
    os.mkdir('DATBOX')

dim = 3
bodies = pre.avatars()
mat    = pre.materials()
svs    = pre.see_tables()
tacts = pre.tact_behavs()

#create materials
memb = pre.material(name='MEMBx',materialType='RIGID',density=1000.)
cyto = pre.material(name='CYTOx',materialType='RIGID',density=1000.)
mat.addMaterial(memb,cyto)
# create a model of rigid
mod = pre.model(name='rigid', physics='MECAx', element='Rxx3D', dimension=dim)

# propriétes
stiff_paroi = 1.e7
visc_paroi  = 1.e6
stiff_cyto = 1.e6
stiff_cytoparoi = 1.e5
# rayons
rcell  = 50.
rparoi = 4.
rcytomin = 6.
rcytomax = 12.
gap_ini = 1
#spheres a creer
nb_cyto = 500
nb_paroi = 100

# calcul du nombre de spheres sur la paroi de la cellule
circ_cell = 2. * math.pi * rcell
nb_paroi_ini = int( circ_cell / (2*rparoi+gap_ini) )
#initialisation liste spheres paroi
list_sph_paroi = []
#spheres paroi initiales
angle = 2. * math.pi / float( nb_paroi_ini )
for i in range(nb_paroi_ini):
    spher = pre.rigidSphere( r=rparoi, center=[rcell,0.,0.],
                             material=memb, model=mod, color='PAROI')
    spher.rotate(description='axis', alpha=i*angle, axis=[0., 0., 1.] , center=[0., 0., 0.])
    spher.addContactors('PT3Dx','PAROI')
```

```
    spher.imposeDrivenDof(component=3, dofty='vlocy')
    #spher.imposeDrivenDof(component=[1], dofty='force',ct=-200.)
    bodies.addAvatar(spher)
    list_sph_paroi.append([bodies.index(spher)+1,str(i+1).zfill(5),1])
#spheres paroi ajoutables
for i in range(nb_paroi-nb_paroi_ini):
    spher=pre.rigidSphere(r=rparoi, center=[0.,0.,0.],
                          material=memb, model=mod, color='PAROI', number=None)
    spher.addContactors('PT3Dx','PAROI')
    spher.imposeDrivenDof(component=3, dofty='vlocy')
    bodies.addAvatar(spher)
    list_sph_paroi.append([bodies.index(spher)+1,str(nb_paroi_ini+i+1).zfill(5),-1])

#creation cytoplasme
list_sph_cyto = []
r_list_sph_cyto=pre.granulo_Random(nb_cyto, rcytomin, rcytomax)
r_deposit_cyto = rcell-rparoi-rcytomax
[nb_cyto_ini, coor_sph_cyto]=pre.depositInDisk2D(r_list_sph_cyto, r_deposit_cyto,
                                                 deposited_radii=None, deposited_coor=None)
#spheres cyto ini
for i in range(nb_cyto_ini):
    spher=pre.rigidSphere(r=r_list_sph_cyto[i],
                          center=[coor_sph_cyto[2*i],coor_sph_cyto[2*i+1],0.],
                          material=cyto, model=mod, color='CYTOx', number=None)
    spher.translate(dx=-r_deposit_cyto,dy=-r_deposit_cyto,dz=0.)
    spher.addContactors('SPHER','CYTO2',byrd=r_list_sph_cyto[i])
    spher.imposeDrivenDof(component=3, dofty='vlocy')
    bodies.addAvatar(spher)
    list_sph_cyto.append([bodies.index(spher)+1,1])
#spheres cyto ajoutables
r_list_sph_cyto=pre.granulo_Random(nb_cyto-nb_cyto_ini, rcytomin, rcytomax)
for i in range(nb_cyto-nb_cyto_ini):
    spher=pre.rigidSphere(r=r_list_sph_cyto[i], center=[0.,0.,0.],
                          material=cyto, model=mod, color='CYTOx', number=None)
    spher.addContactors('SPHER','CYTO2',byrd=r_list_sph_cyto[i])
    spher.imposeDrivenDof(component=3, dofty='vlocy')
    bodies.addAvatar(spher)
    list_sph_cyto.append([bodies.index(spher)+1,-1])


#lois d'interaction
bl_pt3dx_paroi = pre.tact_behav(name='vgtr1', law='VOIGT_ROD', stiffness=stiff_paroi,
                                prestrain=-0.01, viscosity = visc_paroi )
tacts += bl_pt3dx_paroi
bl_spher_paroi = pre.tact_behav(name='iqsc0', law='IQS_CLB',
                                fric=0.1)
tacts += bl_spher_paroi
bl_spher_cyto  = pre.tact_behav(name='iqsc1',law='IQS_CLB',
                                fric = 0.1)
tacts += bl_spher_cyto
bl_spher_cyto2 = pre.tact_behav(name='elrpl',law='ELASTIC_REPELL_CLB',
                                stiffness=stiff_cyto,fric=0.1)
tacts += bl_spher_cyto2
bl_spher_cyto_paroi  = pre.tact_behav(name='iqsc2',law='IQS_CLB',
                                      fric = 0.1)
```

```
tacts += bl_spher_cyto_paroi
bl_spher_cyto_paroi2 = pre.tact_behav(name='elrp1',law='ELASTIC_REPELL_CLB',
                                      stiffness=stiff_cytoparoi,fric = 0.1)
tacts += bl_spher_cyto_paroi2

alert_spher_paroi = 1.e-2 * rparoi
alert_pt3dx_paroi = 4*rparoi
alert_spher_cyto  = 1.e-2 * rcytomax
alert_spher_cyto2 = 1.e-2 * rcytomax
alert_spher_cyto_paroi  = 1.e-2 * rcytomax
alert_spher_cyto_paroi2 = 1.e-2 * rcytomax

st_spher_paroi = pre.see_table(CorpsCandidat   ='RBDY3',candidat   ='SPHER',
                               colorCandidat   ='PAROI',behav=bl_spher_paroi,
                               CorpsAntagoniste='RBDY3',antagoniste='SPHER',
                               colorAntagoniste='PAROI',alert=alert_spher_paroi)
svs+=st_spher_paroi
st_pt3dx_paroi = pre.see_table(CorpsCandidat   ='RBDY3',candidat   ='PT3Dx',
                               colorCandidat   ='PAROI',behav=bl_pt3dx_paroi,
                               CorpsAntagoniste='RBDY3',antagoniste='PT3Dx',
                               colorAntagoniste='PAROI',alert=alert_pt3dx_paroi)
svs += st_pt3dx_paroi
st_spher_cyto  = pre.see_table(CorpsCandidat   ='RBDY3',candidat   ='SPHER',
                               colorCandidat   ='CYTOx',behav=bl_spher_cyto,
                               CorpsAntagoniste='RBDY3',antagoniste='SPHER',
                               colorAntagoniste='CYTOx',alert=alert_spher_cyto)
svs+=st_spher_cyto
st_spher_cyto2 = pre.see_table(CorpsCandidat   ='RBDY3',candidat   ='SPHER',
                               colorCandidat   ='CYTO2',behav=bl_spher_cyto2,
                               CorpsAntagoniste='RBDY3',antagoniste='SPHER',
                               colorAntagoniste='CYTO2',alert=alert_spher_cyto2)
svs+=st_spher_cyto2
st_spher_cyto_paroi  = pre.see_table(CorpsCandidat   ='RBDY3',candidat   ='SPHER',
                                     colorCandidat   ='PAROI',behav=bl_spher_cyto_paroi,
                                     CorpsAntagoniste='RBDY3',antagoniste='SPHER',
                                     colorAntagoniste='CYTOx',alert=alert_spher_cyto_paroi)
svs+=st_spher_cyto_paroi
st_spher_cyto_paroi2  = pre.see_table(CorpsCandidat   ='RBDY3',candidat   ='SPHER',
                                      colorCandidat   ='PAROI',behav=bl_spher_cyto_paroi2,
                                      CorpsAntagoniste='RBDY3',antagoniste='SPHER',
                                      colorAntagoniste='CYTO2',alert=alert_spher_cyto_paroi2)
svs+=st_spher_cyto_paroi2

#ecriture fichier liaisons
file = open('DATBOX/PTPT3_NETWORK.DAT','w')
dist_ini = rcell*(2-2*math.cos(angle))**0.5
for i in range(nb_paroi_ini-1):
    line = str(list_sph_paroi[i][1]) + ' '
                + str(list_sph_paroi[i+1][1]) + ' 0 0 0 '
                + str(dist_ini) + '\n'
    file.write(line)
line = str(list_sph_paroi[0][1]) + ' '
        + str(list_sph_paroi[nb_paroi_ini-1][1]) + ' 0 0 0 '
        + str(dist_ini) + '\n'
file.write(line)
```

```
file.close()
#ecriture fichiers datbox
post = pre.postpro_commands()
pre.writePostpro(commands=post, parts=bodies, path='DATBOX/')
pre.writeBodies(bodies,chemin='DATBOX/')
pre.writeBulkBehav(mat,chemin='DATBOX/', gravy=[0., 0., 0.])
pre.writeTactBehav(tacts,svs,chemin='DATBOX/')
pre.writeDrvDof(bodies,chemin='DATBOX/')
pre.writeDofIni(bodies,chemin='DATBOX/')
pre.writeVlocRlocIni(chemin='DATBOX/')
#visualisation
pre.visuAvatars(bodies)
#transfert donnees
import pickle
pre_data = ( dim, rcell , rparoi, list_sph_paroi, dist_ini, list_sph_cyto)
with open('pre_data.pickle','wb') as f:
  pickle.dump( pre_data, f, pickle.HIGHEST_PROTOCOL)

#FIN FICHIER
```

Annexe B Fichier de calcul

```
#DEBUT FICHIER
def takeFirst(elem):
    return elem[0]

# getting data from pre
from pylmgc90 import chipy
import re
import numpy as np
import shutil
import random
import pickle
with open('pre_data.pickle','rb') as f:
  pre_data = pickle.load( f )
dim         = pre_data[0]
rcell       = pre_data[1]
rparoi      = pre_data[2]
list_sph_paroi = pre_data[3]
dist_ini        = pre_data[4]
list_sph_cyto = pre_data[5]

#variables de controle
iter_ajout = 50
dist_max = 4*rparoi
r_add_cyto = 0.2*rcell
dist_norm = 0.5
proba_div = 0.2
paroi_max = 50

# modeling hypothesis ( 1 = plain strain, 2 = plain stress, 3 = axi-symmetry)
mhyp = 0
# time evolution parameters
tmax = 100
dt = 1.e-1
nb_steps = int(tmax/dt)
# theta integrator parameter
theta = 0.5
# deformable  yes=1, no=0
deformable = 0
# interaction parameters
freq_detect = 1
Rloc_tol = 5.e-2
# nlgs parameters
tol = 1e-4
relax = 1.0
norm = 'Quad'
gs_it1 = 200
gs_it2 = 10
solver_type='Stored_Delassus_Loops'
# write parameter
freq_write = 1
n_write_final = nb_steps//freq_write
restart_write = nb_steps//freq_write
# display parameters
```

```
freq_display = 1
restart_display = nb_steps//freq_display
ref_radius = 1.e0
# restart count:
ii = 0
chipy.Initialize()
chipy.checkDirectories()
chipy.nlgs_3D_DiagonalResolution()
# Set space dimension
chipy.SetDimension(dim,mhyp)
#
chipy.utilities_logMes('INIT TIME STEPPING')
chipy.TimeEvolution_SetTimeStep(dt)
chipy.Integrator_InitTheta(theta)
#
chipy.utilities_logMes('READ BEHAVIOURS')
chipy.ReadBehaviours()
if deformable: chipy.ReadModels()
#
chipy.utilities_logMes('READ BODIES')
chipy.ReadBodies()
#
chipy.utilities_logMes('LOAD BEHAVIOURS')
chipy.LoadBehaviours()
if deformable: chipy.LoadModels()
#
chipy.utilities_logMes('LOAD TACTORS')
chipy.LoadTactors()
#
chipy.utilities_logMes('READ DRIVEN DOF')
chipy.ReadDrivenDof()
#
chipy.utilities_logMes('READ INI')
if ii==0:
    chipy.ReadIniDof()
    chipy.ReadIniVlocRloc()
    if deformable:
        chipy.ReadIniGPV()
else:
    chipy.ReadIniDof(ii*restart_write)
    chipy.ReadIniVlocRloc(ii*restart_write)
    if deformable:
        chipy.ReadIniGPV(ii*restart_write)

#rend invisible les spheres
for i in list_sph_paroi:
    if i[2]==-1:
        chipy.RBDY3_SetInvisible(i[0])
for i in list_sph_cyto :
    if i[1]==-1:
        chipy.RBDY3_SetInvisible(i[0])

chipy.utilities_logMes('WRITE BODIES')
chipy.WriteBodies()
chipy.utilities_logMes('WRITE BEHAVIOURS')
```

```
chipy.WriteBehaviours()
chipy.utilities_logMes('WRITE DRIVEN DOF')
chipy.WriteDrivenDof()
chipy.utilities_logMes('DISPLAY & WRITE')
if ii==0:
    chipy.OpenDisplayFiles()
else:
    chipy.OpenDisplayFiles(ii*restart_display+1)
chipy.OpenPostproFiles()
chipy.utilities_logMes('COMPUTE MASS')
chipy.ComputeMass()
#lecture fichier de relations
chipy.PTPT3_LoadNetwork()

#initialisation list_cell
list_cell = []
list_paroi = []
for i in list_sph_paroi :
    if i[2]==1:
        list_paroi.append(i[0])
cdg = [0.,0.,0.,0.,0.,0.]
list_cell.append([1, list(list_paroi), 0, list(cdg), [-1.,-1.,-1.]])
nb_cell=1
#initialisation listes temporaires
list_line = []
list_ajout = []

#debut boucle calcul
for k in range(0,nb_steps):

    #ajout sphere cytoplasme
    if (k % iter_ajout) == 0:
        for j in list_cell :
            if j[2]==0 :
                id_cyto_ajout = list_sph_cyto.index(next(x for x in list_sph_cyto if x[1]==-1))
                pos_ajout = j[3]
                chipy.RBDY3_PutBodyVector("Coor0", list_sph_cyto[id_cyto_ajout][0], pos_ajout)
                list_sph_cyto[id_cyto_ajout][1]=1
                chipy.RBDY3_SetVisible(list_sph_cyto[id_cyto_ajout][0])

    chipy.utilities_logMes('INCREMENT STEP')
    chipy.IncrementStep()
    chipy.utilities_logMes('COMPUTE Fext')
    chipy.ComputeFext()
    chipy.utilities_logMes('COMPUTE Fint')
    chipy.ComputeBulk()

    #DIVISION
    for j in list_cell :
        if j[2]==1:
            id_div_1 = j[4][1]-1
            id_div_2 = j[4][2]-1
            pos1 = chipy.RBDY3_GetBodyVector("Coorb", list_sph_paroi[id_div_1][0])
            pos2 = chipy.RBDY3_GetBodyVector("Coorb", list_sph_paroi[id_div_2][0])
            dist_division = ((pos1[0]-pos2[0])**2+(pos1[1]-pos2[1])**2)**0.5
```

```
164             vectD = (pos2-pos1)/dist_division #vecteur directeur unitaire
165
166             if dist_division > dist_ini : #imposition force division
167                 chipy.RBDY3_PutBodyVector("Fext_", list_sph_paroi[id_div_1][0],
168                                           1.e3*vectD*dist_division*j[4][0])
169                 chipy.RBDY3_PutBodyVector("Fext_", list_sph_paroi[id_div_2][0],
170                                           -1.e3*vectD*dist_division*j[4][0])
171                 j[4][0] = j[4][0]+1
172
173             else : #DEBUT SEPARATION
174                 vectN = [-vectD[1], vectD[0], 0, 0, 0, 0] #vecteur normal unitaire
175                 list_ajout[:] = []
176                 list_paroi_ini = list(j[1])
177
178                 for i in [id_div_1, id_div_2]:
179                     pos = chipy.RBDY3_GetBodyVector("Coorb", list_sph_paroi[i][0])
180                     file = open('DATBOX/PTPT3_NETWORK.DAT','r')
181                     list_line[:] = []
182                     #cherche lignes de relations ou sph apparait
183                     for line in file:
184                         linefile = line.split(' ')
185                         if linefile[0] == list_sph_paroi[i][1]
186                                 or linefile[1] == list_sph_paroi[i][1]:
187                             list_line.append(line)
188                     file.close()
189
190                     #pour chaque relation trouvee
191                     for linefile in list_line:
192                         line = linefile.split(' ')
193                         #recuperation sphere invisible
194                         id_paroi_ajout = list_sph_paroi.index(
195                                 next(x for x in list_sph_paroi if x[2]==-1))
196                         #recuperation numero sphere a cote et range par ordre croissant
197                         if line[0] == list_sph_paroi[i][1]:
198                             numCote = int(line[1])-1
199                         else:
200                             numCote = int(line[0])-1
201
202                         #positionnnement nouvelle sphere
203                         pos2 = chipy.RBDY3_GetBodyVector("Coorb", list_sph_paroi[numCote][0])
204                         vect12 = pos2 - pos #vecteur sph_centr->sph_cote
205                         c = (vect12[0]*vectN[0]+vect12[1]*vectN[1])
206                         vect1N = [c*vectN[0], c*vectN[1], 0, 0, 0, 0] #projection sur la normale
207                         posN = [pos[0]+vect1N[0], pos[1]+vect1N[1], 0, 0, 0, 0]
208                         chipy.RBDY3_PutBodyVector("Coor0", list_sph_paroi[id_paroi_ajout][0], posN)
209                         #rendre visible
210                         list_sph_paroi[id_paroi_ajout][2] = 1
211                         chipy.RBDY3_SetVisible(list_sph_paroi[id_paroi_ajout][0])
212
213                         #stockage sph ajoutee + cote de la normale
214                         list_ajout.append([id_paroi_ajout, c])
215
216                         #reecriture du fichier
217                         if id_paroi_ajout > numCote :
218                             new_line = str(list_sph_paroi[numCote][1]) + ' '
```

```
                                    + str(list_sph_paroi[id_paroi_ajout][1]) +' 0 0 0 '
                                    + str(dist_ini) + '\n'
                        else :
                            new_line = str(list_sph_paroi[id_paroi_ajout][1]) + ' '
                                    + str(list_sph_paroi[numCote][1]) +' 0 0 0 '
                                    + str(dist_ini) + '\n'
                        file = open('DATBOX/PTPT3_NETWORK.DAT','a')
                        file.write(new_line)
                        file.close()

                        #ajout dans la liste des spheres de paroi de la cellule
                        list_paroi_temp = list(j[1])
                        index_i = list_paroi_temp.index(i+1)
                        index_cote = list_paroi_temp.index(numCote+1)
                        if index_cote < index_i :
                            list_paroi_temp.insert(index_i,list_sph_paroi[id_paroi_ajout][0])
                        else:
                            list_paroi_temp.insert(index_cote,list_sph_paroi[id_paroi_ajout][0])
                        j[1]=list(list_paroi_temp)


                #rearrangement liste sph ajoutee dans lordre [[1,X],[2,Y]...]
                list_ajout = sorted(list_ajout, key=takeFirst)
                id1 = list_ajout[0][0]
                if np.sign(list_ajout[0][1]) == np.sign(list_ajout[1][1]):
                    id2 = list_ajout[1][0]
                    id3 = list_ajout[2][0]
                    id4 = list_ajout[3][0]
                elif np.sign(list_ajout[0][1]) == np.sign(list_ajout[2][1]):
                    id2 = list_ajout[2][0]
                    id3 = list_ajout[1][0]
                    id4 = list_ajout[3][0]
                else :
                    id2 = list_ajout[3][0]
                    id3 = list_ajout[1][0]
                    id4 = list_ajout[2][0]
                #ecriture relations sphs de chaque cote
                new_line1 = str(list_sph_paroi[id1][1]) + ' '
                            + str(list_sph_paroi[id2][1]) +' 0 0 0 '
                            + str(dist_ini) + '\n'
                new_line2 = str(list_sph_paroi[id3][1]) + ' '
                            + str(list_sph_paroi[id4][1]) +' 0 0 0 '
                            + str(dist_ini) + '\n'
                file = open('DATBOX/PTPT3_NETWORK.DAT','a')
                file.write(new_line1)
                file.write(new_line2)
                file.close()

                #reecriture fichier relations
                file_old = open('DATBOX/PTPT3_NETWORK.DAT','r')
                file_new = open('DATBOX/PTPT3_NETWORK_NEW.DAT','w')
                for line in file_old :
                    if line not in list_line : #on ne copie que les lignes a garder
                        file_new.write(line)
                file_old.close()
```

```
                file_new.close()
                shutil.copy('DATBOX/PTPT3_NETWORK_NEW.DAT', 'DATBOX/PTPT3_NETWORK.DAT')
                #reset fichier relations
                chipy.utilities_logMes('RESET PTPT3 DETECTION')
                chipy.PTPT3_SelectProxTactors(1)
                #rend invisible les anciennes spheres
                chipy.RBDY3_SetInvisible(list_sph_paroi[id_div_1][0])
                list_sph_paroi[id_div_1][2] = -1
                chipy.RBDY3_SetInvisible(list_sph_paroi[id_div_2][0])
                list_sph_paroi[id_div_2][2] = -1

                #creation des nvls listes sphs
                list_paroi_temp = list(j[1])
                list_paroi_temp.remove(j[4][1])
                list_paroi_temp.remove(j[4][2])
                index1 = 1 + next(list_paroi_temp.index(x)
                                  for x in list_paroi_temp if x not in list_paroi_ini)
                index2 = 1 + next(list_paroi_temp.index(x)
                                  for x in list_paroi_temp if (x not in list_paroi_ini)
                                                           and list_paroi_temp.index(x)>index1 )
                list_paroi_temp1 = list_paroi_temp[:index1]
                list_paroi_temp2 = list_paroi_temp[index1:index2]
                list_paroi_temp3 = list_paroi_temp[index2:]
                nvl_list1 = list_paroi_temp1 + list_paroi_temp3
                nvl_list2 = list_paroi_temp2
                #mise a jour de list_cell
                nb_cell = nb_cell+1
                list_cell.append([nb_cell,list(nvl_list1),0,[0.,0.,0.,0.,0.,0.],[-1.,-1.,-1.]])
                j[1] = list(nvl_list2)
                j[2] = 0
                j[4] = list([-1.,-1.,-1.])
                #fin division
        #FIN DIVISION

        chipy.utilities_logMes('COMPUTE Free Vlocy')
        chipy.ComputeFreeVelocity()
        chipy.utilities_logMes('SELECT PROX TACTORS')
        chipy.SelectProxTactors(freq_detect)
        chipy.utilities_logMes('RESOLUTION' )
        chipy.RecupRloc(Rloc_tol)
        chipy.ExSolver(solver_type, norm, tol, relax, gs_it1, gs_it2)
        chipy.UpdateTactBehav()
        chipy.StockRloc()
        chipy.utilities_logMes('COMPUTE DOF, FIELDS, etc.')
        chipy.ComputeDof()
        chipy.utilities_logMes('UPDATE DOF, FIELDS')
        chipy.UpdateStep()
        chipy.utilities_logMes('WRITE OUT DOF')
        chipy.WriteOutDof(freq_write)
        chipy.utilities_logMes('WRITE OUT Rloc')
        chipy.WriteOutVlocRloc(freq_write)
        chipy.utilities_logMes('VISU & POSTPRO')
        chipy.WriteDisplayFiles(freq_display,ref_radius)
        chipy.WritePostproFiles()

```

```
    #determination si lancement de la division ou non
    for j in range(len(list_cell)):
        if len(list_cell[j][1])>=paroi_max and list_cell[j][2]==0:
            chance_div = np.random.rand()
            if chance_div > proba_div: #si division acceptee
                list_cell[j][2]=1
                id_rand = int(((len(list_cell[j][1])//2)-1)*np.random.rand())+1
                id_div_1 = list_cell[j][1][id_rand]
                id_div_2 = list_cell[j][1][id_rand+((len(list_cell[j][1])//2)-1)]
                list_cell[j][4][0] = 0
                list_cell[j][4][1] = id_div_1
                list_cell[j][4][2] = id_div_2


    #CROISSANCE MEMBRANE
    for j in range(len(list_cell)):
        if list_cell[j][2] == 0 : #si cellule est en croissance
            list_ajout[:] = []
            list_paroi_temp = list(list_cell[j][1])
            for i in range(len(list_paroi_temp)-1): #test de distance sur tous les couples
                pos1 = chipy.RBDY3_GetBodyVector("Coor_",
                                list_sph_paroi[list_paroi_temp[i]-1][0])
                pos2 = chipy.RBDY3_GetBodyVector("Coor_",
                                list_sph_paroi[list_paroi_temp[i+1]-1][0])
                dist_att = ((pos1[0]-pos2[0])**2+(pos1[1]-pos2[1])**2)**0.5
                if dist_att > dist_max : #si la distance est trop grande
                    list_ajout.append([i,pos1,pos2])
            pos1 = chipy.RBDY3_GetBodyVector("Coor_",
                        list_sph_paroi[list_paroi_temp[len(list_paroi_temp)-1]-1][0])
            pos2 = chipy.RBDY3_GetBodyVector("Coor_",
                        list_sph_paroi[list_paroi_temp[0]-1][0])
            dist_att = ((pos1[0]-pos2[0])**2+(pos1[1]-pos2[1])**2)**0.5
            if dist_att > dist_max :
                list_ajout.append([len(list_paroi_temp)-1,pos1,pos2])

            for i in range(len(list_ajout)): #pour chaque couple trouve
                num_sph1 = list_paroi_temp[list_ajout[i][0]] #recuperation num sphs
                if list_ajout[i][0] == len(list_paroi_temp)-1 :
                    num_sph2 = list_paroi_temp[0]
                else:
                    num_sph2 = list_paroi_temp[list_ajout[i][0]+1]

                #creation et positionnement nvl sphere
                id_paroi_ajout = list_sph_paroi.index(
                                next(x for x in list_sph_paroi if x[2]==-1) )
                pos1 = list_ajout[i][1]
                pos2 = list_ajout[i][2]
                vectD = pos2-pos1
                ptMilieu = (pos1+pos2)/2
                vectN = [-vectD[1]*dist_norm, vectD[0]*dist_norm, 0, 0, 0, 0] #vecteur normal
                pos_norm1 = [ptMilieu[0]+vectN[0], ptMilieu[1]+vectN[1],0,0,0,0,]
                pos_norm2 = [ptMilieu[0]-vectN[0], ptMilieu[1]-vectN[1],0,0,0,0,]
                distCDM1 = ((pos_norm1[0]-cdg[0])**2+(pos_norm1[1]-cdg[1])**2)**0.5
                distCDM2 = ((pos_norm2[0]-cdg[0])**2+(pos_norm2[1]-cdg[1])**2)**0.5
                if distCDM1 > distCDM2: #choix normale sortante
```

```
                    pos_norm = pos_norm1
                else:
                    pos_norm = pos_norm2
                chipy.RBDY3_PutBodyVector("Coor0", list_sph_paroi[id_paroi_ajout][0], pos_norm)
                #rend visible
                list_sph_paroi[id_paroi_ajout][2]=1
                chipy.RBDY3_SetVisible(list_sph_paroi[id_paroi_ajout][0])

                #modification liste paroi
                list_paroi_temp.insert(list_ajout[i][0]+1,list_sph_paroi[id_paroi_ajout][0])
                list_cell[0][1]=list(list_paroi_temp)

                #cherche ligne de relations ou les 2 sph apparaissent
                file = open('DATBOX/PTPT3_NETWORK.DAT','r')
                for line in file:
                    linefile = line.split(' ')
                    if ( linefile[0] == list_sph_paroi[num_sph1-1][1]
                                or linefile[1] == list_sph_paroi[num_sph1-1][1] )
                        and ( linefile[0] == list_sph_paroi[num_sph2-1][1]
                                or linefile[1] == list_sph_paroi[num_sph2-1][1] ):
                        line_rel = line
                        break
                file.close()

                #tri lignes ordre croissant
                line = line_rel.split(' ')
                if id_paroi_ajout > int(line[0])-1:
                    new_line1 = str(list_sph_paroi[int(line[0])-1][1]) + ' '
                                + str(list_sph_paroi[id_paroi_ajout][1]) +' 0 0 0 '
                                + str(dist_ini) + '\n'
                    if id_paroi_ajout > int(line[1])-1:
                        new_line2 = str(list_sph_paroi[int(line[1])-1][1]) + ' '
                                    + str(list_sph_paroi[id_paroi_ajout][1]) +' 0 0 0 '
                                    + str(dist_ini) + '\n'
                    else :
                        new_line2 = str(list_sph_paroi[id_paroi_ajout][1]) + ' '
                                    + str(list_sph_paroi[int(line[1])-1][1]) +' 0 0 0 '
                                    + str(dist_ini) + '\n'
                else :
                    new_line1 = str(list_sph_paroi[id_paroi_ajout][1]) + ' '
                                + str(list_sph_paroi[int(line[0])-1][1]) +' 0 0 0 '
                                + str(dist_ini) + '\n'
                    new_line2 = str(list_sph_paroi[id_paroi_ajout][1]) + ' '
                                + str(list_sph_paroi[int(line[1])-1][1]) +' 0 0 0 '
                                + str(dist_ini) + '\n'

                #ecriture lignes
                file = open('DATBOX/PTPT3_NETWORK.DAT','a')
                file.write(new_line1)
                file.write(new_line2)
                file.close()

                #reecriture fichier relations
                file_old = open('DATBOX/PTPT3_NETWORK.DAT','r')
                file_new = open('DATBOX/PTPT3_NETWORK_NEW.DAT','w')
```

```
439                 for line in file_old :
440                     if line != line_rel : #on ne copie que les lignes a garder
441                         file_new.write(line)
442                 file_old.close()
443                 file_new.close()
444                 shutil.copy('DATBOX/PTPT3_NETWORK_NEW.DAT', 'DATBOX/PTPT3_NETWORK.DAT')
445                 #reset fichier relations
446                 chipy.utilities_logMes('RESET PTPT3 DETECTION')
447                 chipy.PTPT3_SelectProxTactors(1)
448             #fin pour chaque couple
449     #FIN CROISSANCE MEMBRANE
450 
451     #calcul CDG cellules
452     for j in range(len(list_cell)) :
453         cdg = [0,0,0,0,0,0]
454         list_paroi = list(list_cell[j][1])
455         for i in list_paroi:
456             cdg+=chipy.RBDY3_GetBodyVector("Coor_", i)
457         cdg = cdg/len(list_paroi)
458         list_cell[j][3] = list(cdg)
459 
460 #FIN BOUCLE CALCUL
461 
462 chipy.CloseDisplayFiles()
463 chipy.ClosePostproFiles()
464 chipy.Finalize()
465 
466 #FIN FICHIER
```