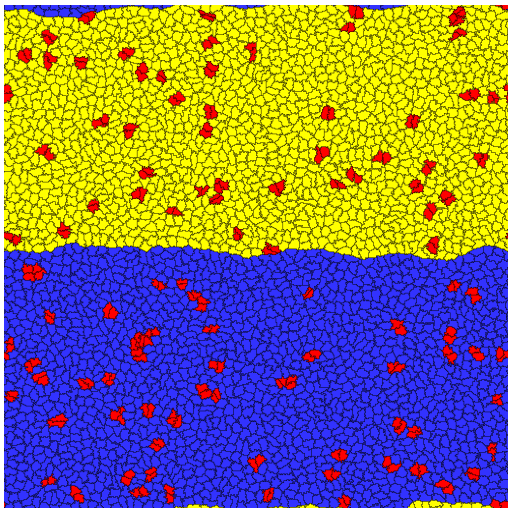
# Modélisation dynamique de l'interface entre deux tissus cellulaires

Rapport de Stage

**Djamal Chekroun**
**Maître de Stage Marc Durand**



Laboratoire Matière et Systèmes Complexes
Avril-Juin 2022

Université Paris Cité
Master-2 Systèmes Biologiques et Concepts Physiques

# Contents

# 1 Introduction

Le stage de M2 s'est déroulé au sein du laboratoire Matière et Systèmes Complexes de l'Université Paris Cité sous la supervision de Marc Durand. Le stage a duré du 3 avril au 1er Juillet.
L'objet du stage est de:
- mettre au point un programme informatique modélisant l'interface entre deux tissus cellulaires actifs, soumis à l'agitation thermique et aux réarrangemements entre cellules,
- simuler la division et l'apoptose des cellules,
- etudier la forme des interfaces entre deux populations de cellules et le spectre spatial des interfaces,
- comparer la forme des spectres au modèle théorique,
- étudier la corrélation temporelle des spectres obtenus sur une longue période pour estimer le caractère stationnaire des configurations obtenues.

Le modèle se limite à une projection en 2D. Les cellules sont représentées sur une grille rectangulaire de 512 x 512 pixels (automate cellulaire). Les cellules baignent dans un milieu intra-cellulaire.
Leur évolution dynamique est simulée au travers d'un modèle Cellular Potts. Le principe de ce modèle est de faire évoluer sur la grille les populations de cellules de façon aléatoire, de façon à faire baisser l'energie du système global, en le soumettant néanmoins à l'agitation thermique. Il faut veiller au maintien de la simple connexité des cellules.

Le programme a été élaboré en partant d'un logiciel existant réalisé par Marc Durand. Ce logiciel incorpore déjà le modèle cellulaire de Potts et s'assure que la connexité des cellules est maintenue.

L'interface entre tissus cellulaires a déjà été étudiée et le labo MSC a fait plusieurs publications sur ce thème (voir bibliographie) mais le spectre des interfaces n'a pas été étudié spécifiquement à notre connaissance.

Le stage s'est déroulé en plusieurs étapes :
- se familiariser avec le logiciel Potts existant,
- modéliser la simple agitation thermique,
- modéliser division et apoptose,
- exécuter le programme sur le cluster du labo, muni de 96 processeurs,
- etudier les spectres obtenus et les corrélations temporelles pour chaque type d'agitation (thermique vs division+apoptose).

Les simulations ont montré que l'activité des tissus cellulaires augmentait l'intensité de tous les modes, notamment ceux de faible fréquence spatiale (de période supérieure à la longueur typique d'une cellule). La correspondance avec le modèle théorique est bonne si on se limite à ces premiers modes. La corrélation temporelle est très longue en comparaison de la fréquence de division des cellules.

Les fichiers source sont rassemblés dans le repository GitHub `https://github.com/djachk/CPM_stage.git` .

# 2 Aspects théoriques

Dans le cas de populations de cellules biologiques actives en 2D, on peut montrer que l'Hamiltonien s'ecrit

$$H = \sum_{cells<i,j>} \gamma_{ij}\mathcal{L}_{ij} + \frac{B}{2A_0}\sum_{cells\ i}(A_i - A_i^0)^2 \tag{1}$$

Le premier terme correspond à l'energie de contact entre cellules voisines. $\mathcal{L}_{ij}$ est la longueur du contact entre deux cellules i et j.

Le deuxième terme correspond à l'energie de compression de chacune des cellules.

Dans le cas d'une représentation sous forme d'automate cellulaire, l'Hamiltonien s'ecrit

$$H = \sum_{sites<k,l>} J(\sigma_k, \sigma_l)(1 - \delta_{\sigma_k,\sigma_l}) + \lambda \sum_{cells\ i} \frac{(A_i - A_i^0)^2}{A_i^0} \tag{2}$$

$\sigma_k$ est le type de cellule.

La correspondance avec l'Hamiltonien physique ci-dessus est: $B = 2\lambda$, $\gamma_{ij} = zJ(i,j)$ avec z "facteur de forme" indiquant la portée de l'interaction, égal dans notre cas à 11.3.

Les $J_{ij}$ vont commander l'évolution des interfaces. Plus $J_{12}$ sera grand devant $J_{11}$ et $J_{22}$, plus la longueur de l'interface entre les deux types de cellules sera minimisée. L'agitation thermique représente l'activité biologique de la membrane.

On rapproche cette configuration du cas générique d'une membrane de hauteur h(x,y) sur une surface projetée $A_p$, où l'Hamiltonien s'écrit:

$$H = \frac{\kappa_b}{2} \int\int_{A_p} (\nabla^2 h)^2 dxdy + \frac{\sigma}{2} \int\int_{A_p} (\overrightarrow{\nabla} h)^2 dxdy + \sigma A_p \tag{3}$$

$\kappa_b$ est la rigidité de courbure (bending rigidity), $\sigma$ est la tension intrinsèque de la membrane.

Le spectre s'obtient par Transformée de Fourier:

$$\tilde{h}(\overrightarrow{k}) = \frac{1}{L^2} \int\int_A h(\overrightarrow{r}) e^{-i\overrightarrow{k}.\overrightarrow{r}} dxdy \tag{4}$$

On peut montrer que l'amplitude des différents modes se calcule ainsi:

$$< |\tilde{h}(\overrightarrow{k})|^2 >= \frac{k_b T}{\sigma k^2 + \kappa_b k^4} \tag{5}$$

On cherchera à appprocher par une telle formule le spectre des interfaces entre deux populations de cellules dans notre modèle.

## 3 Le modèle cellulaire de Potts

Le modèle Potts Cellulaire (aussi appelé modèle Glazier-Graner-Hogeweg) est une généralisation du modèle d'Ising et représente un système cellulaire sur une grille de pixels, chaque pixel étant labellisé par le numéro de la cellule à laquelle il appartient.

A une cellule sont associées plusieurs caractéristiques, comme son type et sa surface cible $A_0$.

L'Hamiltonien s'ecrit donc selon l'equation (5).

L'ensemble du système est à une « température » T.

L'evolution du systeme se fait comme suit.
A chaque pas de temps est exécuté un algorithme de Monte Carlo 512 x 512 fois(MCS, Monte Carlo Sweep) :
- on choisit aléatoirement un pixel,
- on étudie le scénario où ce pixel prend comme nouveau label le label d'un pixel voisin différent, diminuant ainsi la surface de la cellule à laquelle il appartient et augmentant la surface de la cellule

à laquelle appartient le pixel candidat,
- on calcule la modification d'energie correspondante (en étendant l'interaction aux 20 cellules voisines afin d'atténuer l'anisotropie dûe à la géométrie de la grille),
- on accepte cette transformation si l'energie diminue ou bien (algorithme de Metropolis) aléatoirement avec la probabilité exp(-dE/T).

# 4 L'architecture logicielle

Le programme réalisant la modélisation des populations de cellules est écrit en C et exploite un logiciel existant, qui utilise la librairie Cash pour les représentations graphiques. Ce programme produit en particulier un fichier contenant les interfaces entre les deux populations de cellules, échantillonnées à intervalles réguliers.

Un script a été par ailleurs développé en Python pour lire les fichiers d'interfaces et en faire l'analyse.

Les deux structures de données les plus importantes dans le programme de modèlisation sont :
- la grille de 512 x 512 pixels représentant l'évolution spatiale des deux types de cellules,
- la structure correspondant à l'ensemble des cellules et leurs caractéristiques.

Le déroulement du programme de modélisation est le suivant:
- initialisation: on génére au départ un seul type de cellules,
- durée de dispersion: on laisse les cellules grandir pendant quelques pas de temps. A la fin de cette durée on transforme certaines cellules en cellule de type 2 pour générer les deux populations,
- boucle principale: on éxécute à chaque pas de temps l'algorithme de Monte Carlo, tout en s'assurant du maintien de la connexité des cellules,
- on calcule régulièrement, selon les critères figurant dans le fichier de paramètres, l'interface entre les deux types de cellules, et on écrit cette interface dans le fichier de sortie.

Un fichier de paramètres ainsi qu'un directory pour conserver les résulats sont données comme arguments au programme.
Le script Python, quant à lui:
- lit le fichier contenant les interfaces,
- les représente graphiquement,
- calcule leur spectre,
- calcule le modèle théorique le plus proche,
- calcule les corrélations temporelles entre modes en faisant une moyenne d'ensemble sur plusieurs réalisations.

# 5 Simulations avec seule agitation thermique

## 5.1 Principe de la simulation

On commence par faire une modélisation avec deux populations de cellules soumises à la seule agitation thermique (pas de division cellulaire ni d'apoptose).
On place au départ des cellules d'un seul type de façon aléatoire sur la grille, avec conditions aux bords périodiques. Chaque cellule occupe au départ un petit nombre de pixels et le nombre de cellules est calculé de telle sorte que la totalité de la grille soit occupée lorsque les cellules atteignent leur taille cible. A la fin de la durée de dispersion on transforme les cellules de la moitiè inférieure de la grille en cellules de type 2.
Après agitation thermique, on obtient le type de grille donné sur la figure ci-dessous, à peu près

stable dans le temps, l'interface oscillant sous l'effet de la température.

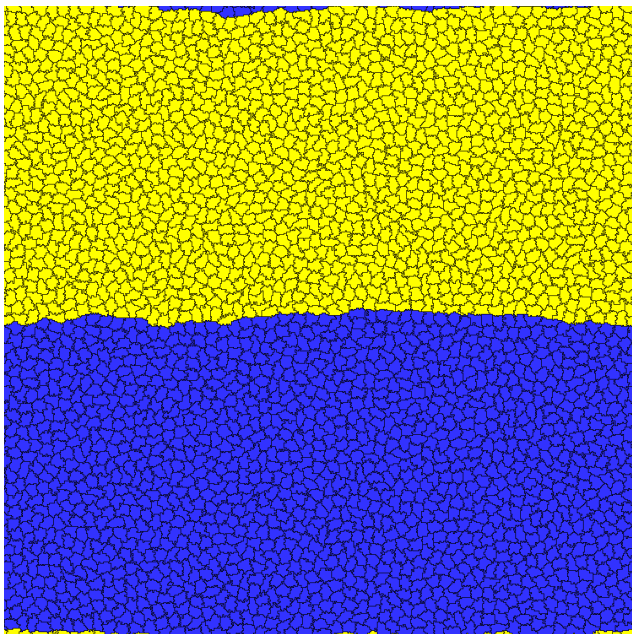


Figure 1: Deux populations de cellules soumises à la seule agitation thermique

L'ensemble des interfaces obtenues sur une longue durée (ici 200000 pas) est représenté par le script Python comme ci-après. On voit que les perturbations sont assez faibles.

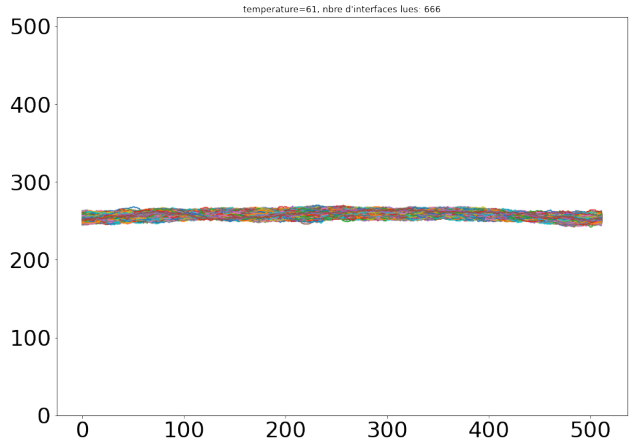



Figure 2: Toutes les interfaces pour un cas avec seule agitation thermique

La moyenne des modules des spectres de toutes les interfaces est donnée ci-après. On voit qu'au delà de q=50, l'intensité des modes est très faible (correspondant à la taille des cellules).

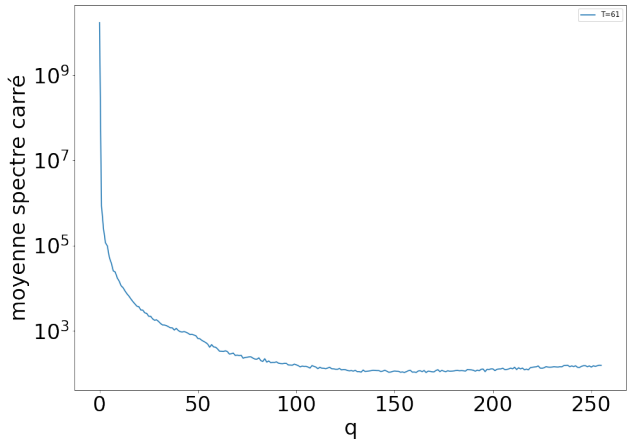



Figure 3: Spectre moyen dans un cas avec seule agitation thermique

Le modèle théorique indique:

$$< |\tilde{h}_q|^2 >= \frac{1}{bq^2 + cq^4} \tag{6}$$

La correspondance avec le modèle théorique est bonne si on se limite aux fréquences spatiales faibles (de longueur d'onde supérieure à la longueur des cellules). On représente ci-après $\frac{1}{<|\tilde{h}_q|^2>}$.

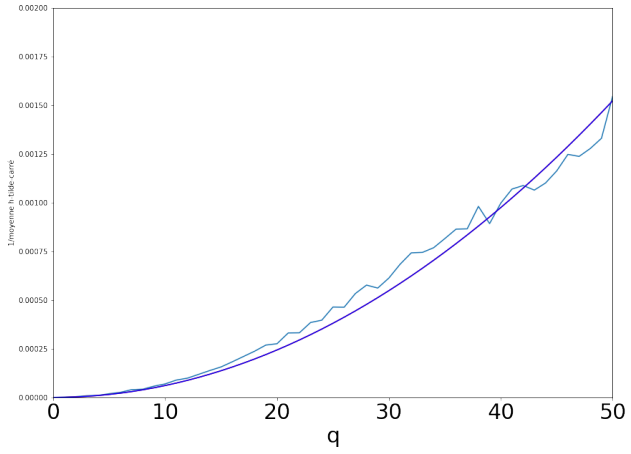



Figure 4: Correspondance de $\frac{1}{<|\tilde{h}_q|^2>}$ avec le modèle théorique

Pour une température de 60 et une interaction de contact $J_{12}$ de 25, on a b = 6.0 E-7 et c = 5.5 E-22. Le paramètre c est donc négligeable si bien que le spectre peut être approché par une formule du type:

$$< |\tilde{h}_q|^2 >= \frac{1}{aq^2} \tag{7}$$

Le module de flexion de l'interface entre 2 tissus semble donc négligeable.

### 5.2 Stationnarité de la configuration obtenue avec seule agitation thermique

On veut vérifier le caractère stationnaire des configurations obtenues.

Pour cela, on représente $| < \tilde{h}(q)\tilde{h}^*(q+\tau) >_t |$ en faisant varier tau pour chaque mode q.


Figure 5: Corrélation temporelle des spectres avec seule agitation thermique

Le temps de corrélation des spectres se situe vers une durée de 200 000 pas. La corrélation ensuite s'accroît en raison de la duré de simulation limitée.

Les modes de faibles q sont les plus longs à relaxer, comme attendu. Pour le mode q=1 en particulier, la durée de la simulation n'est pas très grand par rapport au temps de corrélation, ce qui explique pourquoi la courbe remonte.

On calcule la limte de $| < \tilde{h}(q)\tilde{h}^*(q+\tau) >_t |$ pour chaque mode quand $\tau$ tend vers l'infini. En faisant l'hypothèse de l'indépendance des $\tilde{h}$ pour $\tau$ grand, alors la limite est $| < \tilde{h}(q) >_t |^2$
Cette limite est inférieure à 10-7 (elle est en théorie égale à 0). Il faut une durée beaucoup plus longue pour s'approcher de cette limite.

# 6 Division cellulaire et apoptose

## 6.1 Modélisation de la division cellulaire

La division cellulaire est modélisée en coupant la zone occupée par la cellule sur la grille par une droite passant par son centre de gravité (recalculé à chaque pas) et d'orientation aléatoire. On vérifie la connexité de chaque cellule-fille ainsi obtenue. Dans le cas ou la connexité n'est pas maintenue, on change l'orientation d'un angle fixe et on vérifie à nouveau. Après 5 tentatives, la division est refusée et une nouvelle division sera tentée au pas de simulation suivant.

Sur la figure suivante, on voit une tentative de division qui échoue car la connexité de la cellule fille 1 n'est pas assurée.

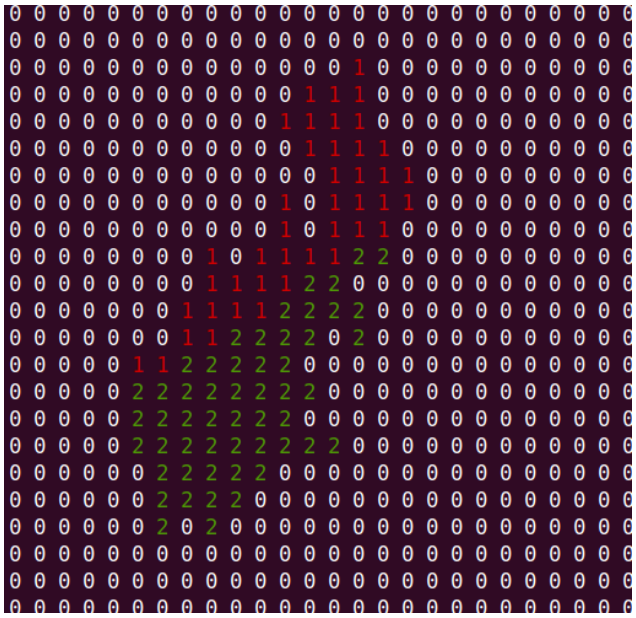


Figure 6: Division cellulaire et vérification de la connexité

A chaque pas de simulation, un pourcentage des cellules ayant atteint au moins 3/4 de leur taille cible sont divisées.

### 6.2 Modélisation de l'apoptose

L'apoptose est déclenchée lorsque les cellules sont trop compressées (il s'agit en fait d'une extrusion en 3 dimensions). Elle est déclenchée dans la modélisation lorsque la surface moyenne pour un type de cellules est plus faible d'un certain seuil que la surface cible. Et alors un pourcentage équivalent des cellules sont détruites.
La disparition d'une cellule ainsi désignée n'est pas immédiate. La cellule est en fait marquée et ne gagnera désormais plus de pixel lors des tours de Monte Carlo. La cellule ainsi "condamnée" perd assez rapidement tous ses pixels et disparaît (tout en laissant son label libre pour de nouvelles cellules, afin de ne pas saturer la mémoire).
Dans la simulation, les cellules divisées (les deux cellules filles) depuis le dernier affichage apparaissent en rouge à chaque affichage comme sur la figure ci-après.

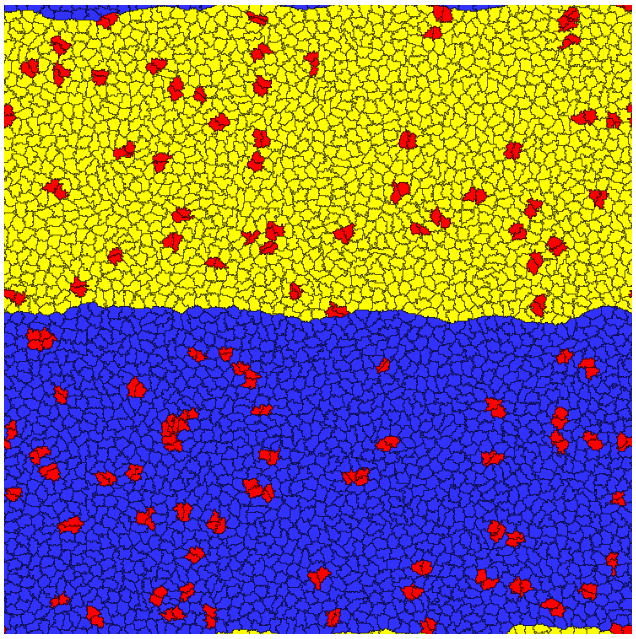


Figure 7: Exemple de grille en présence de division cellulaire et apoptose

### 6.3 Stationnarité de la configuration obtenue avec division/apoptose

On prend une température de 60 et une interaction J12 de 25, un rythme de division de 0.002 divisions par pas de temps et un seuil d'apoptose de 10%.

On représente ici aussi $| < \tilde{h}(q)\tilde{h}^*(q+\tau) >_t |$ en faisant varier tau pour chaque mode q.

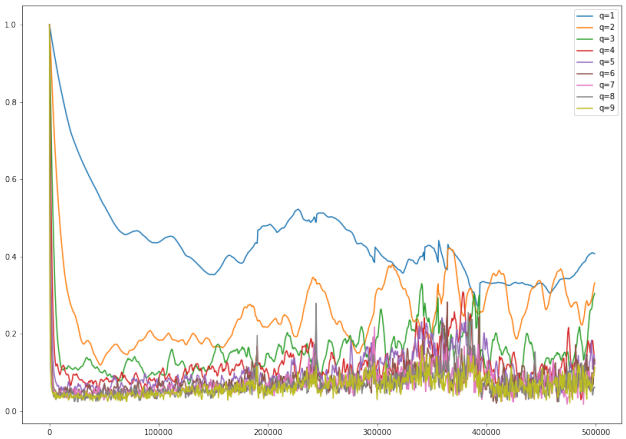



Figure 8: Corrélation temporelle des spectres avec division/apoptose

Le temps de corrélation des spectres se situe ici vers une durée de 150 000 pas. La corrélation ensuite s'accroît en raison de la duré de simulation limitée.

On calcule la limte de $| < \tilde{h}(q)\tilde{h}^*(q+\tau) >_t |$ pour chaque mode quand $\tau$ tend vers l'infini. En faisant l'hypothèse de l'indépendance des $\tilde{h}$ pour $\tau$ grand, alors la limite est $| < \tilde{h}(q) >_t |^2$.
Cette limite est la aussi inférieure à 10-7 (égale à 0 pour t infini). Il faut une durée beaucoup plus longue pour s'approcher de ces limites.

## 7 Simulations avec division et apoptose

La simulation est beaucoup plus dynamique en présence de division et d'apoptose. Le rythme d'apoptose s'adapte automatiquement et un régime stationnaire est rapidement obtenu en termes de nombre de cellules vivantes.
L'interface peut être très perturbée comme dans la figure ci-après.

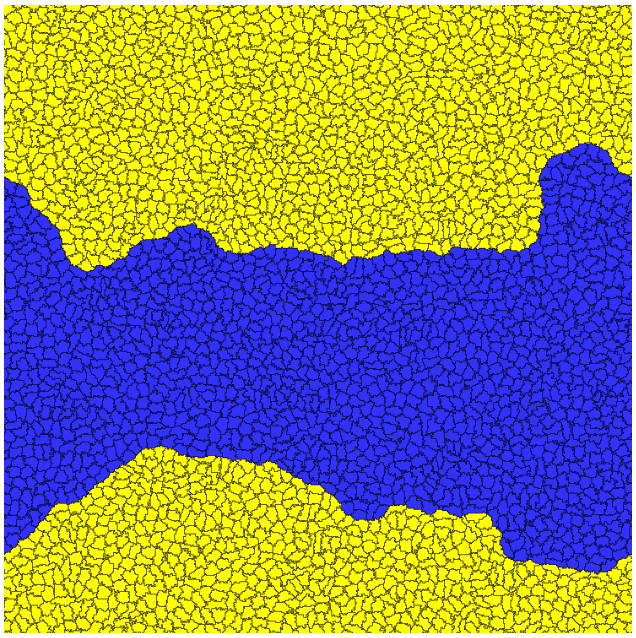


Figure 9: Exemple d'interface en présence de division et d'apoptose

Pour de faibles valeur de J12, l'interface peut se déformer de façon importante et atteindre une configuration de périmètre minimal, un cercle, le nombre de cellules étant stationnaire.

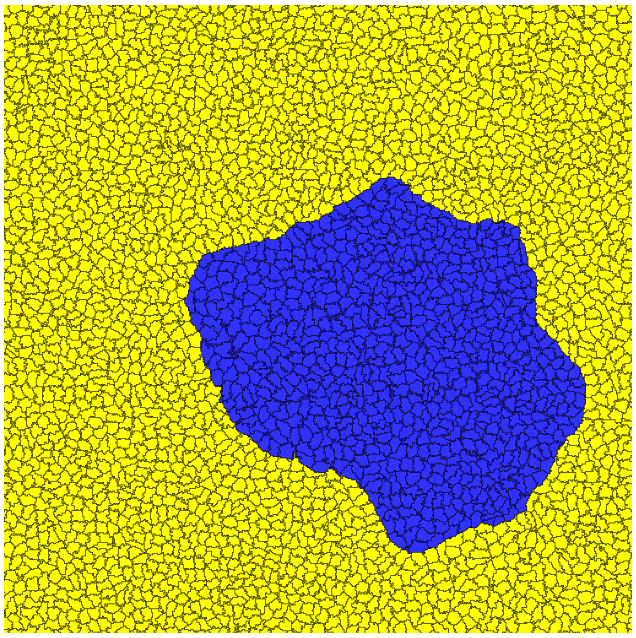


Figure 10: Interface atteignant une longueur minimale

L'ensemble de toutes les interfaces forment une figure plus mouvementée.

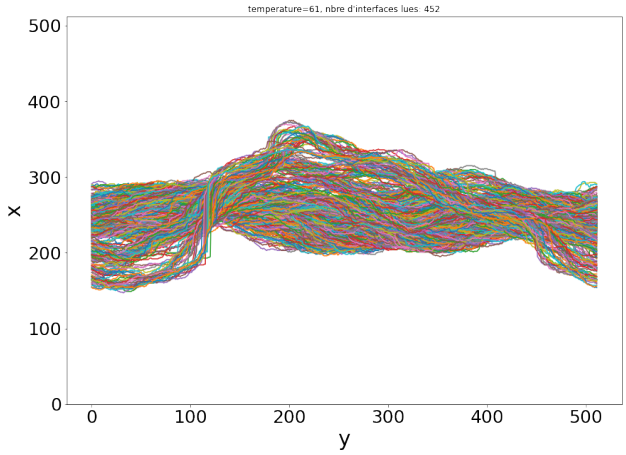



Figure 11: Ensemble des interfaces sur une longue durée en présence de divsion et apoptose

Cependant la durée de la simulation (ici 200000 pas) laisse encore voir une corrélation temporelle.

On prend encore une température T=60, une interaction J12=25, un rythme de division de 0.002 divisions par pas de temps et un seuil d'apoptose de 10%. On compare les spectres obtenus avec et sans division/apoptose.

La moyenne des spectres montre une intensité supérieure pour tous les modes à la moyenne des pectres associés à la seule agitation thermique.

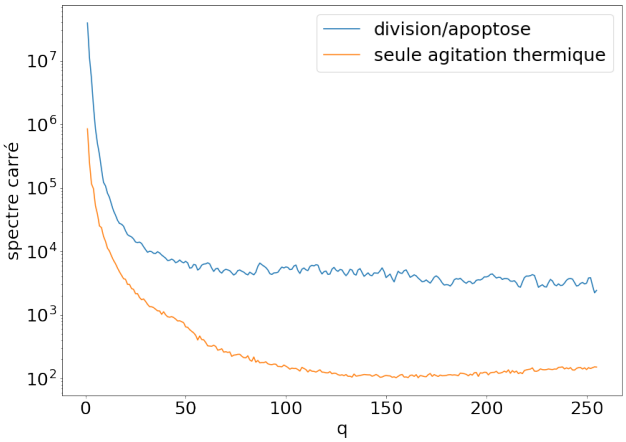



Figure 12: Moyennes des spectres avec agitation thermique et division/apoptose

La représentation de $\frac{1}{<|\tilde{h}_q|^2>}$ montre une bonne adéquation avec le modèle théorique avec un coefficient b plus faible.

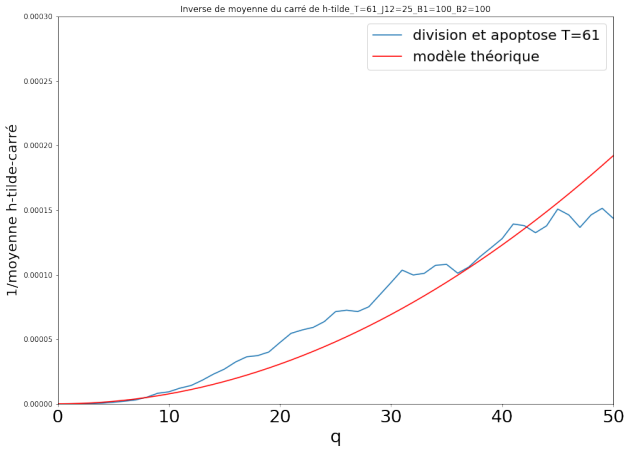



Figure 13: Ajustement avec modèle théorique aux basses fréquences

Mais la qualité de l'approximation est beaucoup moins bonne. On peut avancer deux causes à cela:
- on n'a pas atteint le régime stationnaire,
- on n'est pas dans le régime de petites fluctuations, ce que l'expression utilisée pour l'approximation pré-suppose.

Pour une température de 60 et une interaction de contact $J_{12}$ de 25, on a dans cette configuration b = 7.0 E-8 et c = 1.3 E-20.
c est donc négligeable si bien que le spectre peut être approché par une formule du type:

$$< |\tilde{h}_q|^2 >= \frac{1}{aq^2} \tag{8}$$

# 8 Forme des interfaces et recherche de température effective

Voici les valeurs typiques de tous les paramètres impôrtants des simulations:
- température: de 0.01 à 90
- Energies de contact entre cellules: J11=J22=14; J12=25; J01=J02=18
- surface cible des cellules: B=100
- coefficient de compression: A0=100
- duree de simulation: entre 200000 et 600000 pas
- taux de division cellulaire: 0.002 divsions par pas de temps
- seuil d'apoptose: 1%

On va étudier ici plusieurs scénarios avec et sans agitation thermique et division/apoptose pour évaluer la forme des interfaces.

**Forme des interfaces avec seule agitation thermique**

On calcule la largeur moyenne de l'interface $< h^2 >_t$ pour différentes configurations, avec $h^2 = \sum_{i=0}^{N} h_i^2$, $h_i$ étant la hauteur centrée de l'interface à la position i.

**Forme des interfaces avec seule agitation thermique**
(J12=25 durée=500000)

On calcule $< h^2 >_t$ pout T=200, 300, 400, 500.

On obtient la figure ci-après:

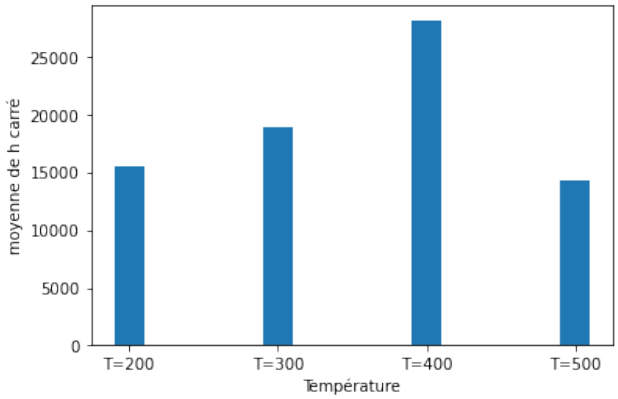



Figure 14: Largeur moyenne des interfaces pour les températures 200, 300, 400, 500

La baisse de la largeur moyenne des interfaces à T=500 laisse penser que le système n'a pas encore atteint un état stationnaire après cette durée de simulation.

**Forme des interfaces avec division cellulaire/apoptose**

On prend une température très faible, pour avoir une agitation thermique négligeable.

On étudie les scénarios suivants:

Scénario 0:
(T=0.07, taux de division:0.000002, seuil apoptose:0.0002 J12=25 durée=500000)

Scénario 1:
(T=0.08, taux de division:0.000005, seuil apoptose:0.0005 J12=25 durée=500000)

Scénario 2:
(T=0.09, taux de division:0.00001, seuil apoptose:0.001 J12=25 durée=500000)

Scénario 3:
(T=0.11, taux de division:0.00002, seuil apoptose:0.002 J12=25 durée=500000)

Scénario 4:
(T=0.12, taux de division:0.00004, seuil apoptose:0.004 J12=25 durée=500000)

Le premier scénario ne permet pas de former une interface continue au bout de la durée 500000. les perturbations apportées par la division/apoptose ne sont pas suffisantes.

*Corrélations:*
On va vérifier la forme de la corrélation temporelle des spectres des interfaces pour les scénarios 1, 2, 3 et 4.

Scénario 1:

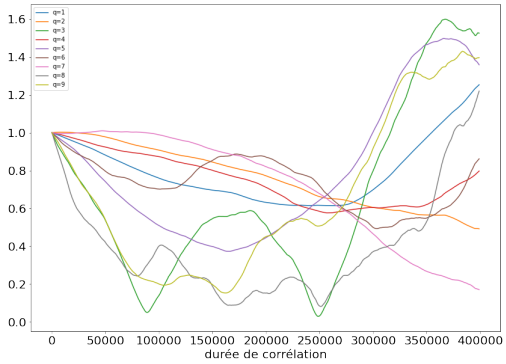



Figure 15: Corrélations temporelle des spectres des interfaces pour le scénario 1

Scénario 2:

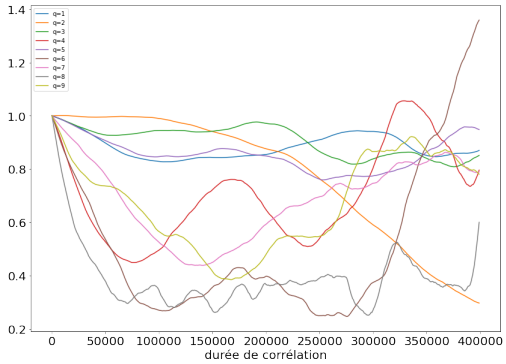



Figure 16: Corrélations temporelle des spectres des interfaces pour le scénario 2

Scénario 3:

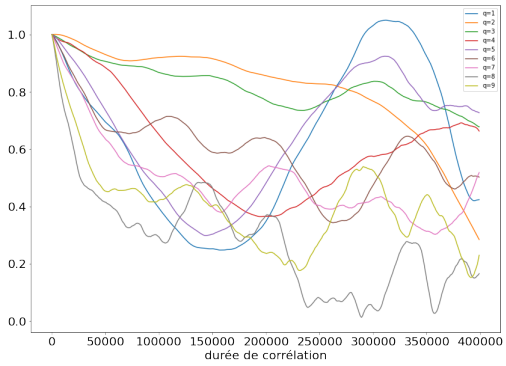



Figure 17: Corrélations temporelle des spectres des interfaces pour le scénario 3

Scénario 4:

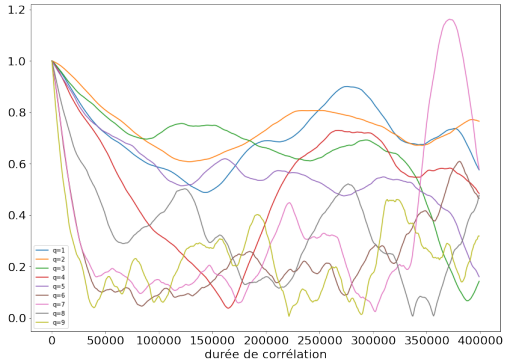



Figure 18: Corrélations temporelle des spectres des interfaces pour le scénario 4

Les temps de corrélation sont très longs (supérieurs à 250 000 pas) et les corrélations remontent ensuite en raison de durées de simulations limitées.

Les systèmes ne sont toujours pas dans un état stationnaire après ces durées de simulation.

*Largeur des interfaces pour les scénarios division/apoptose:*
On calcule $< h^2 >_t$ pour les scénarios 1, 2, 3 et 4.

On obtient les valeurs de la figure suivante:

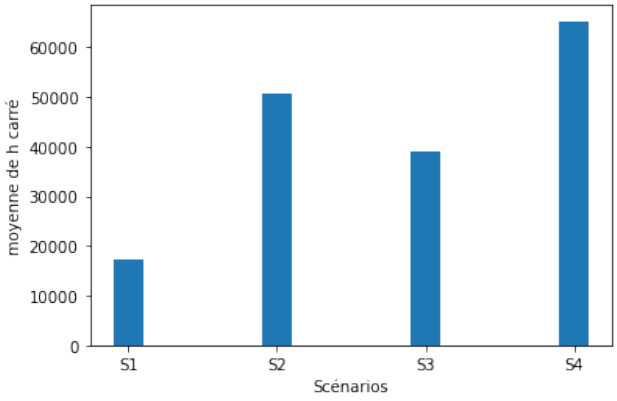



Figure 19: Largeur moyenne des interfaces pour les différents scénarios divsion/apoptose

Le scénario S1 correspond à une température effective de 300.

Il est intéressant de comparer les spectres moyens des interfaces pour les scénarios extrêmes 1 et 4 (taux de division et apoptose multipliés par 16). On obtient la figure ci-après:

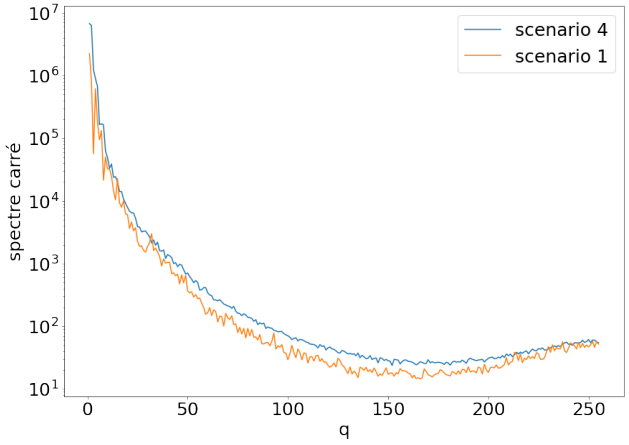



Figure 20: Spectres des interfaces pour les scénarions 1 et 5

Le spectre moyen est plus perturbé et de moindre amplitude dans le scénario 1.

## 9 Conclusion

Le programme modélisant l'interface entre deux tissus cellulaires actifs avec agitation thermique, division cellulaire, apopotose est opérationnel de même que le script en Python faisant l'analyse des interfaces et de leur spectre.

On a montré que l'évolution avec division et appotose est plus dynamique qu'avec la seule agitation thermique, l'intensité des fluctuations étant supérieure à tous les modes.

L'adéquation est bonne avec le modèle théorique, avec un module de flexion négligeable dans tous les cas.

La durée de la simulation, en lien avec le rythme de division cellulaire, doit être très grande afin que l'autocorrélation temporelle des spectres atteigne sa valeur limite.

La configuration division/apoptose provoque des réarrangements plus importants que la simple agitation thermique.

## References

[1] Durand, M., and Heu, J. (2019). Thermally driven order-disorder transition in two-dimensional soft cellular systems. Physical Review Letters, 123(18), 188001.

[2] Durand, M., and Guesnet, E. (2016). An efficient Cellular Potts Model algorithm that forbids cell fragmentation. Computer Physics Communications, 208, 54-63.

[3] Villemot, F., and Durand, M. (2021). Quasistatic rheology of soft cellular systems using the cellular Potts model. Physical Review E, 104(5), 055303.

# A Annex A: Exécution du programme

L'ensemble des fichiers source est contenue dans le repository github: `https://github.com/djachk/CPM_stage.git`
Pour compiler le programme et générer l'éxecutable, Utiliser le fichier Makefile et faire:
$ ./potts <fichier parametres> <directory de sortie>

# B Annex B: Lecture commentée du programme de modélisation existant

**Introduction**

Le code CPM écrit par Marc Durand a été utilisé dans le cadre d'un stage afin de simuler l'interface entre deux tissus cellulaires. Cette note vise à faciliter l'analyse du code.

Documents utiles :
- note Marc Durand sur l'Hamiltonien
- note Gian-Marco sur ce code CPM
- manuel librairie Cash

Le code est constitué des fichiers originaux :
- potts.h, potts.c
- bubble.h, bubble.c
- allocate.h, allocate.c
- operation.h, operation.c

Le code utilise pour affichage la librairie ***cash***. Il faut aussi vérifier la présence des librairies ***x11*** et ***png***. L'installation de ces librairies sous Linux se fait par :
$ sudo apt-get install libx11-dev
$ sudo apt-get install libpng-dev

On récupère la totalité du code en copiant et décompressant les fichiers ***cashpotts-0-1.zip*** (librairie cash) et ***Server-version.zip*** (les fichiers source).
Le programme principal *main* est dans *potts.c*. Un fichier *Makefile* est fourni.
On génère l'executable en tapant :
$ make potts

Plusieurs fichiers de paramètres sont fournis à titre d'exemple (cristalparam...).
On exécute en tapant :
$ ./potts fichierparam directory-output
On va supposer dans la suite que le directory de sortie s'appelle ***test***.
Nous allons passer en revue les principaux fichiers source. Ces fichiers sont contenus dans le repository https://github.com/djachk/CPM_original et correspondent au commit 0673104857951c6f683bccbd98908a2008a97fb5 ("original files")

**potts.c**

lignes 70-101
- on commence par lire les différents paramètres dans le fichier de paramètres. Pour cela on utilise la fonction *Indat* de la librairie ***cash*** (dans *basic.c*) qui repère le nom du paramètre et lit sa valeur dans le fichier de paramètres.

lignes 33-62 et 114-118
- on déclare les principales variables. Deux structures de données importantes sont *state* (le réseau),

matrice d'entiers et *cells*, du type *Cell* déclaré dans *allocate.h* qui contient l'ensemble des cellules. Les données sont indicées par le nombre de valeurs de la température. On se limitera dans notre cas à une seule température. La température est fixée à *temperature_min* jusqu'au temps *dispersetime* (où on introduira des *target_area* différentes selon le type de cellules) et on la positionnera alors à *température_max*.
- on fait des vérifications de cohérence sur les différentes durées.

lignes 196-216
- on ouvre les fichiers :
test.coords : contiendra à chaque intervalle d'écriture les coordonnées x et y des centres des cellules,
test.neighbours : contiendra par intervalle d'ecriture, par cellule, plusieurs infos dont nombre de voisins,
test.voisins : contiendra par cellule l'identité des cellules voisines,
test.subsystems : pas utilisé dans notre cas

et on écrit les en-têtes de ces fichiers.

- le nombre maximum de cellules (qui va servir dans les boucles) s'obtient en divisant la surface du réseau (multipliée par un facteur *fillfactor* car certaines cellules peuvent disparaître) par la surface cible des cellules (*target_area*).

lignes 274-286
- on constitue des tableaux de couleurs pour affichage (en positionnant le tableau *userCol* de la librairie cash, dans *color.c*)
- le test *neighbour_connected==6* fait référence à une configuration de cellules hexagonales qui ont un traitement particulier. On ignore ce cas.

lignes 288-300
- si l'affichage *movie* est positionné, on ouvre une fenêtre à l'écran, on crée deux directories désignés T=valeur-temperature et RAW-T = valeur-temperature pour chaque température (pour nous *temperature_min* et *température_max*), où on copiera les fichiers images du réseau à intervalles réguliers

lignes 305-309
- on alloue la mémoire au réseau *state* et à la structure *cells* (en la dimensionnant au maximum avec un nombre de cellules égal à *maxcells*)
- on ne traite pas le cas de l'annealing (« trempe », lissage des interfaces par abaissement de la température)

lignes 329-346
- on constitue le tableau des coefficent Jij de l'Hamiltonien et on lit les valeurs correspondantes dans le fichier parametres (toujours avec la fonction *InDat*)

ligne 349
- on initialise le réseau avec la fonction *InitBubblePlane* (qu'on verra dans *allocate.c*)

lignes 355-429
- si on reprend une simulation précedemment stockée (paramètre *load* différent de 0) on lit l'état stocké dans le fichier correspondant

ligne 438
Début de l'évolution temporelle
- boucle sur les pas de temps

ligne 439-440
- arrêt du programme si la souris est cliquée

lignes 441-456
- au temps *dispersetime*, on génère la polydispersité (fonction *GeneratePolydispersity* qu'on verra dans *allocate.c*), c'est-à-dire qu'on affecte des *target_area* différentes par type de cellules. On calcule l'energie (fonction *ComputeEnergy* qu'on verra dans *bubble.c*) et le périmètre des cellules (fonction *ComputePerimeter* qu'on verra dans dans *bubble.c*). On positionne la température à *temperature_max*.

lignes 458-495
Au délà de *dispersetime*, à chaque *saveinterval*, on écrit dans le fichier *savefile*.

lignes 503-635
- Au delà de *dispersetime*, à chaque *writinginterval*, on écrit dans les fichiers test.neighbours, test.coords, test.subsystems, test.voisins. Pour cela on détermine les voisins (fonction *FindNeighbours* qu'on verra dans *bubble.c*) et les coordonnées des centres des cellules (fonction *ComputeCenterCoords* qu'on verra dans *bubble.c*)

lignes 636-643
- Si *movie* est positionné on affiche dans la fenêtre (fonction *AffichageCouleurs* qu'on verra dans *bubble.c*)

lignes 695-740
- A chaque pas de temps on appelle la fonction *BubbleHamiltonian*, coeur de la simulation (qu'on verra dans *bubble.c*) (Rappel : dans notre cas $nb\_temperature = 1$)

lignes 743-748
- on recalcule l'energie pour vérifier que le calcul par itération est correct

lignes 759-778
- on libère la mémoire et on termine la simulation (donc au bout de *totaltime*, à moins d'un clic de la souris auparavant)

**allocate.c**

lignes 3-59
- fonction ***Cell AllocateCells(int n, int maxneighbour)***
On retourne la donnée *cells* après avoir alloué la mémoire nécessaire, ici avec le nombre maximum de cellules

lignes 100-134
- fonction ***int PutCell(TYPE **plane, int y, int x, TYPE m, int ncol, int nrow, int side1, int side2)*** (sera utilisée dans *InitBubblePlane*)
On positionne une nouvelle graine de cellule dans le réseau (après avoir vérifié que la place était libre). Cette graine est de taille 9 pixels ou bien correspond à un pavage du réseau. On tient compte de la périodicité grâce à la fonction *PeriodicWrap* (dans *operation.c*)

lignes 136-263
- fonction ***void InitBubblePlane(int init_config, float fillfactor,int nrow,int ncol, int target_area, double a1, double a2, TYPE **state, Cell cells, int sliding)***
On initialise le réseau en positionnant des graines de cellules. On determine combien de cellules on veut positionner au maximum. Puis plusieurs cas selon le nombre de cellules qu'on doit placer en

réalité et la façon dont on doit les placer (paramètre *init_config*).
Pour nous c'est un placement aléatoire bipériodique (cas 2). On calcule ainsi i et j, centre de la cellule puis on la place avec la fonction *PutCell*. On écrit dans la structure de données *cells* à la fin en positionnant *celltype* à 1 (les cellules seront diversifiées au temps *dispersetime*).

lignes 272-370
- fonction ***int GeneratePolydispersity(int polydispersity, int blob, int maxcells, double fillfactor, int nrow, int ncol, int target_area, double targetareamu2, int target_area2, double alpha, Cell cells)***
On affecte des *target_area* différentes selon le type de cellule selon le cas (1 seule, 2, 3 ou plusieurs). Pour nous ce sera 2 types de cellules. Avec le paramètre *blob* égal à 4, on répartit les cellules aléatoitrement à partir de la répartition alpha entre les deux populations.

**bubble.c**

lignes 11-185
- fonction ***double BubbleHamiltonian(int time, int dispersetime, int num, int neighbour_energy, int neighbour_copy, int neighbour_connected, int ncol, int nrow, TYPE **state, int mediumcell, int heat_bath, int** Jarray, Cell cells, double area_constraint , double temperature)***
C'est le coeur de la simulation. Les cellules évoluent en baissant l'energie globale ou sous l'action de l'agitation thermique (algorithme de Metropolis). L'évolution des cellules se fait par modification de la valeur d'un pixel d'une cellule (pris au hasard) en recopiant un pixel voisin différent (tout en préservant la simple connexité des cellules). Une telle modification est validée si l'energie globale baisse ce faisant ou bien avec une probabilité dépendant de la température (c'est l'algorithme de Metropolis).

Un appel de cette fonction correspond à N tentatives de modifications (N=taille du réseau). A chaque fois, on choit un pixel au hasard. On positionne un boolean *condwrap* pour savoir si on risque de déborder sur les bords et donc de devoir faire appel à la fonction *PeriodicWrap*. Toutes les actions ensuite sont dédoublées (selon *condwrap* vrai ou faux).
On calcule pour ce pixel le nombre de pixels identiques autour de lui (voisinage en croix, 4 voisins). S'il y a au moins un pixel différent, alors on range dans un tableau *state_copy* à 4 éléments les valeurs des pixels au voisinage (chaque valeur une seule fois).
On vérifie qu'on ne risque pas, pour la cellule à laquelle appartient ce pixel, d'entraîner la rupture de la simple connexité de cette cellule en modifiant ce pixel, en appelant la fonction *connected4* (qu'on verra plus loin, toujours dans *bubble.c*).
Si on peut en effet changer ce pixel, alors on prend un pixel candidat quelconque parmi la liste contenue dans *state_copy* (qui contient au plus 4 éléments). Si le candidat est en effet différent du pixel courant, on va tenter la recopie pixel=candidat.
On va calculer le delta d'energie qu'entrainerait le positionnement du pixel courant à la valeur de *candidat* en évaluant les deltas d'energie entre le pixel courant et les 20 pixels environnants. On accepte la recopie si le delta d'energie est négatif ou bien avec la probabilté égale au facteur de Boltzmann (exp(-deltaE/T).
On aura vérifié auparavant que la cellule cible reste également simplement connexe. On met à jour la structure *cells* en conséquence à la fin.

lignes 210-252
- fonction ***bool connected_4(TYPE **state, int nb_nei_id, bool condwrap, int pixel, int ncol, int nrow, int x, int y)***
On vérifie que la simple connexité d'une cellule n'est pas rompue en modifiant la valeur du pixel en x,y égal à l'un de ses 4 voisins (8 voisins pour la fonction *connected_8*).

lignes 290-310
- fonction ***void ComputePerimeter(int maxcells, Cell cells, int ncol, int nrow, TYPE **state, int***

***mediumcell, int neighbour_energy)***
On incrémente de 1 le périmètre chaque fois qu'un pixel de valeur différente de la valeur du pixel courant est détecté dans le voisinage de 20 pixels.

lignes 312-346
- fonction ***double ComputeEnergy(int maxcells, Cell cells, int ncol, int nrow, TYPE **state, int neighbour_energy, int** Jarray, double area_constraint)***
On additionne les deux termes de l'hamiltonien, énergie d'interaction entre les cellules deux à deux (en divisant par 2 pour ne pas compter deux fois) et énergie de compression par cellule.

lignes 348-418
- fonction ***void ComputeCenterCoords(Cell cells, int ncol, int nrow, TYPE **state, int nbrcells, int mediumcell)***
On trouve le centre de la cellule en égalant les aires à gauche, droite, haut, et bas.

lignes 441-496
- fonction ***void FindNeighbours(int maxcells, Cell cells, int ncol, int nrow, TYPE **state, int mediumcell, int neighbour_connected, int maxneighbours, int* side_interf12, int* side_interf10, int* side_interf20)***
On parcourt tout le réseau et pour chaque cellule on rajoute comme cellule voisine les cellules correspondant aux pixels voisins de chacun de des pixels de la cellule.

lignes 499-560
- fonction void ***AffichageCouleurs(int affichage, Cell cells, int ncol, int nrow, char *subdirname, char *subdirnameRAW ,TYPE **state, TYPE **nstate)***
On affecte la matrice *nstate* qui indique la couleur de chaque pixel, puis on appelle les fonctions de la librairie ***Cash***.