1 Introduction

The pdf is meant to contain all the comments to Marc Durand's code for CPM.

2 Model

We want to simulate a CPM with the following hamiltonian

$$\mathcal{H}_0 = \sum_{\langle i,j \rangle} J(\sigma_i, \sigma_j)\left(1 - \delta_{\sigma_i,\sigma_j}\right) + \sum_i \frac{\lambda_i}{A_i^0}\left(A_i - A_i^0\right)^2 \quad , \tag{1}$$

where ...
It must be compared with the hamiltonian

$$\mathcal{H}_0 = \sum_{\langle i,j \rangle} \gamma_{ij}\mathcal{L}_{ij} + \frac{B}{2A_0}\sum_i \left(A_i - A_0\right)^2 \quad , \tag{2}$$

where $B = 2\lambda$ and $\gamma_{ij} = zJ(i,j)$. z is the line to area factor, $z \simeq 11.3$, estimated numerically and largely used in the literature.
We are interested in adding a confining potential in order to measure the rpessure exerted from the system. We therefore want to study the following hamiltonian

$$\mathcal{H} = \mathcal{H}_0 + V(y) \quad , \tag{3}$$

Add a picture. let's observe that the previous hamiltonian still simualte a passive term, *i.e.* no
We will measure the pressure as

$$P = \int_{x_b}^{\infty} \rho(x,y)\partial_y V(x,y) = \int \sum_{cell:k} \sum_{\text{site } i\in k} \delta(x - x_i)\delta(y - y_i) = \tag{4}$$

$$= \frac{1}{l_x}\sum_{cell:k} \sum_{\text{site } i\in k} \partial_y V(x_i, y_i) \quad , \tag{5}$$

where in the last line we have used the translational invariance of the potential.

2.1 Update of BubbleHamiltonian

We write the updated version of BubbleHamiltonian in BubbleHamiltonian_GS. We want to incorporate in the montecarlo step the possibility to have a penalty associated to the rpesence of a confining potential.
To do that we have to consider the potential when we compute the ΔE.
The Montecarlo step selects a random pixel of the domain and look for a proper switch candidate in the first four neighbours. Once the candidate is chosen (see BubbleHamiltonian description for the way it is chosen) it labels it with icandidate. The test is therefore to check if pixel can or not be changed into icandiate. Here we have to take into account the presence of the external (confining) potential.
At the moment we are not interested in computing the penalty associated to each cell, where we call penalty the fractio of the cell inside the potential and global penalty the total potential acting on the system. The real change in energy happens when pixel or icandidate are one medium and the other cell. If both pixel and candidate are medium, nothing happens and no global-penalty must be added. The absence of a change in the global penalty means that ΔE has only surface and area contributes.
This means that we have to check wether or not one between pixel and icandidate is medium and then add the contribute from the confining wall. We can do that in the following way

```
// Update of the Bubblehamiltonian function
if( pixel*icandidate == 0 ) // since they are different, only one of them is zero, i.e. the medium
```

```
{
  // If pixel is zero, icandidate is a cell. Therefore the final energy has a contribution from the
      potential
  if( pixel == 0 ) delta_E += potential( y, cells.targetarea[pixel] );
  // If icandidate is zero, pixel is a cell. Therefore the initial energy has a contribution from the
      potential
  if( icandidate == 0) delta_E -= potential( y, cells.targetarea[pixel]);
}
```

3 Organization

4 Variables

In this section I would like to collect all the variable and what they do and where they used (if possible). I will skip the obvious ones. I will divide this section in two: Input variables (all the variables given by the users through the params file), and Core Variables (all the variables defined inside the code).

4.1 Input Variables

seed Seed of the simulation for the random number generator. Remember to change it each time

dispersetime It is the time at which I create or not a polydispersity in my system. It is used in some check to control that I store properly the images and I make properly the video. It is used to access the GeneratePolydispersity which is in allocate library.

polydispersity.

ncol. Number of columns.

nrow. Number of rows.

fillfactor. Fraction of area occupied by the cells. It is used in InitBubblePlane and GeneratePolydispersity.

init_config. we are working now with its value at one. TBA the others.

Jarray. We introduce the interactions between the different kind of cells. The medium is considered as a cell of type 0. The code allows to consider more than one kind of medium but there is no debug. This means it could stop working with more than one medium. The real input are the coefficient J_{ij}.

area constraint. The coefficient λ.

targetareamu2. It enters in GeneratePolydispersity. With alpha, it quantifies a possible difference in the target area of a polidisperse system. For a bidisperse system, if it is zero, it implies that the two type of cell have the same target area. the two different target area are compute as $target_{1/2} = (int)(meantargetarea * (1. \pm targetareamu2 * \sqrt{\alpha/(1-\alpha)}))$;

4.2 Core Variables

maxcells It is the maximum number of cells we can put inside. It is obtained as maxcells = (int)(ncol $*$ nrow $*$ fillfactor / target_area)+1

meantargetarea Variable not really used in the case of a monodisperse system.

deleted. Is the number of deleted cells for area restrictions.

mediumcell. Is the number of medium cells. It is set to zero initially. It is used in different functions.

time load. It loads the finalt ime of the simulation we are resuming

totaltime. Final time of the simulation. It is not computed as totaltime = time_load + totaltime. we can change this if needed.

neighbour connected. Numbers of neighbour for each pizel that I have to check.

MAXNEIGHBOURS. It is set at 20 at the beginning of the code. This is the maximum number of neighbours that a cell can hav. We want it to be large enough to avoid problems but small enough to avoid unuseful loop in the code. 20 is a good interpolation.

state. it is an int ** that says in each point of the lattice which is the index of the cell occupying that place. Ok it makes sense. Then I can access the cell type by using cellIdx = state[i][j] and use cells.celltype[cellIdx].

neighbour-energy. number of neighbours to check

neighbour-copy. It is set to 4. It is the number of pixel we want to check.

5 Library allocate.c

GeneratePolydispersity. It is a function that

> **Monodisperse**. It creates a monodisperse system. No problem with meantarget area since at the beginning of the code meantargetarea is set equal to target_area. Teh function return the number of deleted cells.
>
> **Bidisperse.** TBA
>
> **Tridisperse.** TBA

InitBubblePlane. It uses the different init condition and a function call PutCell( ). It nitializes all the bubbles with an area equal to 9, *i.e* each grain is a 3x3 square. We expect them to grow immediatly at the beginning. One small remark. It is possible that, while cells are expanding, one cell eat another one. This is why we usually use a fillfactor which is bigger than one.

PutCell. It checks if the space around it is occupied or not?

AllocateCells. The function checks if the memory is well allocated.

FreeCells. It free the pointer inside the cell type.

Duplicate. It makes a copy of the lattice

AssignNormalTargetarea. It is used if we want to create a polydisperse system with the targetarea normally distribuited. It is not of our business.

6 Library Bubble.c

ComputeCenterCoords. The function computes the geometric center of the cell.

FindNeighbours. Find the index of the neighbours cells.

ComputePerimeter. Conmpute the perimeter consideering the 20 nn of a cell. In this way, especially when I am at the border, I count more cell than the one I should. This is way I consider the line to area factor to correct the value of the perimeter. The value of z change according to many factor (temperatuer, number of nn, targetarea ...),however, it is between 10.5 and 11.3 in the literature for our parameters. In the measurement of the pressure, this factor plays no role since we want to keep all the prefactor constant while changing the stiffness of the potential.

ComputeEnergy. Same consideration as before with the extra factor.

BubbleHamiltonian. Main function. In this function we compute the MonteCarlo step.

Connected 4. This is one of the functions that preserve the cell fragmentation. It is used in two different context, wether the pixel chosen for the update is medium or not. In the two cases we have a different usage. With different usage I mean that the int nb-nei-id number passed as argument is not the same. In the first case (the pixel is not medium) this number is number of neighbours equal to the pixel. In the second case, it is also used with the number of neighbours equal to the icandidate. Let's start by commenting the first case.
First, the number of neighbours can be checked only if the number is 2 or 3. The case are therefore

1. this means that is an appendix, i.e. it can be deleted with no consequence for the appendix.
2. In this case, I want to update only if the pixel I chose is an angle. I thereofre check that the 'corner' are safe. Fig 2
3. It checks if the T are not broke. It wants to avoid situation in which I break the connectivity of an appendix (Fig 1 of my book, sorry for the reader.)
4. This case never happens.

7 Library Potts.c

8 Library operation.c

PeriodicWrap. It is used to take into account the periodic boundary condition.

9 To ask

- What does this line do TYPE **state[nb _temperature] line 107 of Potts.c
- What is the role of nb_temperature. It is set to 1 and avoid many things. It is also used to create a different nmber of cells array.
- Is it ever updated the value of medium cell?
- Role of dispersetime.
- Line 179 neighbour connected != 6, why it should be 6?
- Type state is an int ** that tells me the kind of cell that cover that particular position of the lattice. If I have only one kkiind of cell , e.g. k = 1, I will have only k = 1 in state?
- Can we go together through the role of state?
- What pragma does?
- Where do i compute the area of the single cell?
- Is there any advantage in redefining the variables? Like in bubblehamiltonian with nb nei id.
- Do i have to start nrow and ncol from 1 isnted than from 0

10 Change to implement

Introduce the penalty given by the potential

Compute the energy properly at the beginning.

Create a proper output to store the total force exerted by the wall.

Crete a struct wall with all the info. Can we define inside a pointer to a function? In this way we could also store the information about the potential we are using. Esempio

References