



**Modeling grain boundary
anisotropy-driven microstructural
evolution with arbitrary grain
orientation**

Treball realitzat per:
Josep Maria Gras Ribot

Dirigit per:
**Dr. Brandon Runnels, University of Colorado
Colorado Springs**

Tutor:
Ignacio Carol Vilasarau

Grau en:
Enginyeria Civil

Barcelona, **23/06/1993**

Departament d'Enginyeria del Terreny, Cartogràfica i
Geofísica

Index

1	Introduction to Grain Boundaries	6
1.1	Grain Boundaries and macro-scale properties	6
1.2	State of the art	6
2	Objective of the project	8
3	Problem to be solved	8
3.1	Phase field method	8
3.2	Parameters of the function	9
3.2.1	Amplitude of the boundary	10
3.2.2	Energy of the Boundary	10
3.2.3	The mobility of the boundary	10
3.2.4	Derived parameters	11
3.3	Initial conditions	12
3.3.1	Random	12
3.3.2	Bar	13
3.3.3	Bar with perturbations	14
3.3.4	Square	15
3.3.5	Circle	16
3.3.6	Cross	17
3.4	Boundary conditions	17
4	Experiments	18
4.1	Experiments with Random IC	19
4.2	Experiments with straight bar	21
4.3	Experiments in a bar with a perturbation as IC	23
4.4	Experiments with a Circle as IC	25
4.5	Experiments with a Square as IC	27
4.6	Experiments with a Greek Cross as IC	29
5	Conclusions	30
6	Future work	30
7	Acknowledgements	31
8	References	31

Índex de figures

1	Tilt and Twist of a bicrystal	6
2	Energy depending to the normal vector of the surfaces [1]	7
3	γ influence in GB configuration [2]	7
4	Biphase Phase Field profile	8
5	Energy function depending on the angle of the boundary	10
6	Random IC initial value of order parameter	12
7	Random IC initial boundary energy	12
8	Bar IC initial value of order parameter	13
9	Bar IC initial boundary energy	13
10	Bar with perturbations IC initial value of order parameter	14
11	Bar with perturbations IC initial boundary energy	14
12	Square IC initial value of order parameter	15
13	Square IC initial boundary energy	15
14	Circle IC initial value of order parameter	16
15	Circle IC initial boundary energy	16
16	Cross IC initial value of order parameter	17
17	Cross IC initial boundary energy	17
18	Energy function with $\theta_0 = 0^\circ$	18
19	Energy function with $\theta_0 = 60^\circ$	18
20	Random Isotropic evolution	19
21	Random Anisotropic evolution, $\theta_0 = 0^\circ$	19
22	Random Anisotropic evolution, $\theta_0 = 60^\circ$	20
23	Bar Isotropic evolution	21
24	Bar Anisotropic evolution, $\theta_0 = 60^\circ$	21
25	Bar Anisotropic evolution, $\theta_0 = 60^\circ$	22
26	Bar with perturbations Isotropic evolution	23
27	Bar with perturbations Anisotropic evolution, $\theta_0 = 0^\circ$	23
28	Bar with perturbations Anisotropic evolution, $\theta_0 = 60^\circ$	24
29	Circle Isotropic evolution	25
30	Circle Anisotropic evolution, $\theta_0 = 0^\circ$	25
31	Circle Anisotropic evolution, $\theta_0 = 60^\circ$	26
32	Square Isotropic evolution	27
33	Square Anisotropic evolution, $\theta_0 = 0^\circ$	27

34	Square Anisotropic evolution, $\theta_0 = 60^\circ$	28
35	Cross Isotropic evolution	29
36	Cross Anisotropic evolution, $\theta_0 = 0^\circ$	29
37	Cross Anisotropic evolution, $\theta_0 = 60^\circ$	30

Resum

Algunes de les propietats més importants dels materials pol.licristalins, com la fluència, la conductivitat tèrmica o elèctrica, o la resistència a impactes d'alta velocitat (superior a la de ressonància del material) no depenen només de la configuració dels cristalls (ja siguin cúbics, o d'altres formes, centrats a les cares o al centre del cristall), sinò de les fronteres entre els grans del material i de l'energia lliure que s'hi acumula. El present treball és una fase d'un estudi de llarg recorregut que s'està duent a terme a la Universitat de Colorado a Colorado Springs (UCCS), l'objectiu d'aquest treball ha sigut crear un model computacional que demostrí l'influència de la anisotropia dels grans en la configuració de les fronteres i de l'energia lliure que hi queda emmagatzemada. Aquest treball vol servir de base a pròxims models on s'estudii els efectes sobre la configuració de les interfícies dels grans de diferents accions o condicions externes, ja siguin tensionals, térmiques o d'altres tipus, i com això afecta les propietats del material, especialment la resistència a impactes a gran velocitat.

Abstract

Some of the most important properties of the polycrystalline materials, such as yield resistance, electrical and thermal conductivity and the resistance to the high velocity impacts; do not depend only in the basic structure of the crystals but in the grain boundaries configuration, and the free energy that they have. The present work is a new phase for the research that is been carried out at the University of Colorado, the aim of this work was to create a computational model that shows the influence that the grain anisotropy of the grains have on the evolution of the grain boundaries of the materials. This work wants to serve as a base for future models to study the configuration of the grain boundaries under different kind o external or internal actions, and the effect on the material properties, especially the resistance to high velocity impacts.

1. Introduction to Grain Boundaries

"The art of metallography is mature and the forms in which various micro-constituents appear are well known. Investigations almost have disclosed the importance of the exact manner of distribution of phases on the physical properties and usefulness of an alloy. Surprisingly, however, relatively little attention has been paid to the forces that are responsible for the particular and varied spatial arrangements of grains and phases that are observed. Like the anatomist, the metallurgist has been more concerned with form and function than with origins" [3]

With this statement starts one of the first essays on Grain Boundaries, and Interfaces, this was written in 1948, and was the first to look into the Interfaces of the metallic materials and found that some of them had a very low energy but others don't, It was the first to claim that the anisotropy of the orientations of the grains must be important on their configuration.

1.1. Grain Boundaries and macro-scale properties

"Interfaces often exist in polycrystalline, multiphase materials in a large number of configurations" [4]. We call grain boundary to an interface in an homogeneous and homo-phase material, where the existence of the boundary is driven by the different orientations of the grains of the material.

"In fact, it has been commonly remarked that the study of the behaviour of polycrystalline materials is often reduced to the study of the behaviour of their interfaces. Since the properties of interfaces between crystals depend upon their structure, the study of the structure and properties of interfaces has emerged as a central area in the larger field of the science and engineering of materials" [4]

Following this statement, we can say that most of the properties of the crystalline materials are affected by the configuration and structure of their boundaries, because of this the study of the boundaries and their structure is so important.

1.2. State of the art

In the last decades, thanks to the development of high-resolution techniques for the study of the atomic structure and to the exponential increase in the modelling capacity (thanks to the rapid rise of powerful computer simulation methods and better theory); the study of the interfaces in crystalline materials had grown a lot.

Some of the recent papers are focused on finding models to determine the grain boundary free energy in each possible situation.

The misorientation of the boundaries can be represented by the combination of two angles, one for tilt and one for twist.

Figura 1: Tilt and Twist of a bicrystal

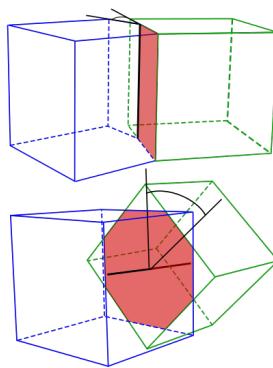
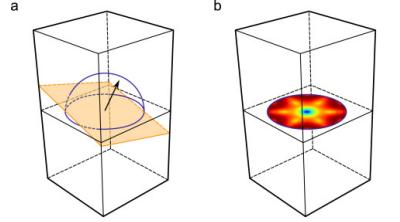


Figura 2: Energy depending to the normal vector of the surfaces [1]

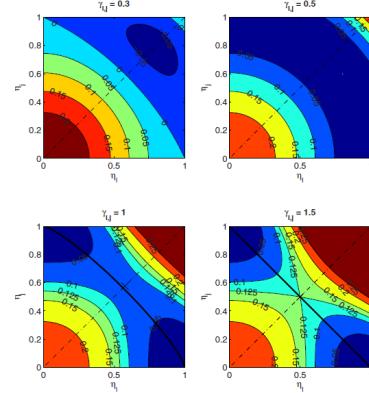


Fixing the orientation relationship between the upper and lower crystals, all remaining interface configurations are determined by normal vector, mapping interface energy to the unit sphere.

The upper unit 2-hemisphere is homeomorphic to the unit 2-ball over which the energy is plotted. For example, the value at the center corresponds to the normal vector $\mathbf{n} = [0 \ 0 \ 1]^T$.

Also, some successfully approaches using the Phase Field method have been done, that has been so helpful for the development of this project. As an example the article "Quantitative analysis of grain boundary properties in a generalized phase field model for grain growth in anisotropic systems" (N. Moelans, B. Blanpain, and P. Wollants), that have developed a set of parameters that suits perfectly for the purpose of this project.

Figura 3: γ influence in GB configuration [2]



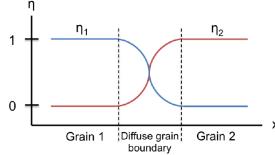
2. Objective of the project

The objective of this project is to show the effect of the anisotropy of the grain boundaries in the evolution on their configuration. For that, the the simplest case will be studied, with two possible configurations only and a simple 2D tilt as possible misorientation.

3. Problem to be solved

We will build a model that drives the solution between two different phases, the Phase Field model has two equilibrium points for each phase η_i .

Figura 4: Biphase Phase Field profile



3.1. Phase field method

Define the following energy functional

$$F[\eta_1, \dots, \eta_n] = \int_{\Omega} f(\eta_1, \dots, \eta_n, \nabla \eta_1, \dots, \nabla \eta_n) dV \quad (1)$$

Evolve η_i according to the equation

$$\frac{\partial \eta_i}{\partial t} = -M \frac{\delta f}{\delta \eta_i} \quad (2)$$

where the r.h.s. is the *variational* derivative of f , that is

$$\frac{\delta f}{\delta \eta_i} = \frac{\partial f}{\partial \eta_i} - \sum_j \frac{\partial}{\partial x_j} \frac{\partial f}{\partial \eta_{i,j}} \quad (3)$$

Let the integrand have the following form

$$f(\eta_1, \dots, \eta_n, \nabla \eta_1, \dots, \nabla \eta_n) = W(\eta_1, \dots, \eta_n) + \sum_i \kappa_i(\theta) \|\nabla \eta_i\|^2 \quad (4)$$

where θ is the angle of $\nabla \eta$ computed using \arctan . The free energy functional will be defined as

$$W(\eta_1, \dots, \eta_n) = \mu \sum_{p=1}^n \left(\frac{1}{4} \eta_p^4 - \frac{1}{2} \eta_p^2 + \frac{\gamma}{2} \sum_{q \neq p} \eta_p^2 \eta_q^2 \right) \quad (5)$$

It is the only part that depends on η_i , so we compute the derivative here:

$$\frac{\partial f}{\partial \eta_i} = \frac{\partial W}{\partial \eta_i} = \mu \left(\eta_i^3 - \eta_i + \gamma \eta_i \sum_{q \neq i} \eta_q^2 \right) \quad (6)$$

We choose to represent the grain boundary energy function in terms of θ where

$$\theta = \tan^{-1} \left(\frac{\eta_{,2}}{\eta_{,1}} \right) \quad (7)$$

To compute the subsequent derivatives, we will need the following:

$$\frac{\partial \theta}{\partial \eta_{,1}} = \frac{1}{1 + (\frac{\eta_{,2}}{\eta_{,1}})^2} \left(-\frac{\eta_{,2}}{\eta_{,1}^2} \right) = -\frac{\eta_{,2}}{\eta_{,1}^2 + \eta_{,2}^2} = -\frac{\eta_{,2}}{\|\nabla \eta\|^2} \quad (8)$$

$$\frac{\partial \theta}{\partial \eta_{,2}} = \frac{1}{1 + (\frac{\eta_{,2}}{\eta_{,1}})^2} \left(\frac{1}{\eta_{,1}} \right) = \frac{\eta_{,1}}{\eta_{,1}^2 + \eta_{,2}^2} = \frac{\eta_{,1}}{\|\nabla \eta\|^2} \quad (9)$$

Armed with these derivatives, let us compute the gradient derivative terms:

$$f = W(\eta_1, \dots, \eta_n) + \frac{1}{2} \sum_k \kappa_k(\theta_k) \|\nabla \eta_k\|^2 \quad (10)$$

$$\frac{\partial f}{\partial \eta_{i,j}} = \frac{1}{2} \frac{d\kappa_i(\theta_i)}{d\theta_i} \underbrace{\frac{\partial \theta}{\partial \eta_{i,j}} \|\nabla \eta_i\|^2}_{\text{indep. of } \eta_{i,j}} + \kappa_i(\theta_i) \eta_{i,j} \quad (11)$$

$$\frac{\partial}{\partial x_j} \frac{\partial f}{\partial \eta_{i,j}} = \frac{\partial}{\partial x_j} \left[\frac{1}{2} \frac{d\kappa_i(\theta_i)}{d\theta_i} \left(\frac{\partial \theta}{\partial \eta_{i,j}} \|\nabla \eta_i\|^2 \right) + \kappa_i(\theta_i) \eta_{i,j} \right] \quad (12)$$

$$= \frac{1}{2} \frac{d^2 \kappa_i(\theta_i)}{d\theta_i^2} \frac{\partial \theta_i}{\partial \eta_{i,j}} \frac{\partial \eta_{i,j}}{\partial x_j} \underbrace{\left(\frac{\partial \theta}{\partial \eta_{i,j}} \|\nabla \eta_i\|^2 \right)}_{\text{indep. of } \eta_{i,j}} + \frac{d\kappa_i(\theta_i)}{d\theta_i} \frac{\partial \theta_i}{\partial \eta_{i,j}} \frac{\partial \eta_{i,j}}{\partial x_j} \eta_{i,j} + \kappa_i(\theta_i) \frac{\partial \eta_{i,j}}{\partial x_j} \quad (13)$$

We will see that these simplify a little when we introduce specific values for j :

$$\frac{\partial}{\partial x_1} \frac{\partial f}{\partial \eta_{i,1}} = \frac{\eta_{,2}^2}{2\|\nabla \eta_i\|^2} \frac{d^2 \kappa_i(\theta_i)}{d\theta_i^2} \frac{\partial \eta_{i,1}}{\partial x_1} - \frac{\eta_{i,1}\eta_{i,2}}{\|\nabla \eta_i\|^2} \frac{d\kappa_i(\theta_i)}{d\theta_i} \frac{\partial \eta_{i,1}}{\partial x_1} + \kappa_i(\theta_i) \frac{\partial \eta_{i,1}}{\partial x_1} \quad (14)$$

$$\frac{\partial}{\partial x_2} \frac{\partial f}{\partial \eta_{i,2}} = \frac{\eta_{,1}^2}{2\|\nabla \eta_i\|^2} \frac{d^2 \kappa_i(\theta_i)}{d\theta_i^2} \frac{\partial \eta_{i,2}}{\partial x_2} - \frac{\eta_{i,1}\eta_{i,2}}{\|\nabla \eta_i\|^2} \frac{d\kappa_i(\theta_i)}{d\theta_i} \frac{\partial \eta_{i,2}}{\partial x_2} + \kappa_i(\theta_i) \frac{\partial \eta_{i,2}}{\partial x_2} \quad (15)$$

Combining these two and collecting terms, we have

$$\frac{\partial}{\partial x_1} \frac{\partial f}{\partial \eta_{i,1}} + \frac{\partial}{\partial x_2} \frac{\partial f}{\partial \eta_{i,2}} = \frac{1}{2\|\nabla \eta_i\|^2} \frac{d^2 \kappa_i(\theta_i)}{d\theta_i^2} \left(\eta_{,2}^2 \frac{\partial \eta_{i,1}}{\partial x_1} + \eta_{,1}^2 \frac{\partial \eta_{i,2}}{\partial x_2} \right) + \left(\kappa_i(\theta_i) - \frac{\eta_{i,1}\eta_{i,2}}{\|\nabla \eta_i\|^2} \frac{d\kappa_i(\theta_i)}{d\theta_i} \right) \left(\frac{\partial \eta_{i,1}}{\partial x_1} + \frac{\partial \eta_{i,2}}{\partial x_2} \right) \quad (16)$$

Finally, our evolution equation is given by

$$\begin{aligned} \frac{d\eta_i}{dt} = & -L \left[\mu \left(\eta_i^3 - \eta_i + 2\gamma_i \eta_i \sum_{j \neq i} \eta_j^2 \right) \right. \\ & \left. + \frac{1}{2\|\nabla \eta_i\|^2} \frac{d^2 \kappa_i(\theta_i)}{d\theta_i^2} \left(\eta_{,2}^2 \frac{\partial \eta_{i,1}}{\partial x_1} + \eta_{,1}^2 \frac{\partial \eta_{i,2}}{\partial x_2} \right) + \left(\kappa_i(\theta_i) - \frac{\eta_{i,1}\eta_{i,2}}{\|\nabla \eta_i\|^2} \frac{d\kappa_i(\theta_i)}{d\theta_i} \right) \left(\frac{\partial \eta_{i,1}}{\partial x_1} + \frac{\partial \eta_{i,2}}{\partial x_2} \right) \right] \end{aligned} \quad (17)$$

3.2. Parameters of the function

For the function that we have above, when solving we have to define some parameters that answer to the possible structures of the boundaries and their properties. The general reference for this section is Moelans *et al* [2].

3.2.1. Amplitude of the boundary

The amplitude of the boundary is one of the most important parameters, if it is too large it drives to an unstable boundary, but if they are too small the mobility of the boundary is lowered and then it is very difficult for the interface to change making it too stable no allowing the boundary to find a lower energy equilibrium.

As reference, the experience says to take a boundary width of five times the spacing of the merge of the model.

3.2.2. Energy of the Boundary

σ_{gb} is the most important parameter, the higher σ_{gb} is, the more unstable the boundary is, and then is easier for this point to change the configuration.

For computing our model, it was opted for a very simplified energy function, because we are working with a planar simulation, and the case of study of a simple tilt, that makes that all the energy function is only depending on the boundaries orientation.

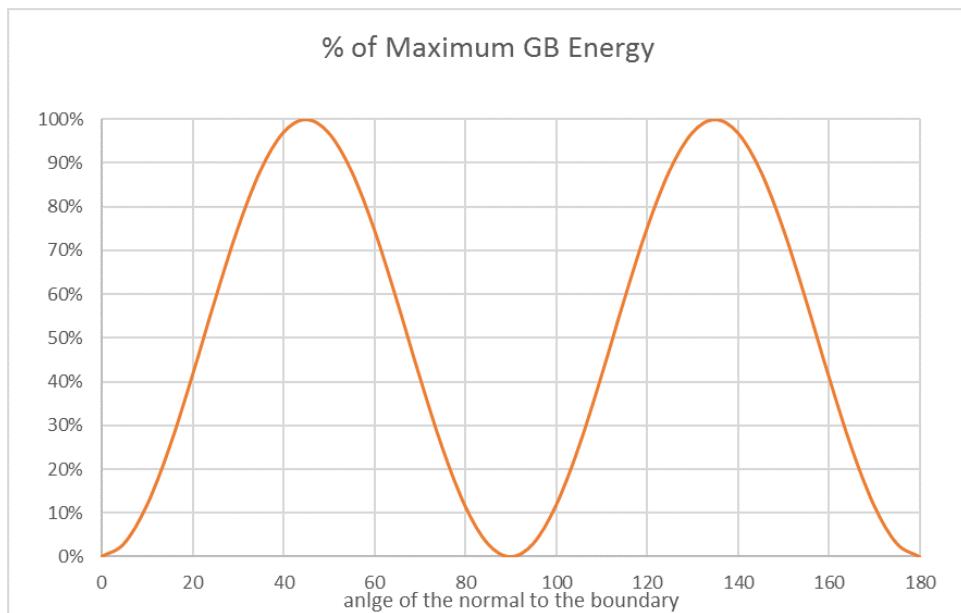
For all that, the energy function used is the following:

$$\sigma_{gb} = \sin^2(2\theta_i + \theta_0) \quad (18)$$

Where σ_i is the angle of the boundary in each point, and σ_0 is the misorientation of the two configurations that we are looking at.

Creating a profile of the Boundary Energy as the one below:

Figura 5: Energy function depending on the angle of the boundary



3.2.3. The mobility of the boundary

Because we are working with an indeterminate material we will keep it equal to 1. That does not affect to the model, because the mobility its also driven by I_{gb} that will be enough.

3.2.4. Derived parameters

Following recent research published in the magazine Physical Review B in 2008 [2] we take the parameters for uniform systems. All of them have a fixed value or are depending only in the grain boundary energy.

$$f_{0,saddle} = 0.23 \quad L = \frac{4}{3} \frac{m_{gb}}{l_{gb}} \quad \kappa = \frac{3}{4} \sigma_{gb} m_{gb} \quad \mu \approx \frac{3}{4} \frac{1}{f_{0,saddle}(\gamma)} \frac{\sigma_{gb}}{l_{gb}}, \quad (19)$$

3.3. Initial conditions

For the purpose of studying the effect of the anisotropy several initial conditions have been designed, because we observed that the effect of the initial conditions is so important in some of the possible experiments that we created some initial conditions to see the effect of the anisotropy in each of them.

3.3.1. Random

The random IC, takes a value between 0 and 1 in all the merge for η_1 and because $\eta_1 + \eta_2 = 1$ then $\eta_2 = 1 - \eta_1$.

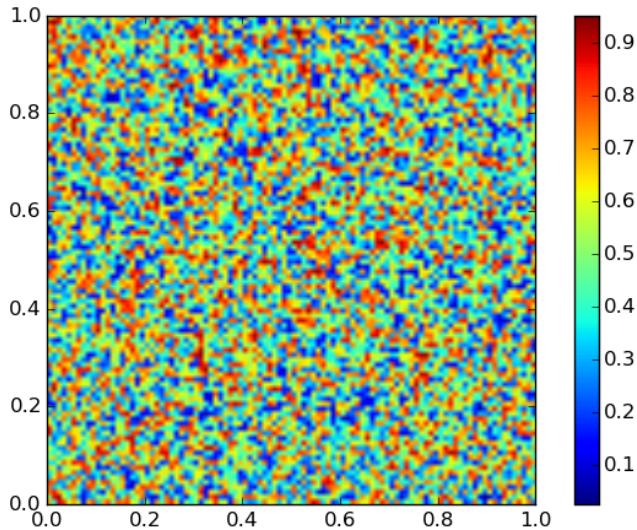


Figura 6: Random IC initial value of order parameter

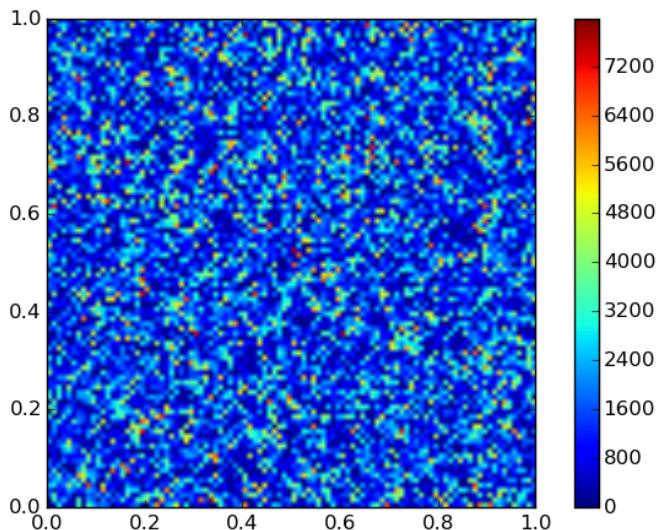


Figura 7: Random IC initial boundary energy

3.3.2. Bar

This Initial Condition consists in a Bar that occupies half of the field. In this Bar $\eta_1 = 1$ and $\eta_2 = 0$ and the opposite outside the Bar.

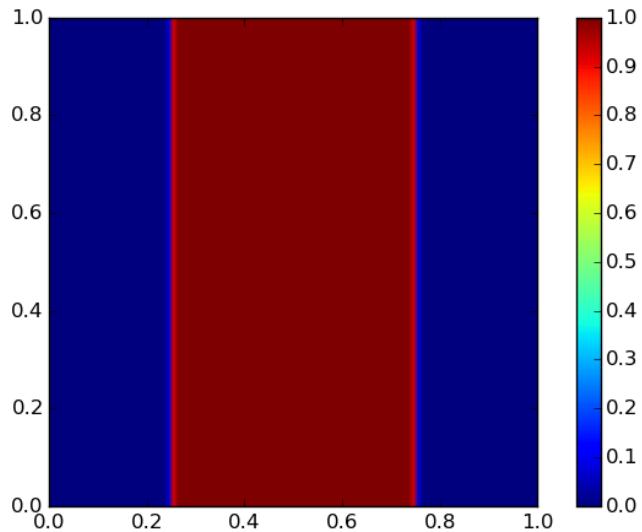


Figura 8: Bar IC initial value of order parameter

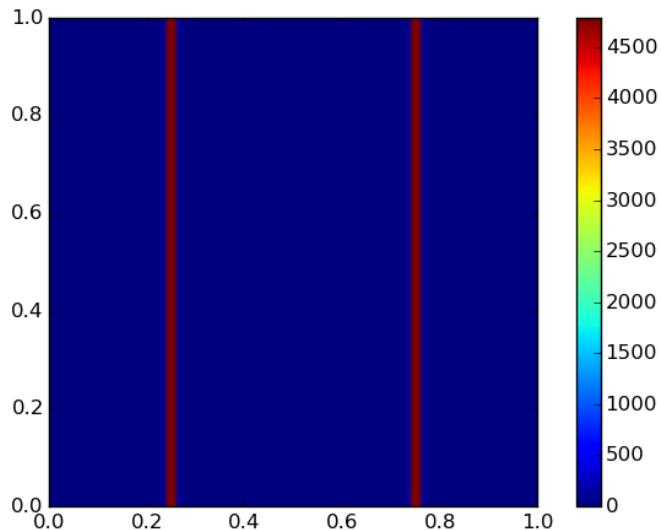


Figura 9: Bar IC initial boundary energy

3.3.3. Bar with perturbations

This Initial Condition is the same as the Bar above, but a periodic perturbation has been added on it, to study the effect on a non-straight configuration.

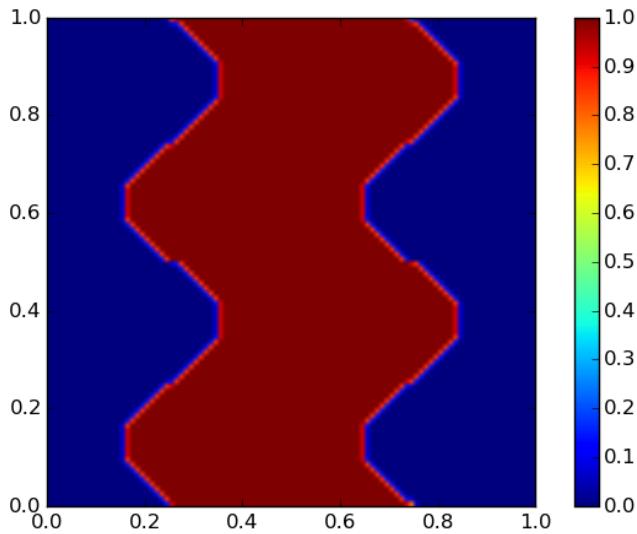


Figura 10: Bar with perturbations IC initial value of order parameter

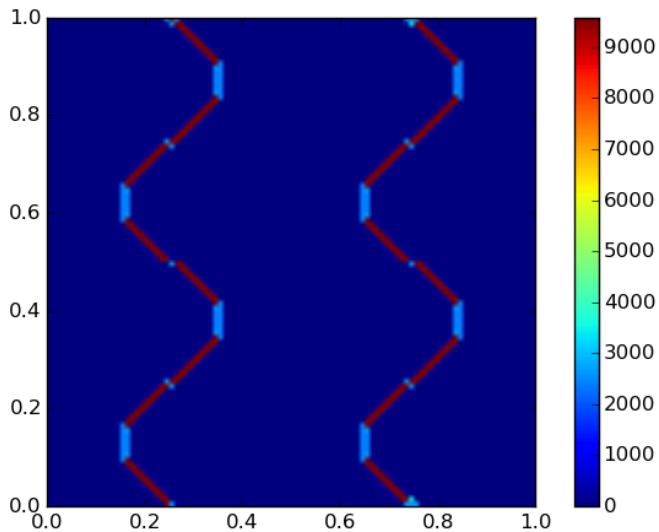


Figura 11: Bar with perturbations IC initial boundary energy

3.3.4. Square

This IC is interesting because it "cuts" the infinite dimension that the bar have, and show the local effects of the model.

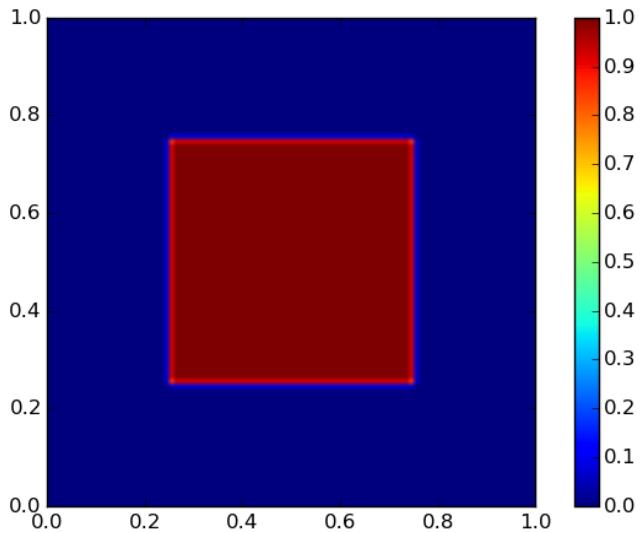


Figura 12: Square IC initial value of order parameter

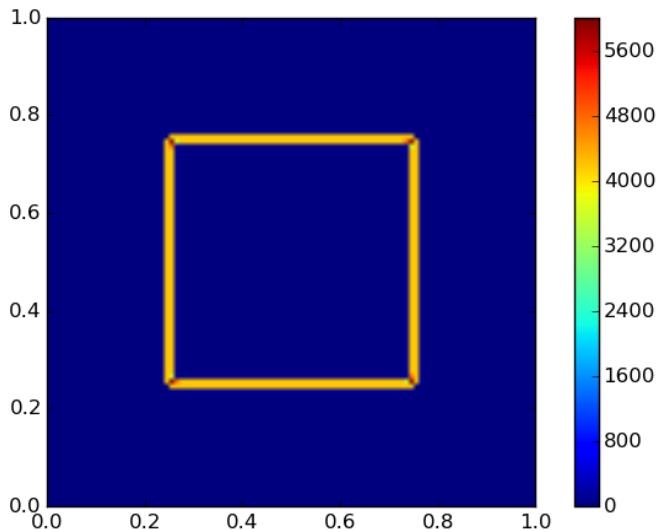


Figura 13: Square IC initial boundary energy

3.3.5. Circle

This IC is interesting because it does not have straight lines, then its useful for studying the effect of the model in curve contours.

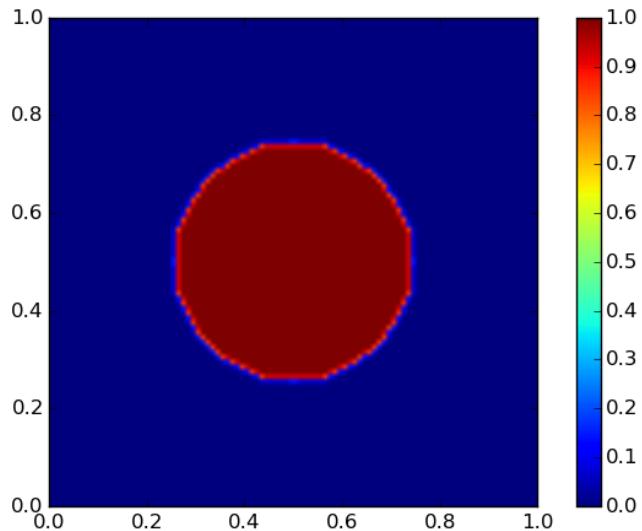


Figura 14: Circle IC initial value of order parameter

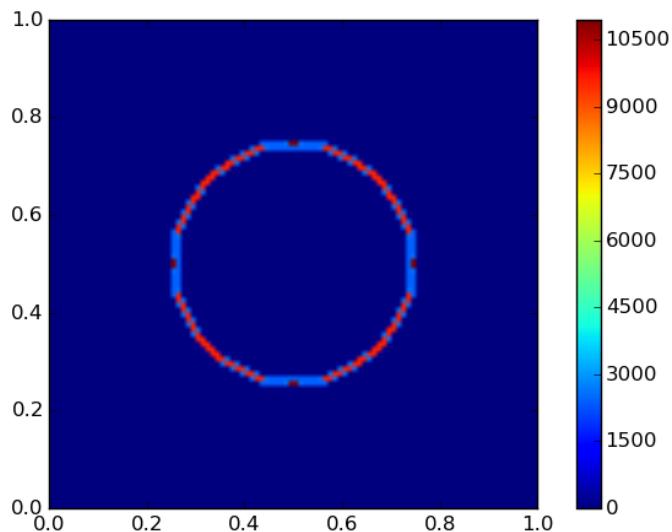


Figura 15: Circle IC initial boundary energy

3.3.6. Cross

This IC will show if orientation of the merge is important, and then if we need higher resolution to evade the merge direction effect.

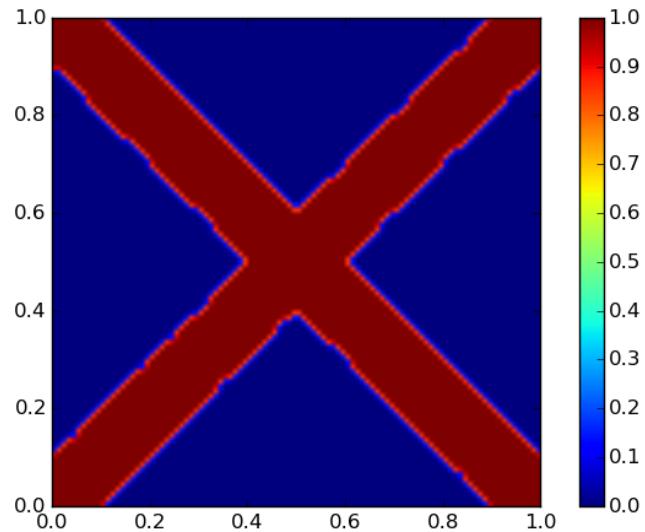


Figura 16: Cross IC initial value of order parameter

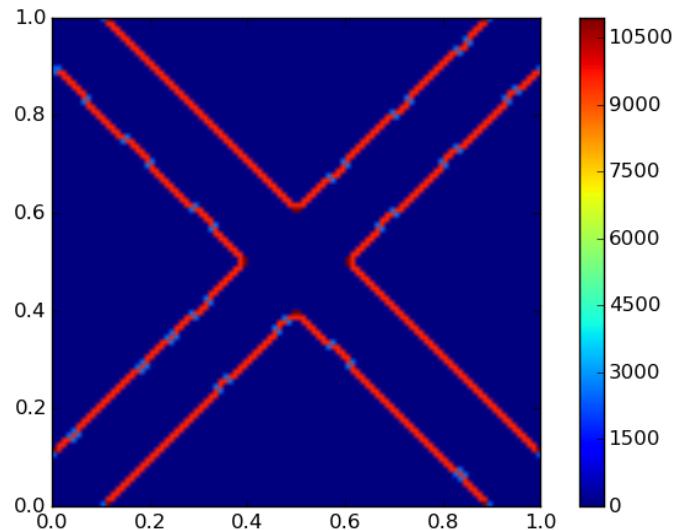


Figura 17: Cross IC initial boundary energy

3.4. Boundary conditions

For ensuring that there is no numerical errors in the borders of the experiment Periodic Conditions are applied.

4. Experiments

With the model and the initial conditions defined, some experiments have been run, to find out if the anisotropy of the grain boundary energy is important to configuration of the boundaries and the faceting of the crystals. Two different misorientations θ_0 will be compared. Each experiment has two steps, first an isotropic evolution, with the boundary energy as a fixed value; and then we turn on the effect of the anisotropy of the phases.

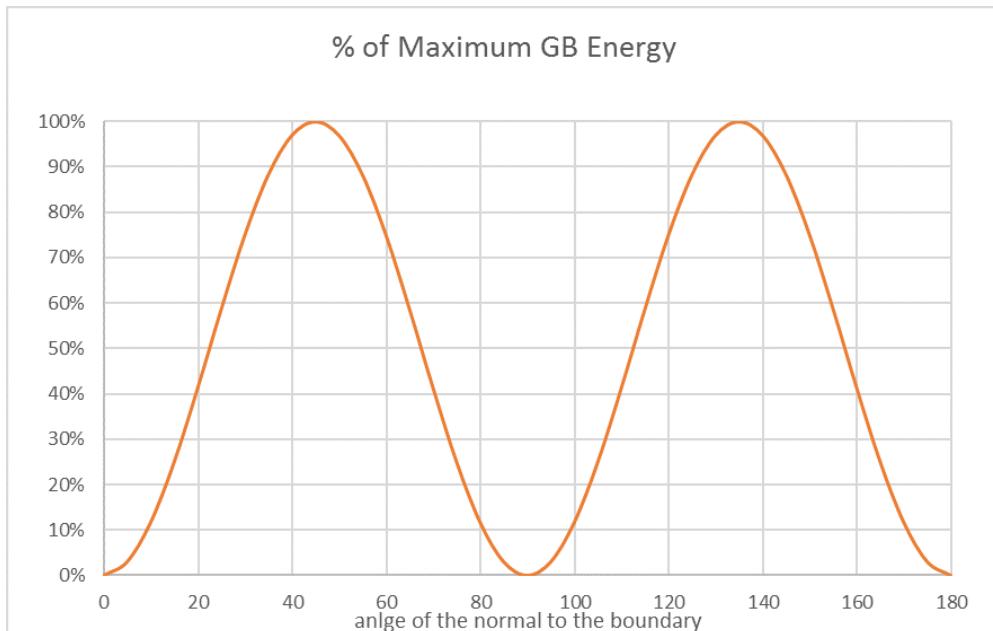


Figura 18: Energy function with $\theta_0 = 0^\circ$

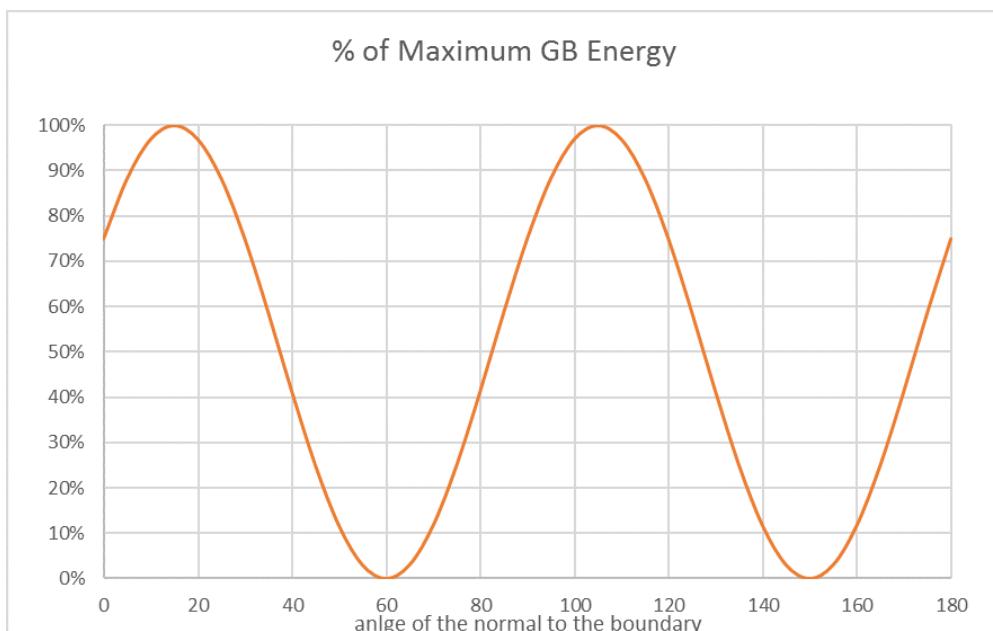


Figura 19: Energy function with $\theta_0 = 60^\circ$

4.1. Experiments with Random IC

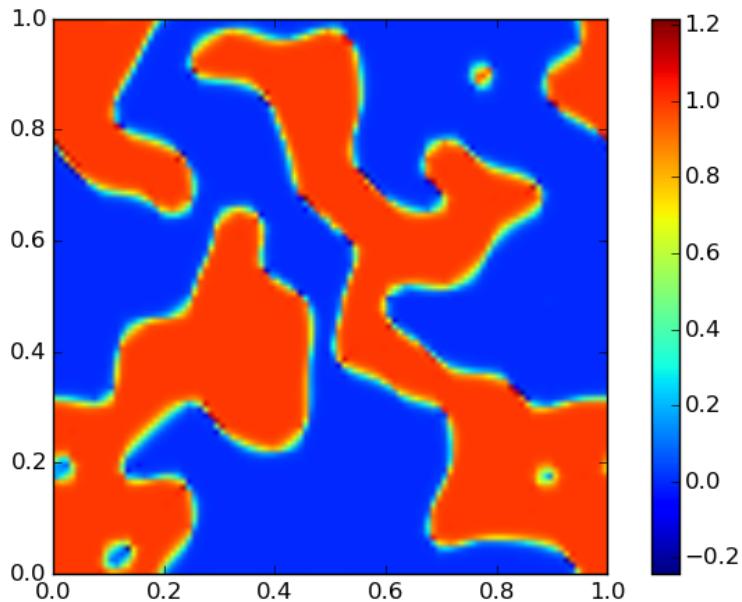


Figura 20: Random Isotropic evolution

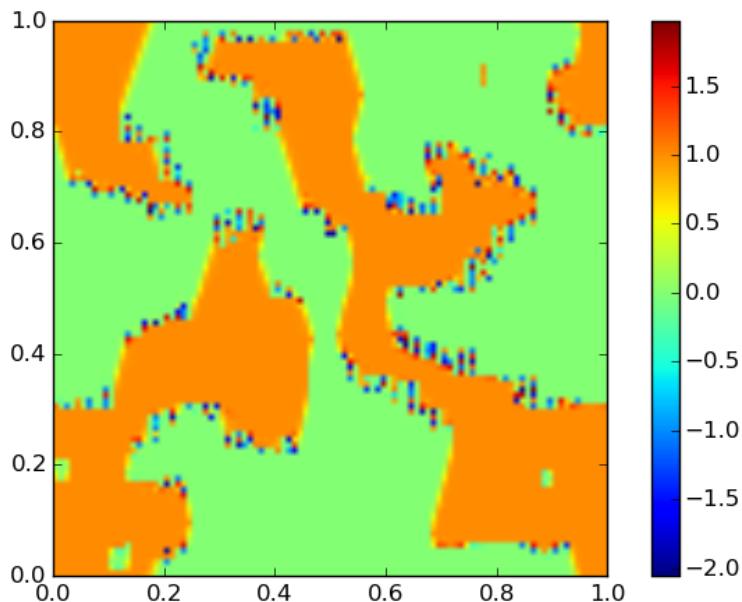


Figura 21: Random Anisotropic evolution, $\theta_0 = 0^\circ$

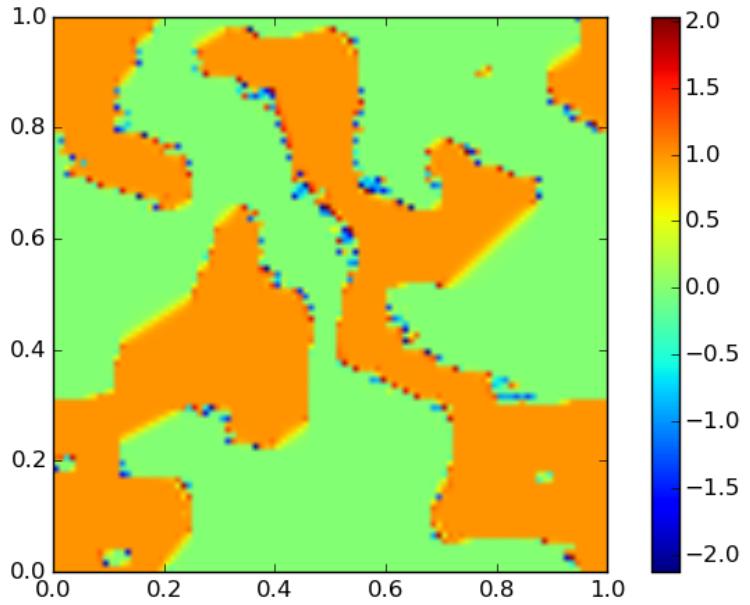


Figura 22: Random Anisotropic evolution, $\theta_0 = 60^\circ$

As is seen in the energy functions, the minimum energy directions for $\theta_0 = 0$ are the 0 and the 90° directions, and with $\theta_0 = 60^\circ$ the minimum energy directions are 60° and 130° , as we can see on the Fig.21 and on Fig. 22, the difference between them is the orientation of the boundaries, each experiment tent to the have the less energy configurations.

4.2. Experiments with straight bar

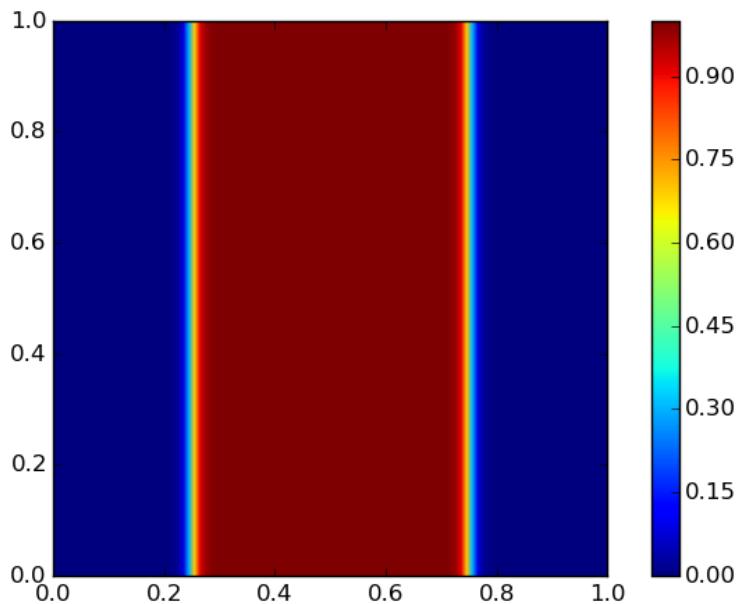


Figura 23: Bar Isotropic evolution

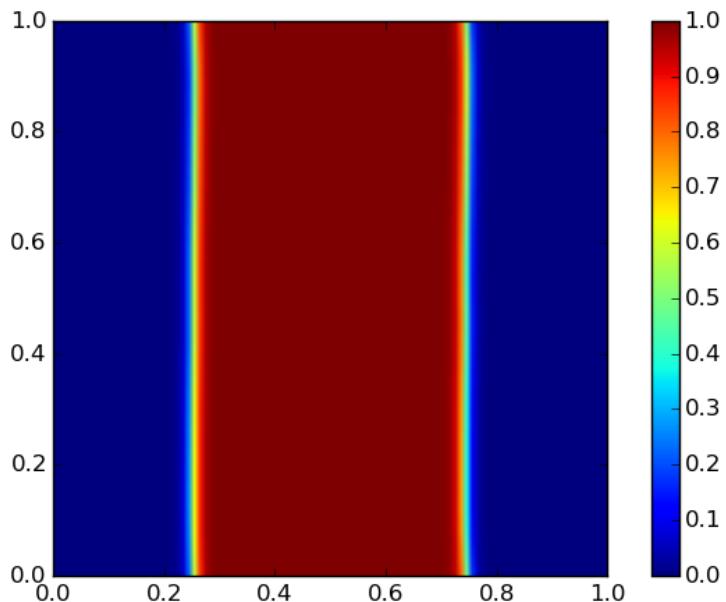


Figura 24: Bar Anisotropic evolution, $\theta_0 = 60^\circ$

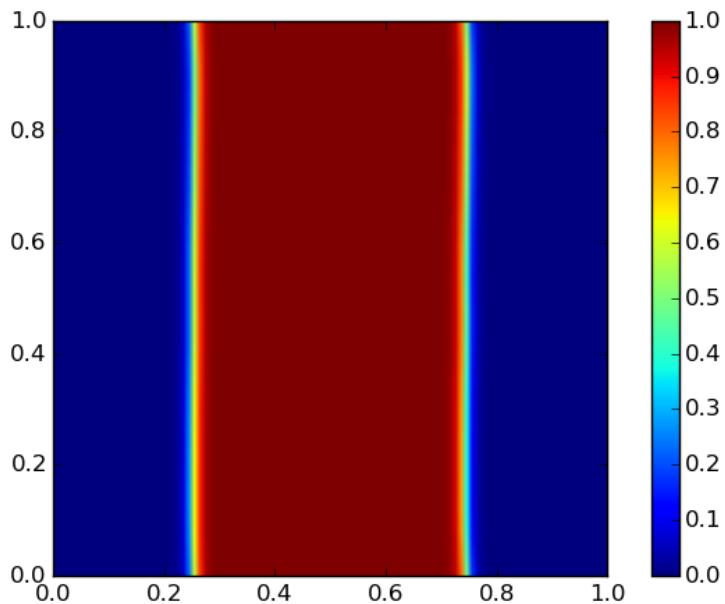


Figura 25: Bar Anisotropic evolution, $\theta_0 = 60^\circ$

This experiment shows that it is hardly impossible for a very large straight line(in this case infinite due to the periodic Boundary Conditions) to change, despite that the free energy on the boundary can be very high the boundary is so stable that the boundary won't change.

4.3. Experiments in a bar with a perturbation as IC

[TODO Add discussion and reference to figures.]

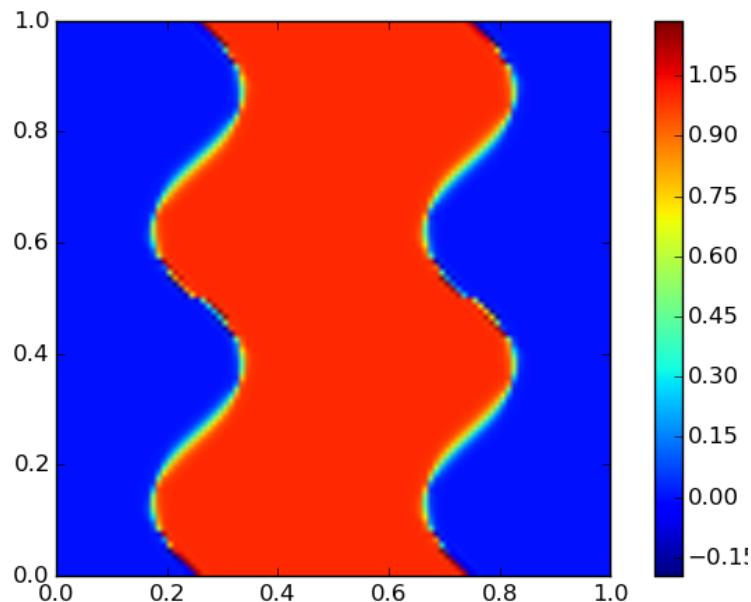


Figura 26: Bar with perturbations Isotropic evolution

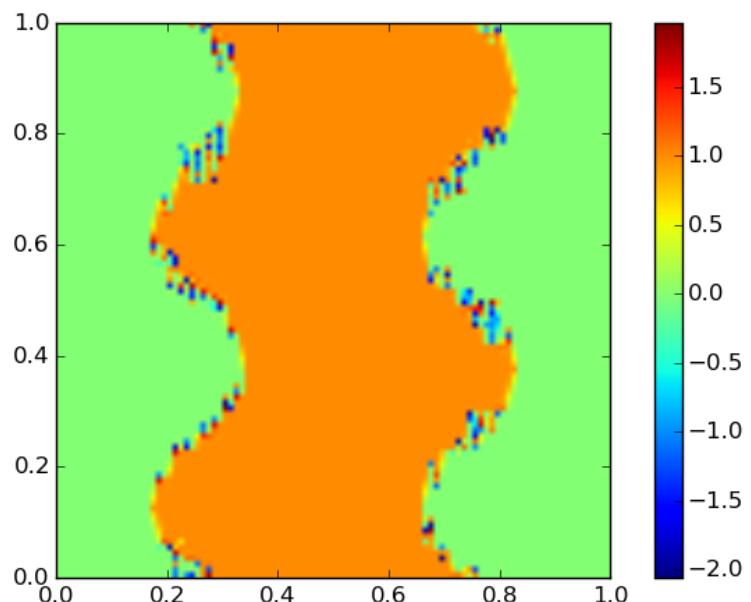


Figura 27: Bar with perturbations Anisotropic evolution, $\theta_0 = 0^\circ$

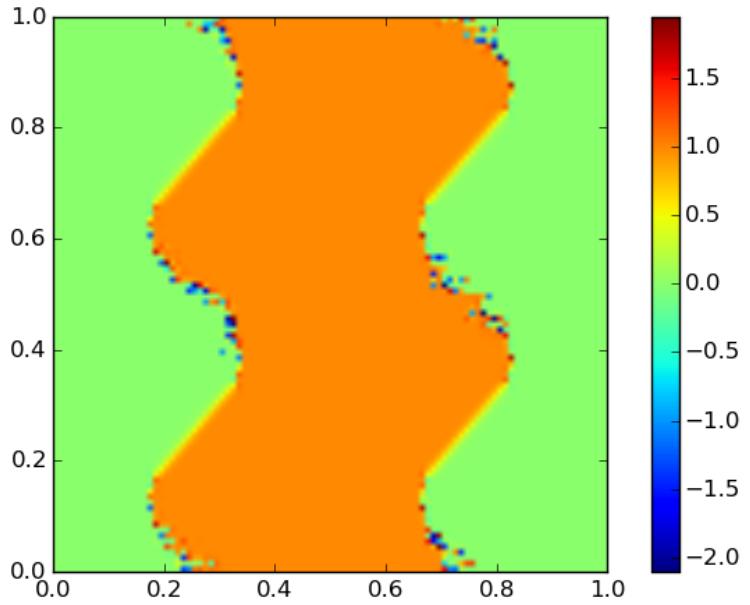


Figura 28: Bar with perturbations Anisotropic evolution, $\theta_0 = 60^\circ$

This experiment shows that if the grain boundaries are too sharp, the direction of the boundaries differ a lot from the minimum energy directions or the boundary can not be moved as a whole (or in big enough sections); the boundary won't tilt, instead of that it creates a faceted pattern with local equilibrium, but a higher resolution merge or a higher order integration may change this unstable facets.

4.4. Experiments with a Circle as IC

[TODO Add discussion and reference to figure.]

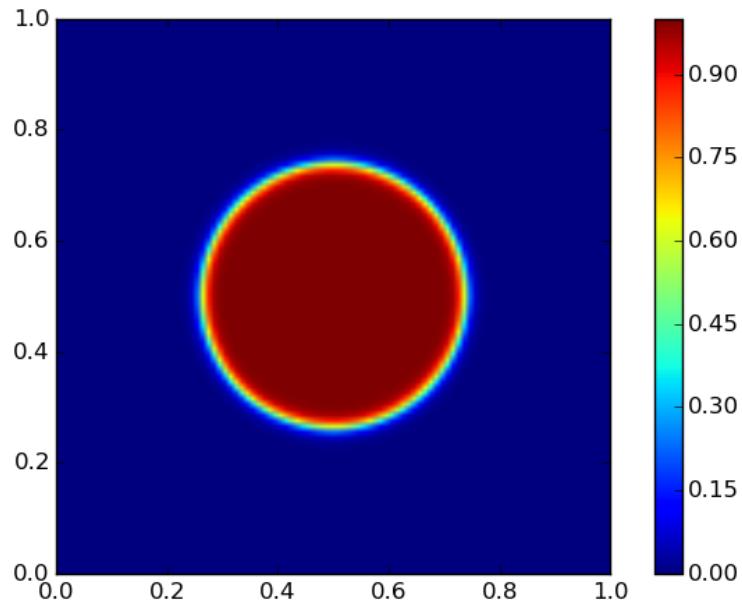


Figura 29: Circle Isotropic evolution

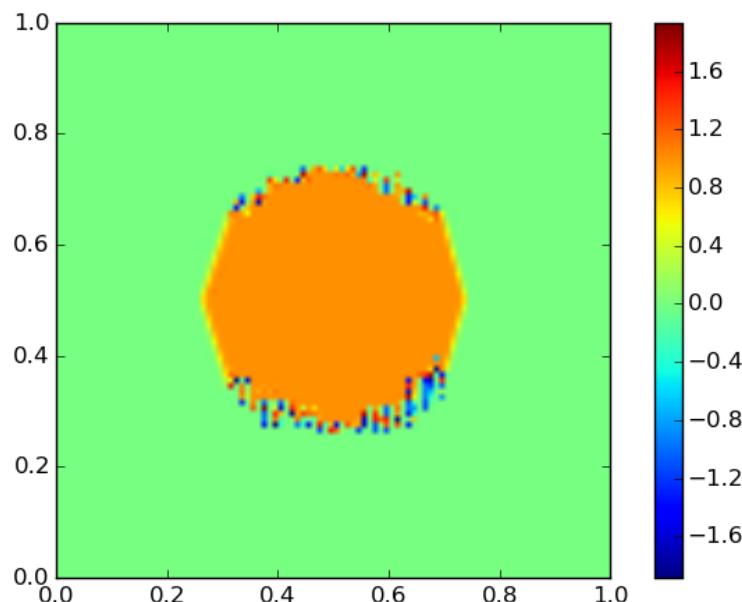


Figura 30: Circle Anisotropic evolution, $\theta_0 = 0^\circ$

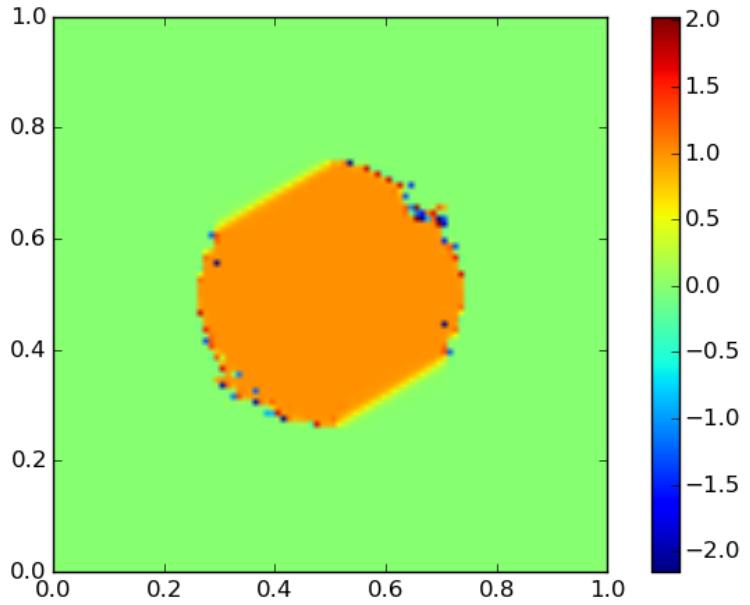


Figura 31: Circle Anisotropic evolution, $\theta_0 = 60^\circ$

With this experiment we see that the initial circle tends to change into an square oriented to the lower energy directions, it does not convert perfectly to an square because once a side is create it is difficult to the perpendicular ones to develop because the corner make it difficult to change.

4.5. Experiments with a Square as IC

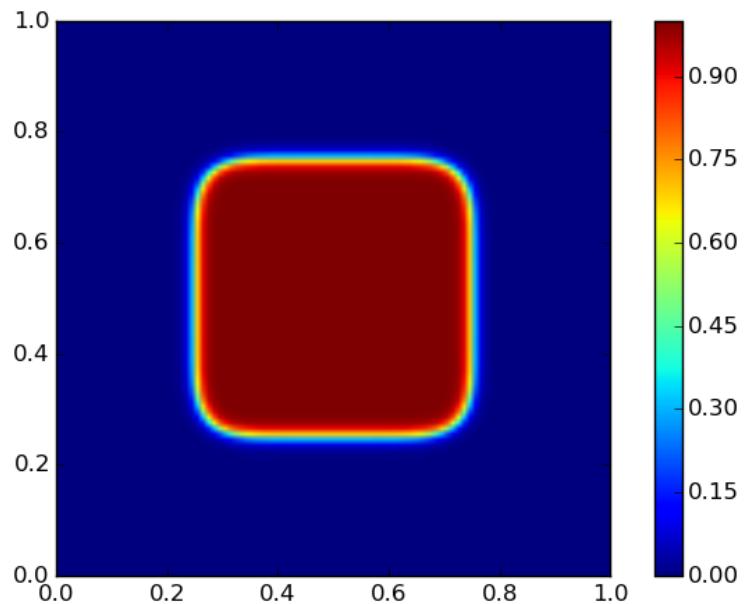


Figura 32: Square Isotropic evolution

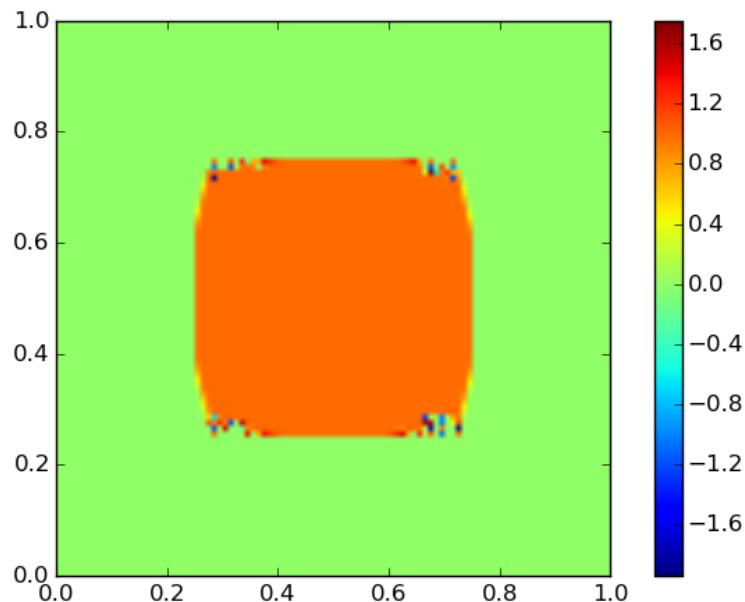


Figura 33: Square Anisotropic evolution, $\theta_0 = 0^\circ$

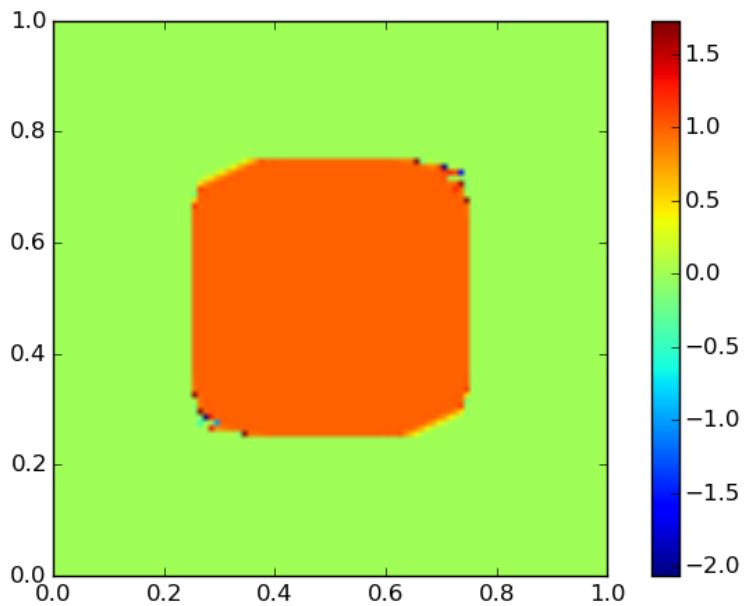


Figura 34: Square Anisotropic evolution, $\theta_0 = 60^\circ$

In the $\theta_0 = 0$ experiment the square converges so fast, only with minor problems in the corners, but in the $\theta_0 = 60^\circ$ the change of the boundary is not easy to take place due to the stability of the straight lines

4.6. Experiments with a Greek Cross as IC

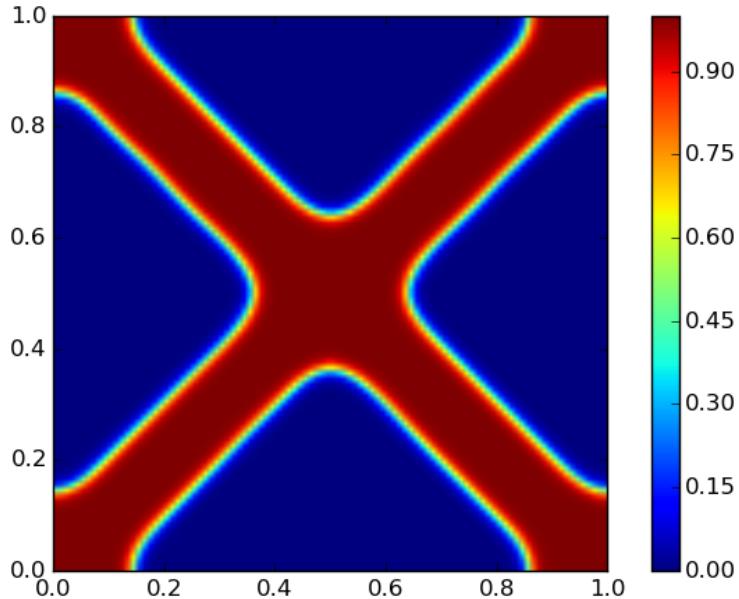


Figura 35: Cross Isotropic evolution

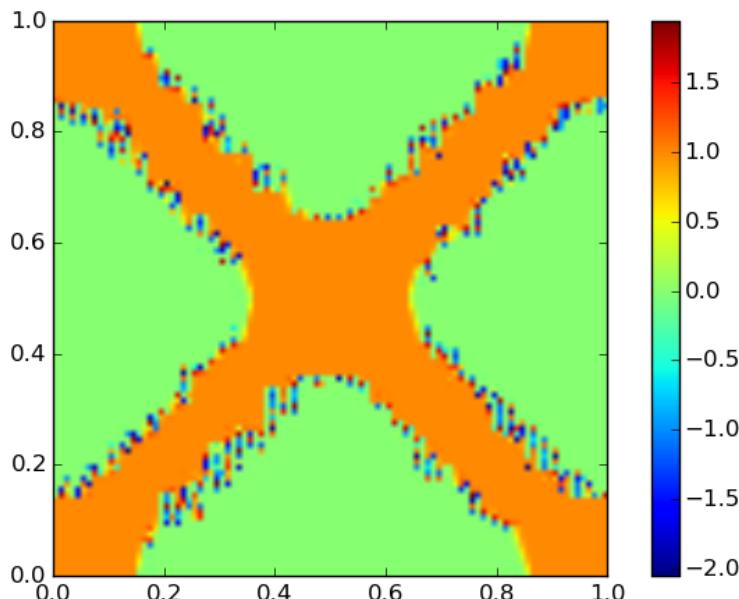


Figura 36: Cross Anisotropic evolution, $\theta_0 = 0^\circ$

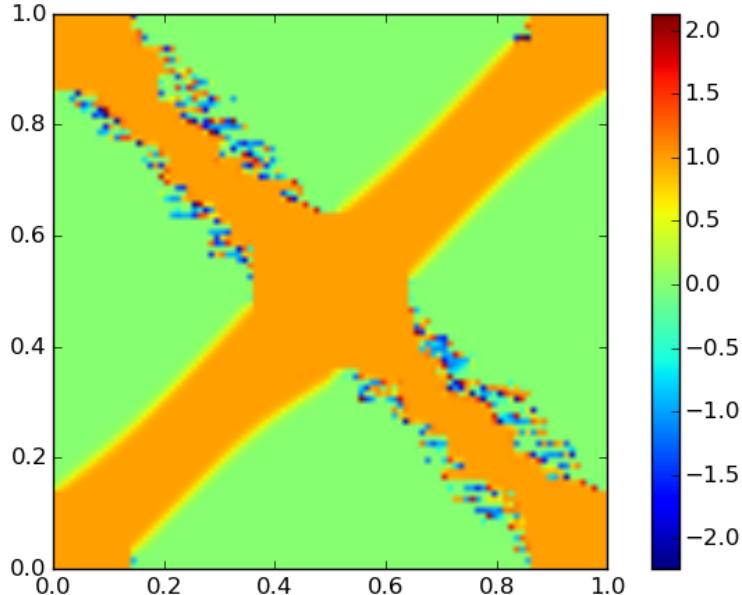


Figura 37: Cross Anisotropic evolution, $\theta_0 = 60^\circ$

This experiment shows perfectly how the different misorientations try unsuccessfully to change the boundary, but in different directions and with less impact when the orientation of the boundaries differ more from the less energy one.

5. Conclusions

The Phase field model created shows the influence of the anisotropy, making the boundaries to try (if possible) to move to less energy orientations. Instead this positive results the model needs more development.

The model shows that large straight boundaries are very stable, and that despite that they may have high energy orientations but they are still very stable, and it's hard for them to migrate to new configurations. Also, the energy at the corners is not well defined by the created model, because of that their energy needs more development to be more realistic.

Furthermore, taking in account higher order parameters, and computing with a higher resolution will drive to better solutions; and probably solve some of the problems in the corners.

6. Future work

Because I get accepted to the Balsells Fellowship program this work is not going to end here, the next two years at the same time that studying the Master in Mechanical Engineering I'll be working with the Multiscale Materials Group as part of the Fellowship Program.

The next fields to work to improve the model are: The energy of the corners (see if it is solved with higher order terms and higher resolution or not), implement Dr. Runnels Grain Boundary energy work to have realistic magnitudes for energy, transform the code to make it easier for parallel running in the UCCS supercomputers

7. Acknowledgements

Support for this work was provided by the Balsells Mobility Program, a UCCS startup fund, and the UCCS Committee for Research and Creative Works program.

8. References

- [1] B. Runnels, I. J. Beyerlein, S. Conti, and M. Ortiz, "A relaxation method for the energy and morphology of grain boundaries and interfaces," *Journal of the Mechanics and Physics of Solids*, vol. 94, pp. 388 – 408, 2016.
- [2] N. Moelans, B. Blanpain, and P. Wollants, "Quantitative analysis of grain boundary properties in a generalized phase field model for grain growth in anisotropic systems," *Phys. Rev. B*, vol. 78, p. 024113, Jul 2008.
- [3] C. Smith, "Grains, phases and interfaces: An interpretation of microstructure," 1948.
- [4] A. P. Sutton and R. W. Balluffi, "Interfaces in crystalline materials," 1995.

Appendices

Phase Field Model for Grain Boundary Anisotropy

```
// test
// Include standard libraries
#include <iostream>
#include <fstream>
#include <vector>
#include <stdlib.h>
#include <math.h>

// Include the Eigen library
#include "Eigen/Core"

// Include tclap
#include "tclap/CmdLine.h"
#include "tclap/IgnoreArg.h"

// Include the Reader library
#include "Reader.h"

// Include our own custom file , Field.h
#include "PF/Field.h"
#include "PF/IC/IC.h"
#include "PF/IC/Random.h"
#include "PF/IC/Square.h"
#include "PF/IC/Circle.h"
#include "PF/IC/Bar.h"
#include "PF/IC/Cross.h"

#define PI 3.14159265359

// This is the program's entry point
//

int main(int argc, char ** argv)
{
    //
    // COMMAND LINE PARSING
    //
    TCLAP::CmdLine cmd("PFGA");
    TCLAP::UnlabeledValueArg<std::string> argFileName("name", "Path to input file", false,
    "inputfile", cmd);
    TCLAP::IgnoreArg testIgnoreArg("D","User defined variables","","",cmd);
    cmd.parse(argc, argv);
    std::string fileName = argFileName.getValue();
```

```

// Create input file reader
Reader::Reader *reader;
if (fileName == "")
    reader = new Reader::Reader(argc , argv , "$" , "#" , "...");
else
    reader = new Reader::Reader(fileName , argc , argv , "$" , "#" , "...");

bool anisotropy = reader->Read<bool>(" anisotropy" , true);

// FILE I/O VARIABLES
//
// The following variable defines the location of the output files
// and the file names.
//
std::string filename = reader->Read<std::string>("output" );
//
// INITIAL CONDITIONS
//

PF::IC::IC *ic;
std::string initialCondition = reader->Read<std::string>(" IC" , " Type" , " random" );
if (initialCondition=="random")
    ic = new PF::IC::Random(reader->Read<int>(" IC" , " Seed" , 0));
else if (initialCondition=="circle")
    ic = new PF::IC::Circle(reader->Read<double>(" IC" , " x0" , 0.5) ,
                           reader->Read<double>(" IC" , " y0" , 0.5) ,
                           reader->Read<double>(" IC" , " r" , 0.25));
else if (initialCondition=="square")
    ic = new PF::IC::Square(reader->Read<double>(" IC" , " x0" , 0.5) ,
                           reader->Read<double>(" IC" , " y0" , 0.5) ,
                           reader->Read<double>(" IC" , " width" , 0.5) ,
                           reader->Read<double>(" IC" , " height" , 0.5));
else if (initialCondition=="bar")
    ic = new PF::IC::Bar(reader->Read<double>(" IC" , " x0" , 0.5) ,
                         reader->Read<double>(" IC" , " y0" , 0.5) ,
                         reader->Read<double>(" IC" , " width" , 0.5) ,
                         reader->Read<double>(" IC" , " height" , 0.5));
else if (initialCondition=="cross")
    ic = new PF::IC::Cross(reader->Read<double>(" IC" , " x0" , 0.5) ,
                           reader->Read<double>(" IC" , " y0" , 0.5) ,
                           reader->Read<double>(" IC" , " width" , 0.5) ,
                           reader->Read<double>(" IC" , " height" , 0.5));
else
{
    std::cout << "Unkonwn initial condition" << std::endl;
    exit(-1);
}

// ORDER PARAMETERS
//

```

```

// Here we define two order parameters, eta1 and eta2, given the x and y dimension
//(1.0,1.0)
// and number of points in each dimension (100, 100).
// We also define the type of initial condition to use.
// We get eta2 = 1-eta1.
//
double
lenX = reader->Read<double>("lenX",1.0),
lenY = reader->Read<double>("lenY",1.0);
double
nX = reader->Read<double>("nX",101),
nY = reader->Read<double>("nY",101);

PF:: Field<double> eta1(lenX,lenY,nX,nY,*ic);
PF:: Field<double> eta2 = eta1*(-1.) + 1.;

//
// PARAMETERS
//

// Isotropic parameters
double
sigmagb = reader->Read<double>("sigmagb",0.708),
lgb = reader->Read<double>("lgb",0.1),
mgb = reader->Read<double>("mgb",1.0),
f0saddle= reader->Read<double>("f0saddle",0.01),
gamma = reader->Read<double>("gamma",1.5);

// Derived parameters
double
kappa = .75*sigmagb*lgb ,
L=(4.0/3.0)*mgb/lgb ,
mu = .75*(1./f0saddle)*(sigmagb/lgb );

// GBA

double
sigma0 = reader->Read<double>("sigma0",0.708),
sigma1 = reader->Read<double>("sigma1",0.7),
theta = reader->Read<double>("theta",0.5*PI);

//
// EXPLICIT TIME INTEGRATION OF GINZBURG-LANDAU
//
double
dt = reader->Read<double>("dt",0.00001);
int
numSteps = reader->Read<int>("NumSteps",2500);

for (int i = 0; i<numSteps; i++)
{

```

```

PF:: Field<Eigen::Vector2d> GradEta1 = Grad(eta1);
PF:: Field<Eigen::Vector2d> GradEta2 = Grad(eta2);

PF:: Field<double> sigmagb1 = (PF:: sin(PF:: atan2(GradEta1)*2+theta)*PF::
sin(PF:: atan2(GradEta1)*2+theta)*sigma1 + sigma0 );
PF:: Field<double> sigmagb2 = (PF:: sin(PF:: atan2(GradEta2)*2+theta)*PF::
sin(PF:: atan2(GradEta2)*2+theta)*sigma1 + sigma0 );

PF:: Field<double> kappa1 = sigmagb1*0.75*lgb;
PF:: Field<double> kappa2 = sigmagb2*0.75*lgb;

if (i<200 || anisotropy==false)
{
    eta1 ==
        (((eta1*eta1*eta1) - eta1 + (eta1*eta2*eta2*2.0*gamma))*mu -
Laplacian(eta1)*kappa)*L*dt;

    eta2 ==
        (((eta2*eta2*eta2) - eta2 + (eta2*eta1*eta1*2.0*gamma))*mu -
Laplacian(eta2)*kappa)*L*dt;
}
else
{
    PF:: Field<double> G11=Grad1(eta1);
    PF:: Field<double> G12= Grad2(eta1);
    PF:: Field<double> G22= Grad2(eta2);
    PF:: Field<double> G21=Grad1(eta2);

    PF:: Field<double> nG1= Gnorm(G11,G12);
    PF:: Field<double> nG2= Gnorm(G21,G22);

    PF:: Field<double> dk11= (PF:: sin(PF:: atan2(GradEta1)*2+theta)*PF::
cos(PF:: atan2(GradEta1)*2+theta ))*0.75*lgb*4;
    // PF:: Field<double> dk12= (PF:: cos(PF:: atan2(GradEta1)*4+theta *2))*
0.75*8*lgb;
    PF:: Field<double> dk21= (PF:: sin(PF:: atan2(GradEta2)*2+theta )*PF::
cos(PF:: atan2(GradEta2)*2+theta ))*0.75*lgb*4;
    // PF:: Field<double> dk22= (PF:: cos(PF:: atan2(GradEta2)*4+theta *2))*
0.75*8*lgb;
    // PF:: Field<double> det21x = Derivative2x(eta1);
    // PF:: Field<double> det22x = Derivative2x(eta2);
    // PF:: Field<double> det21y = Derivative2y(eta1);
    // PF:: Field<double> det22y = Derivative2y(eta2);
    // PF:: Field<double> Cr1 = PF:: CrossProduct(eta1,eta2);
    // PF:: Field<double> Cr2 = PF:: CrossProduct(eta2,eta1);

    eta1 ==
        (((eta1*eta1*eta1) - eta1 + (eta1*eta2*eta2*2.0*gamma))*mu + kappa1-
G11*G12*dk11*Laplacian(eta1)*nG1 +
        //(((eta1*eta1*eta1) - eta1 + (eta1*eta2*eta2*2.0*gamma))*mu + dk12*Cr1*nG1 +
G11*G12*dk11*Laplacian(eta1)*nG1)*L*dt;
}

```

```

        //dk12*nG1*( eta2*eta2*(det21x)+eta1*eta1*(det21y))
        eta2 ==
            (((eta2*eta2*eta2) - eta2 + (eta2*eta1*eta1*2.0*gamma))*mu + kappa2-
G21*G22*dk21*Laplacian(eta2)*nG2)*L*dt;
        //    (((eta2*eta2*eta2) - eta2 + (eta2*eta1*eta1*2.0*gamma))*mu + dk22*Cr2*nG2*
G21*G22*dk21*Laplacian(eta2)*nG2)*L*dt;
        //    dk22*nG2*( eta2*eta2*(det22y)+eta1*eta1*(det22x))

    }

    //
    // DUMP OUTPUT
    //
    // The code in this block dumps the output to a file.
    // The conditional (i%10 == 0) only evaluates to true when
    // i is evenly divisible by 10.
    //

    if (i%10 == 0)
    {
        PF:: Field<double> boundaries = sigmagb1*Dotproduct(GradEta1,GradEta1);

        std::vector<PF:: Field<double> *> outputs;
        outputs.push_back(&eta1);
        outputs.push_back(&eta2);
        outputs.push_back(&boundaries);
        //outputs.push_back(&sigmagb2);

        char file[30];
        sprintf(file,"%s%05i.dat",filename.c_str(),i);
        std::ofstream out;
        out.open(file);
        //eta1.print(out);
        PF:: Print(out,outputs);
        out.close();
        std::cout << i << std::endl;
    }
}

```

Field with operators

```
#ifndef IC_FIELD_H
#define IC_FIELD_H

#include <math.h>
#include "PF/IC/IC.h"

// Namespaces just help keep things organized
namespace PF
{

// Don't worry about this thing just now
enum BC
{
    PERIODIC
};

// 
// Field Class
//
// This is a *templated* C++ class — in this definition , F could be a double ,
// Vector2d , etc .
// This is the type used to work with order parameters .
//
template<class F>
class Field
{
public:

    //
    // CONSTRUCTORS
    //

    Field<F>(double _lenX , double _lenY , int _nX , int _nY) :
        lenX(_lenX) , lenY(_lenY) ,
        nX(_nX) , nY(_nY) , f(nX*nY)
    {}

    Field<F>(double _lenX , double _lenY , int _nX , int _nY , IC::IC &ic) :
        lenX(_lenX) , lenY(_lenY) ,
        nX(_nX) , nY(_nY) , f(nX*nY)
    {
        double dX = lenX/(nX-1) , dY = lenY/(nY-1);
        for (int i=0; i<nX; i++)
            for (int j=0; j<nX; j++)
            {
                double x = dX*i , y=dY*j ;
                (*this)(i,j) = ic(x,y);
            }
    }
}
```

```

Field<F>(const Field<F> &a):
    lenX(a.lenX), lenY(a.lenY), nX(a.nX), nY(a.nY), f(a.f)
{ }

// 
// OVERLOADED OPERATORS
//
// Here is where we define operators on Field objects.
// You don't need to worry about them too much.
//
void
operator = (Field<F> b)
{
    for (int i=0; i<nX*nY; i++) f[i] = b.f[i];
}
void
operator += (Field<F> b)
{
    for (int i=0; i<nX*nY; i++) f[i] += b.f[i];
}
Field<F>
operator + (Field<F> b)
{
    Field a(*this);
    a += b;
    return a;
}
void
operator += (double alpha)
{
    for (int i=0; i<nX*nY; i++) f[i] += alpha;
}
Field<F>
operator + (double alpha)
{
    Field a(*this);
    a += alpha;
    return a;
}
void
operator -= (Field<F> b)
{
    for (int i=0; i<nX*nY; i++) f[i] -= b.f[i];
}
Field<F>
operator - (Field<F> b)
{
    Field a(*this);
    a -= b;
    return a;
}
void

```

```

operator == (double alpha)
{
    for (int i=0; i<nX*nY; i++) f[i] == alpha;
}
Field<F>
operator - (double alpha)
{
    Field a(*this);
    a -= alpha;
    return a;
}

void
operator *= (Field<F> b)
{
    for (int i=0; i<nX*nY; i++) f[i] *= b.f[i];
}

template<class G>
void
operator *= (Field<G> b)
{
    for (int i=0; i<nX*nY; i++) f[i] *= b.f[i];
}

template<class G>
Field<F>
operator * (Field<G> b)
{
    Field a(*this);
    a *= b;
    return a;
}
void
operator *= (double alpha)
{
    for (int i=0; i<nX*nY; i++) f[i] *= alpha;
}
Field<F>
operator * (double alpha)
{
    Field a(*this);
    a *= alpha;
    return a;
}
void
operator /= (double alpha)
{
    for (int i=0; i<nX*nY; i++) f[i] /= alpha;
}
Field<F>
operator / (double alpha)

```

```

{
    Field a(*this);
    a /= alpha;
    return a;
}
void
print(std::ostream &out)
{
    double dX = lenX/(nX-1), dY = lenY/(nY-1);

    for (int i=0;i<nX; i++)
        for (int j=0;j<nY; j++)
            out << dX*i << " " << dY*j << " " << (*this)(i,j) << std::endl;
}
F &
operator () (int i, int j)
{
    return f[(i+nX)%nX + nY*((j+nY)%nY)];
}

// Fields associated with the object
// double lenX, lenY;
// int nX, nY;
std::vector<F> f;
};

// SCALAR PRODUCT
// Field<double>

Dotproduct ( Field<Eigen::Vector2d> a, Field<Eigen::Vector2d> b)
{
    Field<double> Dotproductab(a.lenX,a.lenY,a.nX,a.nY);
    for (int i = 0; i < a.nX; i++)
        for (int j = 0; j < a.nY; j++)
    {
        Dotproductab(i,j)=a(i,j).dot(b(i,j));
    }
    return Dotproductab;
}

// Fake norm
// Field<double>

Gnorm ( Field<double> a, Field<double> b)
{
    Field<double> gnormab(a.lenX,a.lenY,a.nX,a.nY);

```

```

    for (int i = 0; i < a.nX; i++)
        for (int j = 0; j < a.nY; j++)
            if ((a(i,j)*a(i,j)+b(i,j)*b(i,j))>0.001)
            {
                gnormab(i,j) = 1/(2.*a(i,j)*a(i,j)+2.*b(i,j)*b(i,j));
            }
            else
            {
                gnormab(i,j)=0 ;
            }
    return gnormab;
}

// 
//  SQUARE NORM
//
Field<double>

Sqnorm ( Field<Eigen::Vector2d> a)
{
    Field<double> Sqnorma(a.lenX,a.lenY,a.nX,a.nY);
    for (int i = 0; i < a.nX; i++)
        for (int j = 0; j < a.nY; j++)
        {
            Sqnorma(i,j)=a(i,j).dot(a(i,j));
        }
    return Sqnorma;
}

// 
//  GRADIENT FUNCTION
//
Field<Eigen::Vector2d>
Grad ( Field<double> eta, const BC bc = PERIODIC)
{
    double dX = eta.lenX/(eta.nX-1), dY = eta.lenY/(eta.nY-1);
    Field<Eigen::Vector2d> gradEta(eta.lenX,eta.lenY,eta.nX,eta.nY);
    if (bc == PERIODIC)
        for (int i = 0; i < eta.nX; i++)
            for (int j = 0; j < eta.nY; j++)
            {
                // Three point stencil (first order)
                // gradEta(i,j) <<
                // (eta(i+1,j)-eta(i-1,j))/(2.*dX),
                // (eta(i,j+1)-eta(i,j-1))/(2.*dY);
                // Five point stencil (second order)
                gradEta(i,j) <<
                (-eta(i+2,j) + 8.*eta(i+1,j) - 8.*eta(i-1,j) + eta(i-2,j))/(12.*dX) ,
                (-eta(i,j+2) + 8.*eta(i,j+1) - 8.*eta(i,j-1) + eta(i,j-2))/(12.*dY);
            }
}

```

```

    return gradEta;
}
// 
// GRAD2
// 
Field<double>
Grad2 (Field<double> eta, const BC bc = PERIODIC)
{
    double dY = eta.lenY/(eta.nY-1);
    Field<double> gradEta2(eta.lenX, eta.lenY, eta.nX, eta.nY);
    if (bc == PERIODIC)
        for (int i = 0; i < eta.nX; i++)
            for (int j = 0; j < eta.nY; j++)
            {
                // Three point stencil (first order)
                // gradEta(i,j) <<
                // (eta(i+1,j)-eta(i-1,j))/(2.*dX),
                // (eta(i,j+1)-eta(i,j-1))/(2.*dY);
                // Five point stencil (second order)
                gradEta2(i,j) =
                    (-eta(i,j+2) + 8.*eta(i,j+1) - 8.*eta(i,j-1) + eta(i,j-2))/(12.*dY);

            }
    return gradEta2;
}
// 
// GRAD1
// 
Field<double>
Grad1 (Field<double> eta, const BC bc = PERIODIC)
{
    double dX = eta.lenX/(eta.nX-1);
    Field<double> gradEta1(eta.lenX, eta.lenY, eta.nX, eta.nY);
    if (bc == PERIODIC)
        for (int i = 0; i < eta.nX; i++)
            for (int j = 0; j < eta.nY; j++)
            {
                // Three point stencil (first order)
                // gradEta(i,j) <<
                // (eta(i+1,j)-eta(i-1,j))/(2.*dX),
                // (eta(i,j+1)-eta(i,j-1))/(2.*dY);
                // Five point stencil (second order)
                gradEta1(i,j) =
                    (-eta(i+2,j) + 8.*eta(i+1,j) - 8.*eta(i-1,j) + eta(i-2,j))/(12.*dX)
;
            }
    return gradEta1;
}

```

```

//  

// 1ST DERIVATIVE  

//  

Field<double>  

Derivative1 ( Field<double> eta , const BC bc = PERIODIC)  

{  

    double dX = eta.lenX/(eta.nX-1), dY = eta.lenY/(eta.nY-1);  

    Field<double> dev1(eta.lenX, eta.lenY, eta.nX, eta.nY);  

    if (bc == PERIODIC)  

        for (int i = 0; i < eta.nX; i++)  

            for (int j = 0; j < eta.nY; j++)  

            {  

                dev1(i,j) =  

                    (-eta(i+2,j) + 8.*eta(i+1,j) - 8.*eta(i-1,j) + eta(i-2,j))/(12.*dX)  

                    + (-eta(i,j+2) + 8.*eta(i,j+1) - 8.*eta(i,j-1) + eta(i,j-2))/(12.*dY);  

            }  

    return dev1;  

}  

//  

// 2ND DERIVATIVE X  

//  

Field<double>  

Derivative2x ( Field<double> eta , const BC bc = PERIODIC)  

{  

    double dX = eta.lenX/(eta.nX-1);  

    Field<double> dev2x(eta.lenX, eta.lenY, eta.nX, eta.nY);  

    if (bc == PERIODIC)  

        for (int i = 0; i < eta.nX; i++)  

            for (int j = 0; j < eta.nY; j++)  

            {  

                dev2x(i,j) =  

                    (eta(i+1,j) + eta(i-1,j) - 2.*eta(i,j))/(dX*dX);  

            }  

    return dev2x;  

}  

// 2ND DERIVATIVE Y  

//  

Field<double>  

Derivative2y ( Field<double> eta , const BC bc = PERIODIC)  

{  

    double dY = eta.lenY/(eta.nY-1);  

    Field<double> dev2y(eta.lenX, eta.lenY, eta.nX, eta.nY);  

    if (bc == PERIODIC)  

        for (int i = 0; i < eta.nX; i++)  

            for (int j = 0; j < eta.nY; j++)  

            {

```

```

    dev2y(i,j) =
        (eta(i,j+1) + eta(i,j-1) - 2.*eta(i,j))/(dY*dY) ;
}

return dev2y;
}

// Cross product
//
Field<double>
CrossProduct (Field<double> eta, Field<double> eta2, const BC bc = PERIODIC)
{
    double dX = eta.lenX/(eta.nX-1), dY = eta.lenY/(eta.nY-1);
    Field<double> crossEta(eta.lenX, eta.lenY, eta.nX, eta.nY);
    if (bc == PERIODIC)
        for (int i = 0; i < eta.nX; i++)
            for (int j = 0; j < eta.nY; j++)
            {
                crossEta(i,j) =
                    eta2(i,j)*eta2(i,j)*(eta(i+1,j) + eta(i-1,j) - 2.*eta(i,j))/(dX*dX) +
                    eta(i,j)*eta(i,j)*(eta(i,j+1) + eta(i,j-1) - 2.*eta(i,j))/(dY*dY);
            }
    return crossEta;
}

// LAPLACIAN FUNCTION
//
Field<double>
Laplacian (Field<double> eta, const BC bc = PERIODIC)
{
    double dX = eta.lenX/(eta.nX-1), dY = eta.lenY/(eta.nY-1);
    Field<double> lapEta(eta.lenX, eta.lenY, eta.nX, eta.nY);
    if (bc == PERIODIC)
        for (int i = 0; i < eta.nX; i++)
            for (int j = 0; j < eta.nY; j++)
            {
                lapEta(i,j) =
                    (eta(i+1,j) + eta(i-1,j) - 2.*eta(i,j))/(dX*dX) +
                    (eta(i,j+1) + eta(i,j-1) - 2.*eta(i,j))/(dY*dY);
            }
    return lapEta;
}

// DIVERGENCE FUNCTION
//
Field<double>
Div(Field<Eigen::Vector2d> eta, const BC bc = PERIODIC)

```

```

{
    double dX = eta.lenX/(eta.nX-1), dY = eta.lenY/(eta.nY-1);
    Field<double> divEta(eta.lenX, eta.lenY, eta.nX, eta.nY);
    if (bc == PERIODIC)
        for (int i = 0; i < eta.nX; i++)
            for (int j = 0; j < eta.nY; j++)
            {
                // Five point stencil (second order)
                divEta(i,j) =
                    (-eta(i+2,j)(0) + 8.*eta(i+1,j)(0) -
                     8.*eta(i-1,j)(0) + eta(i-2,j)(0))/(12.*dX) +
                    (-eta(i,j+2)(1) + 8.*eta(i,j+1)(1) -
                     8.*eta(i,j-1)(1) + eta(i,j-2)(1))/(12.*dY);
            }
        return divEta;
    }

    //
    // SIN FUNCTION
    //
    Field<double>
    sin(Field<double> a)
    {
        Field<double> c(a);
        for (int i = 0; i < a.nX; i++)
            for (int j = 0; j < a.nY; j++)
            {
                // Five point stencil (second order)
                c(i,j) = std::sin(a(i,j));
            }
        return c;
    }

    //
    // COS FUNCTION
    //
    Field<double>
    cos(Field<double> a)
    {
        Field<double> c(a);
        for (int i = 0; i < a.nX; i++)
            for (int j = 0; j < a.nY; j++)
            {
                // Five point stencil (second order)
                c(i,j) = std::cos(a(i,j));
            }
        return c;
    }

    //
    // ATAN2 FUNCTION
    //

```

```

Field<double>
atan2( Field<Eigen::Vector2d> a)
{
    Field<double> c(a.lenX,a.lenY,a.nX,a.nY);
    for (int i = 0; i < a.nX; i++)
        for (int j = 0; j < a.nY; j++)
    {
        c(i,j) = std::atan2(a(i,j)(1),a(i,j)(0));
    }
    return c;
}

void
Print(std::ostream &out, std::vector<PF::Field<double> *> outputs)
{
    int num = outputs.size();

    int nX = outputs[0]->nX, nY = outputs[0]->nY;
    double lenX = outputs[0]->lenX, lenY = outputs[0]->lenY;
    double dX = lenX/(nX-1), dY = lenY/(nY-1);

    for (int i=0;i<nX;i++)
        for (int j=0;j<nY;j++)
    {
        out << dX*i << " " << dY*j ;
        for (int k=0; k<num; k++)
            out << " " << (*outputs[k])(i,j);
        out << std::endl;
    }
}
#endif

```

Initial Conditions

```

RANDOM:
#ifndef PF_IC_RANDOM_H
#define PF_IC_RANDOM_H

#include "IC.h"

namespace PF
{
namespace IC
{
//
```

```

// RANDOM IC
//

class Random : public IC
{
public:
    Random() { srand (time(NULL)); }
    Random(int seed) { srand (seed); }

    double operator()(double x, double y)
    {
        return (double)rand()/(double)RAND_MAX;
    }
};

}

#endif

```

BAR(with and without perturbations):

```

#ifndef PF_IC_BAR_H
#define PF_IC_BAR_H
#include <iostream>
#include <fstream>
#include <vector>
#include <stdlib.h>
#include <math.h>
#include "IC.h"
#define PI 3.14159265359
#define freq 0*PI
#define A 0.1
namespace PF
{
namespace IC
{
class Bar : public IC
{
public:
    Bar(double _x0, double _y0, double _width, double _height) :
        x0(_x0), y0(_y0), width(_width), height(_height) {};
    //    double freq =PI, A= 0.0

    double operator()(double x, double y)
    {
        if ( x >= x0 - 0.5*width-A*std::sin(freq*y) &&
            x <= x0 + 0.5*width-A*std::sin(freq*y) )

```

```

        return 1;
    else
        return 0;
}
double x0 ,y0 ,width ,height ;
};

}

}

#endif

CIRCLE:

#ifndef PF_IC_CIRCLE_H
#define PF_IC_CIRCLE_H

#include "IC.h"

namespace PF
{
namespace IC
{
// CIRCULAR INCLUSION IC
//

class Circle : public IC
{
public:
    Circle(double _x0 , double _y0 , double _radius) :
        x0(_x0),y0(_y0),radius(_radius) {}

    double operator()(double x, double y)
    {
        if ( (x-x0)*(x-x0) + (y-y0)*(y-y0) <= radius*radius )
            return 1;
        else
            return 0;
    }
    double x0 ,y0 ,radius ;
};

}

}

#endif

```

SQUARE:

```

#ifndef PF_IC_SQUARE_H
#define PF_IC_SQUARE_H

#include "IC.h"

namespace PF

```

```

{
namespace IC
{
class Square : public IC
{
public:
    Square(double _x0, double _y0, double _width, double _height) :
        x0(_x0), y0(_y0), width(_width), height(_height) {}

    double operator()(double x, double y)
    {
        if (x >= x0 - 0.5*width &&
            x <= x0 + 0.5*width &&
            y >= y0 - 0.5*height &&
            y <= y0 + 0.5*height)
            return 1;
        else
            return 0;
    }
    double x0, y0, width, height;
};
}
}
#endif

```

CROSS:

```

ifndef PF_IC_CROSS_H
#define PF_IC_CROSS_H
#include <iostream>
#include <fstream>
#include <vector>
#include <stdlib.h>
#include <math.h>
#include "IC.h"
#define PI 3.14159265359
#define freq PI
#define A 0.1
namespace PF
{
namespace IC
{
class Cross : public IC
{
public:
    Cross(double _x0, double _y0, double _width, double _height) :
        x0(_x0), y0(_y0), width(_width), height(_height) {}

    ;
    // double freq =PI, A= 0.1;

    double operator()(double x, double y)

```

```
{  
    if ( (x >= y-0.1 &&  
          x <= y+0.1)||  
        (x >= 0.9-y &&  
          x <= 1.1-y))  
        return 1;  
    else  
        return 0;  
}  
    double x0,y0,width,height;  
};  
}  
}  
}  
#endif
```