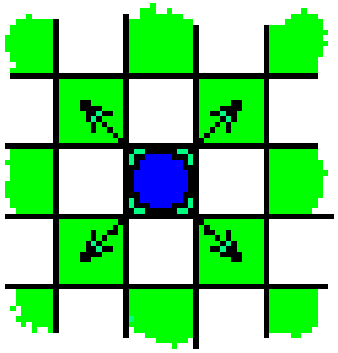
Representation of the membrane bulk

Thickness of the membrane : e
in pixels

Displacements of molecules :
Square diagonal lattice

Gas molecule : pixel



Jump lengh : λ
in pixels

λ= 1.414 pixels

Jump speed : ϕ
in cycles^{-1}

Host sites :
Characterized by affinity parameters

Concentration : C
Number of molecules per site

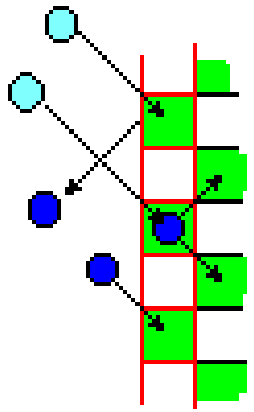
Diffusion coefficient : D
in square pixels per cycle

$D= \phi \lambda^2 / 4$ (bidimensional)

Description of membrane surface

Pressure : P
Number of incident molecules per host site of the surface per cycle

Surface of the membrane



Gas

Molecule is not absorbed

Membrane

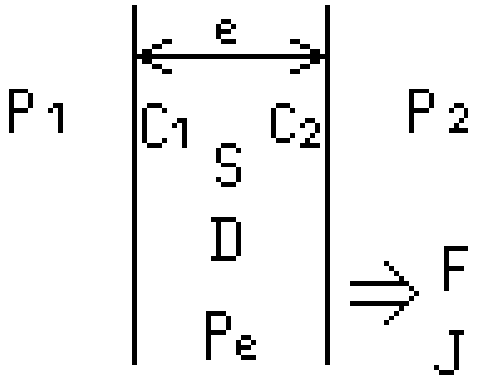
Molecule is absorbed

Solubility constant : S (probability of absorption)

Concentration : C = S * P

Calculations in permeation case

Gradient of concentration : $C_1 > C_2$ inside the membrane



P_1 : upstream pressure
P_2 : downstream pressure

$P_1 > P_2$

$C_1 = S\ P_1$

$C_2 = S\ P_2$

Flux : F
Number of outgoing molecules by the whole right side per cycle

Density of flux : J
Number of outgoing molecules per host border site on the right side

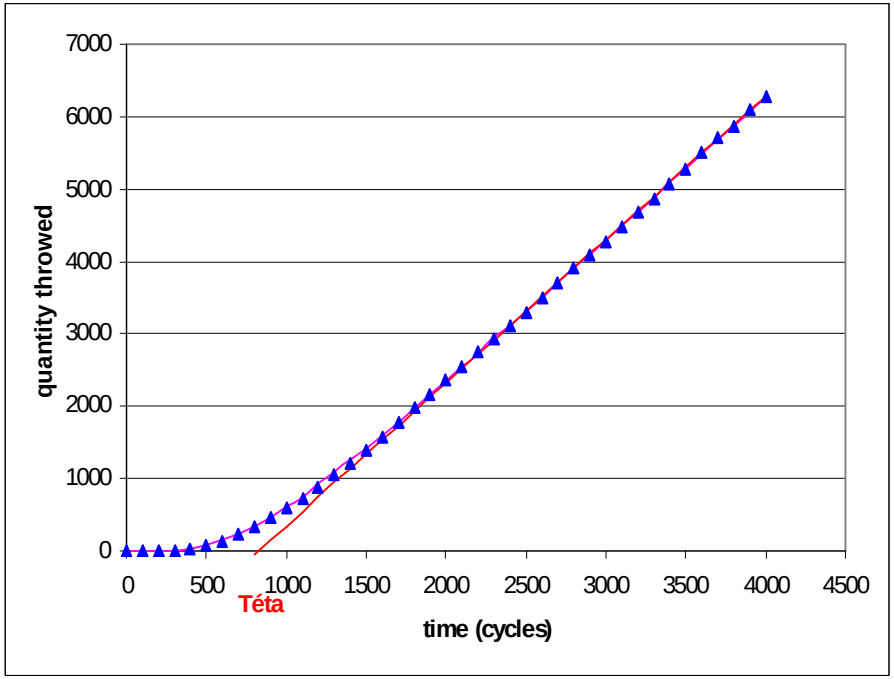
Permeability : Pe
$Pe = J\ (P_1 - P_2) / e$

Fick law : $J = D\ (C_1 - C_2) / e$

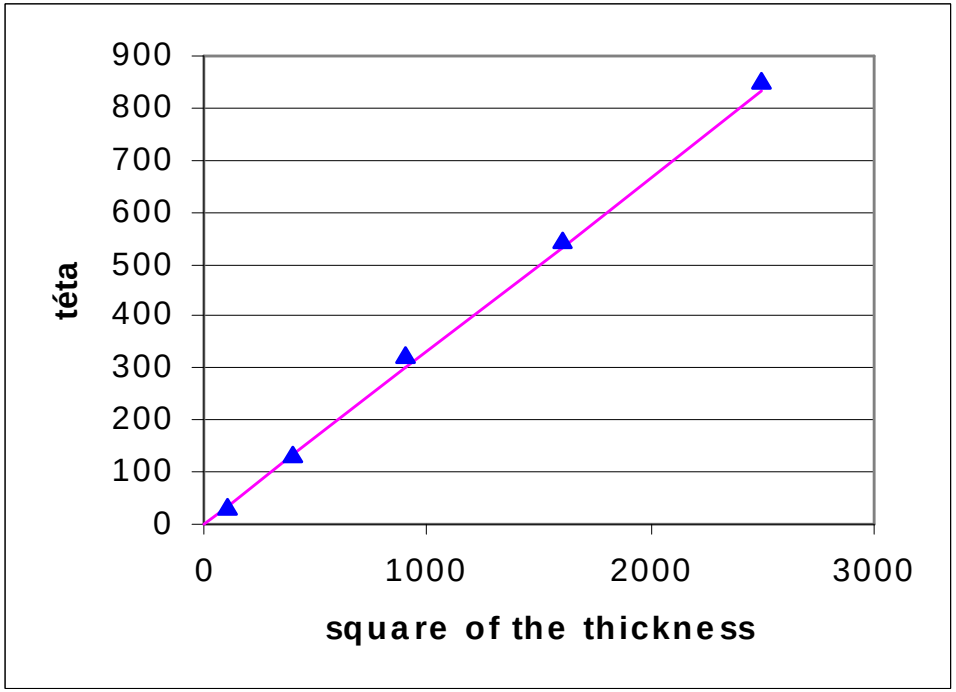
Fickian permeation : $Pe = D\ S$

Some results (permeation)

Permeation kinetics curve :
(Parameters : P1 = 0.4, P2 = 0,
S = 1, e = 50, Diff jump sp. = 1)



Linear increase of the time-lag (Teta, θ) with the square of the thickness :



Estimation of the diffusion coefficient :

$$D = E^2 / 6\,\theta$$ (either D is bidimensional)

Simulation of a drop of liquid

500 <= Nm <= 1000 molecules in the drop

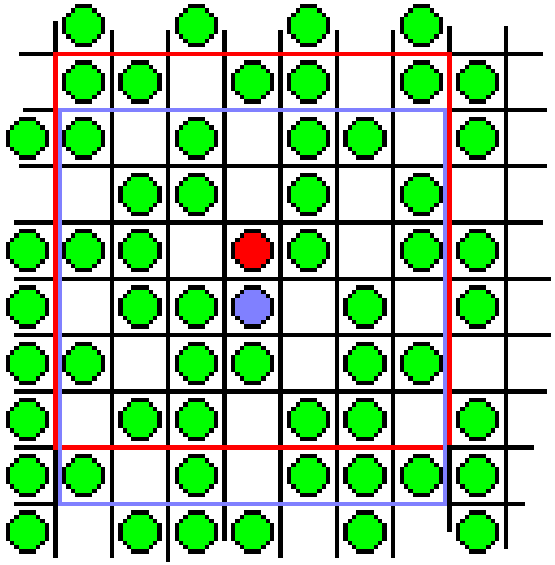
Exploration path for a molecule in test : 7 * 7 pixels

N : number of neighbouring molecules in the path

First position of the molecule

Next position of the molecule

Other molecules



Energetic contribution of a molecule : E

First position : Ni neighbours
Ei = Ni E

Next position : Nf neighbours
Ef = Nf E

Condition of displacment of a molecule :
Ef > Ei or Ef = Eo (mean energy)

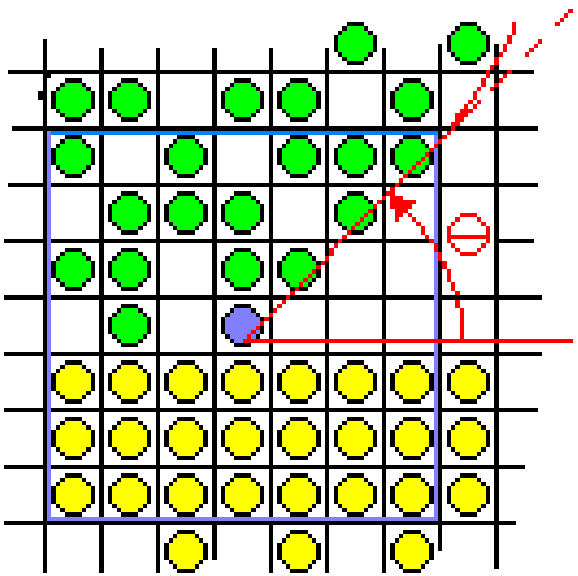
Some results in case of swelling application

- Energetic contribution of a molecule of the liquid (green): Em
- Energetic contribution of a molecule of the sustrate (yellow) : Es

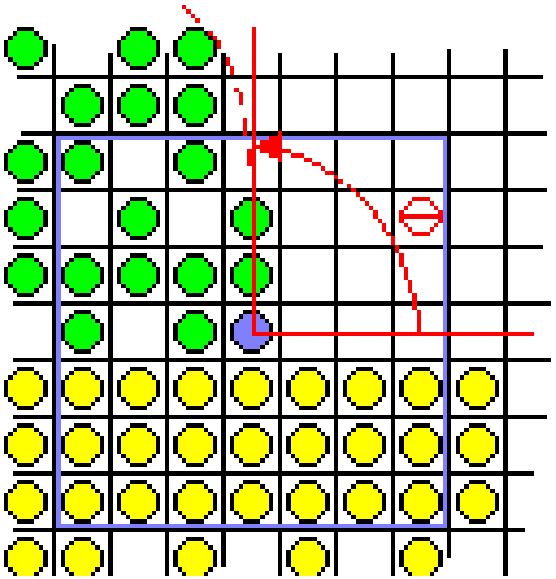
Es is varying in percentage of Em, Es <= Em

Measured parameter : front angle θ between the drop and the substrate :

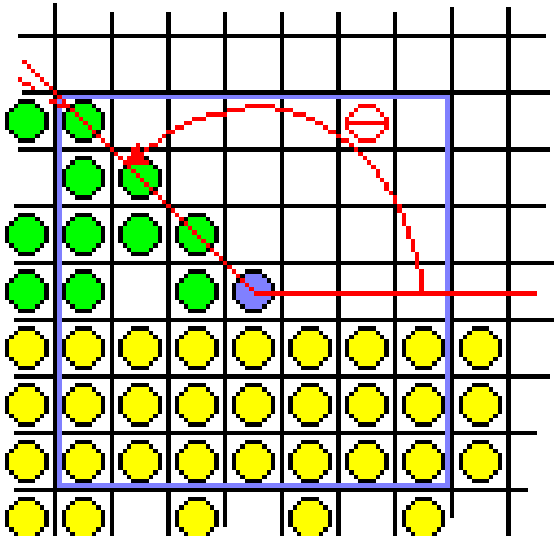
Es low :



Es medium :



Es high :

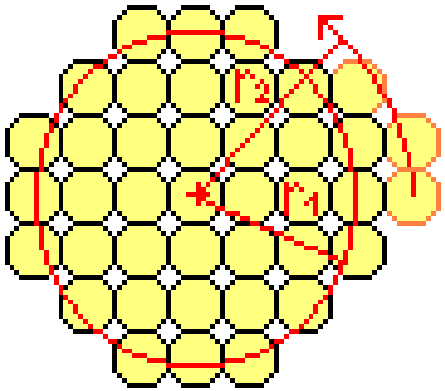


Cristallization of a polymer

Global approach simulation

The sperulite is growing around its central germ (repaired by the red cross)

Each sperulite is repaired by its particular color



New layer : increase the radius ($r_2 > r_1$) and coloring the pixels while turning around the spherulite

A pixel is colored only if it is not yet colored

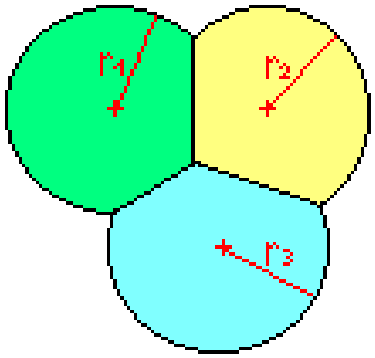
Entered parameters : **- number N of initial germs : Ni**
- germination speed : Vg = dN / dt
- growth speed : Vc = dr / dt

Registered parameter : **- cristallization rate T= number of colored pixels / total number of pixels in the frame**

Some results

Heterogen germination :

Ni > 0, Vg = 0

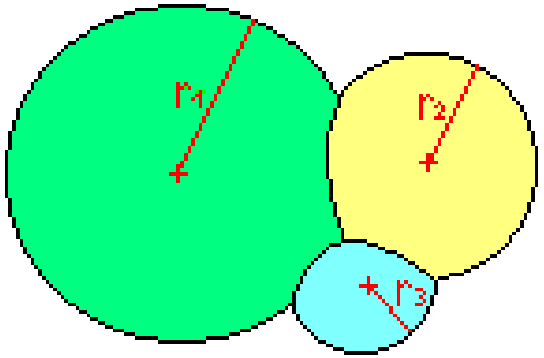


All the limits are straight lines, the radii of the spherulites are equal and kinetic law likes :

$T = 1 - \exp(- kt^2)$

Homogen germination :

Ni = 0, Vg > 0



The limits are more or less curved, the radii of the sperulites are varying and kinetic law likes :

$T = 1 - \exp(- kt^3)$

(T : cristallization rate)

Calculation of random number series

1) Build an **array of integers** containing all the integers between two limits :

```
FOR  I = 1  TO 10 000
     R(I) = I
NEXT I
```

2) mix randomly the array during a sufficient time : (twice or three times the number of values in the array is sufficient)

```
FOR  I = 1  TO  20 000
     A = 1 + int (random * 10 000)
     B = 1 + int (random * 10 000)
     PR = R(A)
     R(A) = R(B)
     R(B) = PR
NEXT  I
```

Remark : Directly allocating random values in the array by a call of the random function leads to a bias in the series (according to normal distribution law) and should cause imperfect simulation results.

Random series (following) – Calling random integers

3) At the beginning of each cycle of the loop, initialize randomly a pointer :

```
PR = 1 + int (random * 10 000)
```

4) Inside the loop, call a random integer from the series through the following way :

```
RI = R(PR)
PR = PR + 1
IF PR = 10001 THEN PR = 1
```

Remark : last statement of (4) can be replaced directly by statement (3) and then the pointer can be initialized just before entering inside the loop.

Advantage of randomly calculated series : calling a random integer from the series is ten times faster than a call of random function.
Inconvenience : Accuracy is then limited by the size of the array.

How to increase the accuracy of calculated random series

Needed : random numbers between 1 and 10^8 (1 < Threshold : K < 10^8)
Available : random series R(I) of integers between 1 and 10 000

1) Calculate the square root of the threshold K : **SK = $K^{1/2}$** (1 < SK < 10 000)

2) Initialize two pointers :

```
PR1 = 1 + int (random * 10 000)
PR2 = 1 + int (random * 10 000)
```

3) Call two random integers from the series :

SK should be converted as an integer.

```
R1 = R(PR1)
PR1 = PR1 + 1
IF PR1 = 10001 THEN PR1 = 1
R2 = R(PR2)
PR2 = PR2 + 1
IF PR2 = 10001 THEN PR2 = 1
```

4) Test the values through a double condition test :

```
IF R1 <= SK AND R2 <= SK THEN ---
```