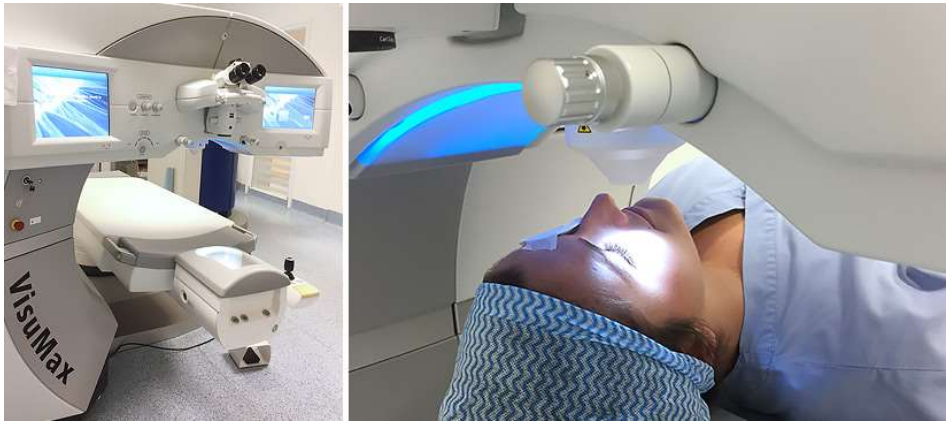
CAILLON Armand - 11796
Avec BARBIER Lucas 40342

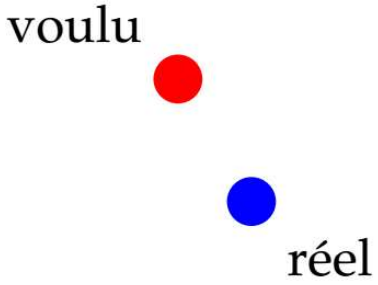
# Mesure des défauts de l'œil

TIPE – Année 2022

# Objectifs




Source : cliniquelamartine.fr

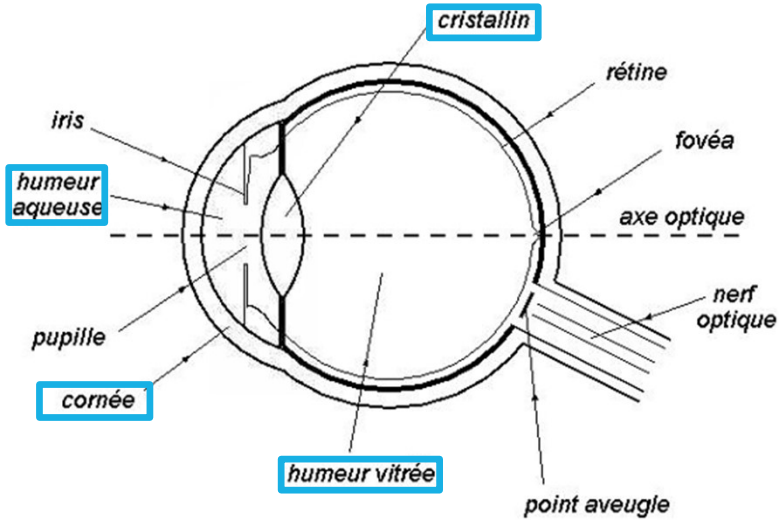


# Sommaire

# I. Introduction

## a) Présentation générale

L'œil

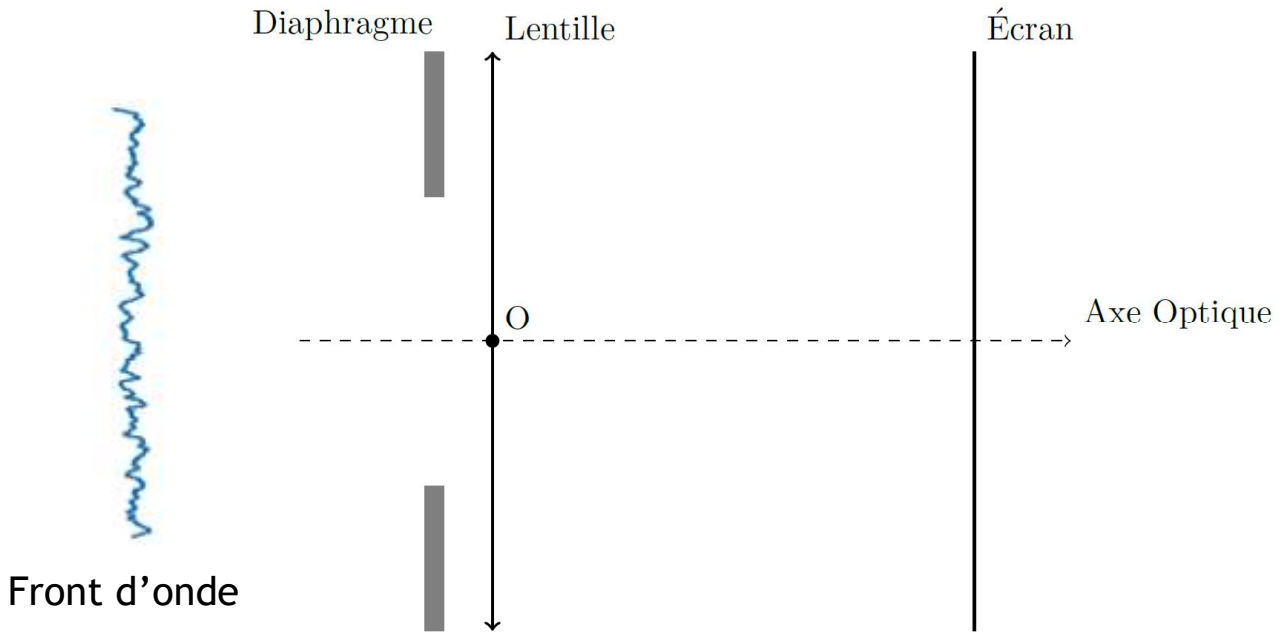



Source : maxicours.com

Quelques origines des défauts :

- Cristallin
- Humeurs

# I. Introduction
## b) Modélisation de l'œil




Modèle plan de l'oeil

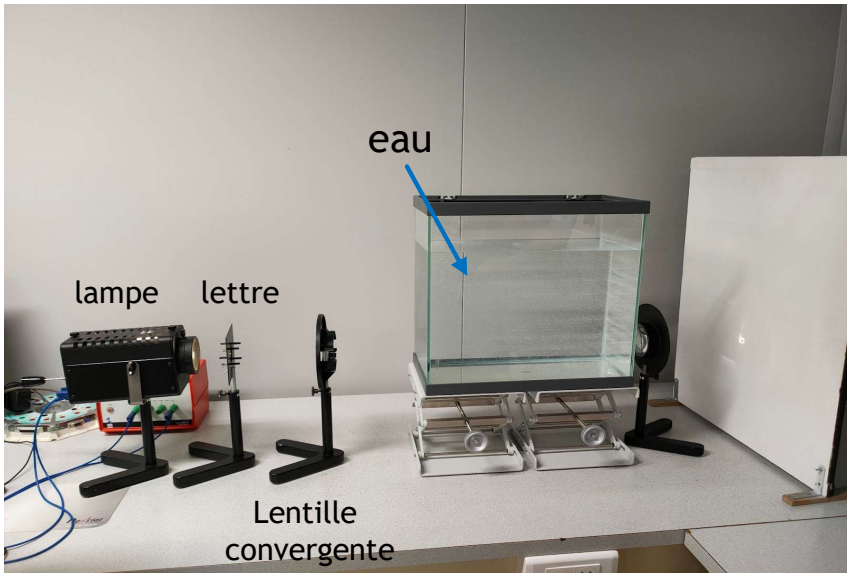
# I. Introduction
## c) Différents défauts de l'œil

| Défauts classiques | Défauts de haut degré |
| --- | --- |
| - Myopie<br>- Hypermétropie<br>- Astigmatisme<br>- Presbytie… | - Inhomogénéités dans les humeurs<br>- Défauts du cristallin (Cornée déformée) |

# II. Mise en évidence de la déformation d'un front d'onde par les inhomogénéités

## a) Expérience

Montage

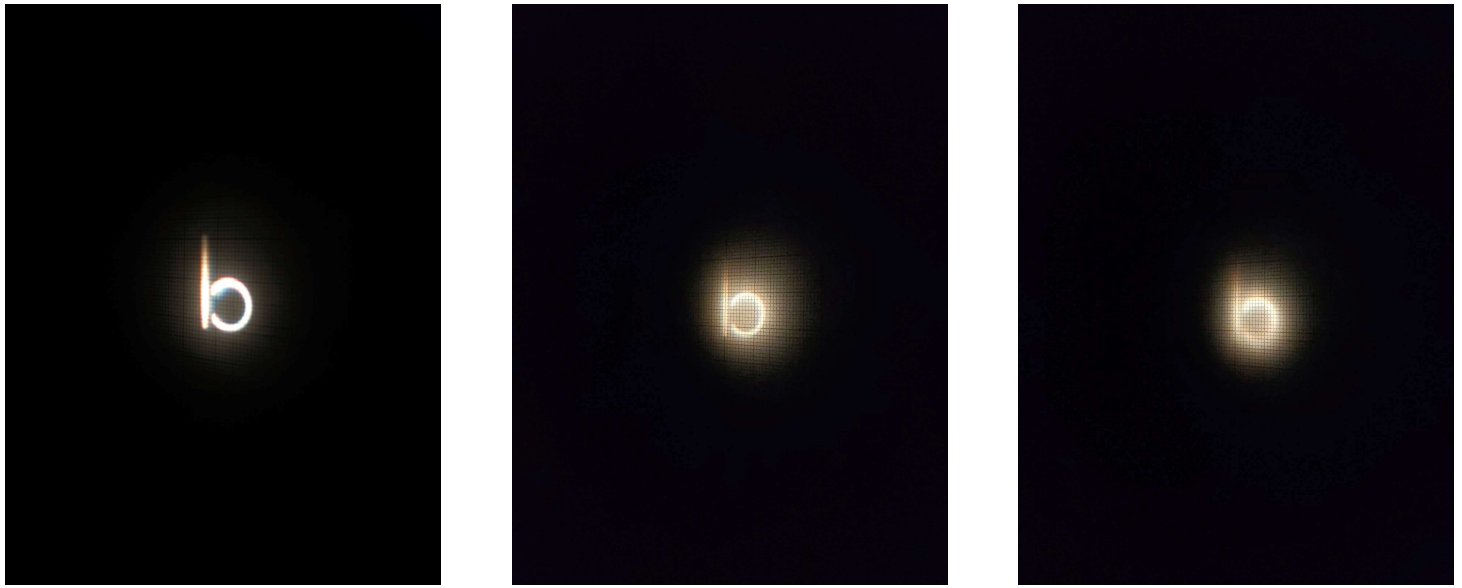


# II. Mise en évidence de la déformation d'un front d'onde par les inhomogénéités

## a) Expérience

**<u>À t=0 :</u>** Ajout d'eau concentrée en sucre → **Milieu Hétérogène**

**<u>Image observée :</u>**



t

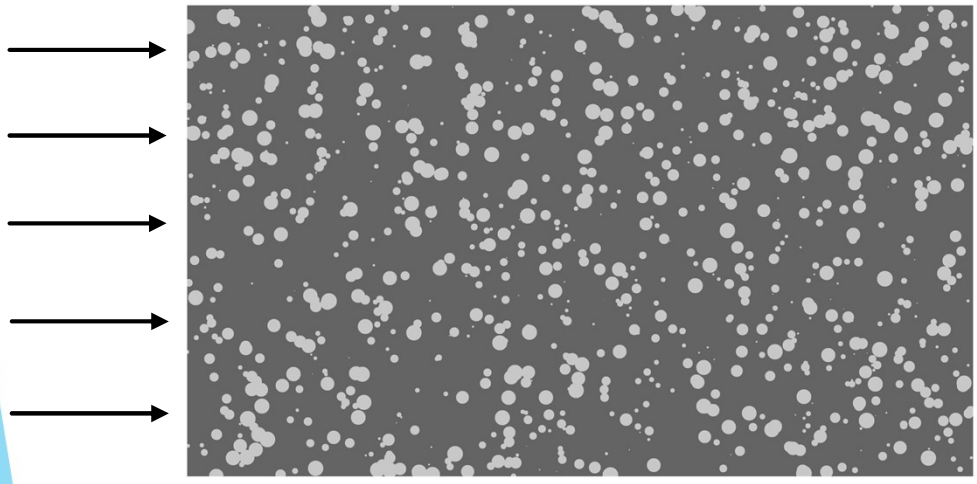
0

**<u>Deux phénomènes entrent en jeu ici :</u>**

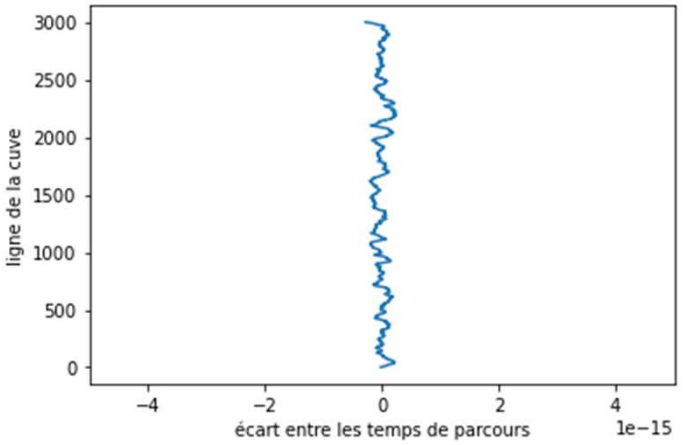
- La réfraction
- La différence de temps de parcours

# II. Mise en évidence de la déformation du front d'onde par les inhomogénéités
## b) Simulation : estimation du temps de parcours

Lumière en phase



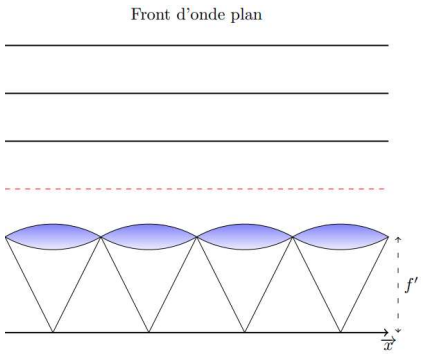
**Simulation des inhomogénéités**

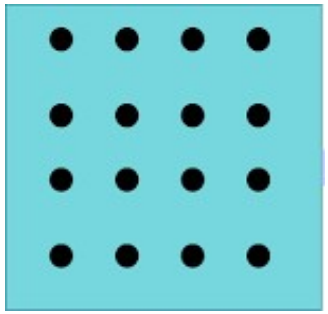



**Front d'onde après traversée de la cuve**
(décalage en fonction du temps)

# III. Mesure de la déformation d'un front d'onde avec la méthode inspirée de Shack-Hartmann

## La méthode Shack-Hartmann

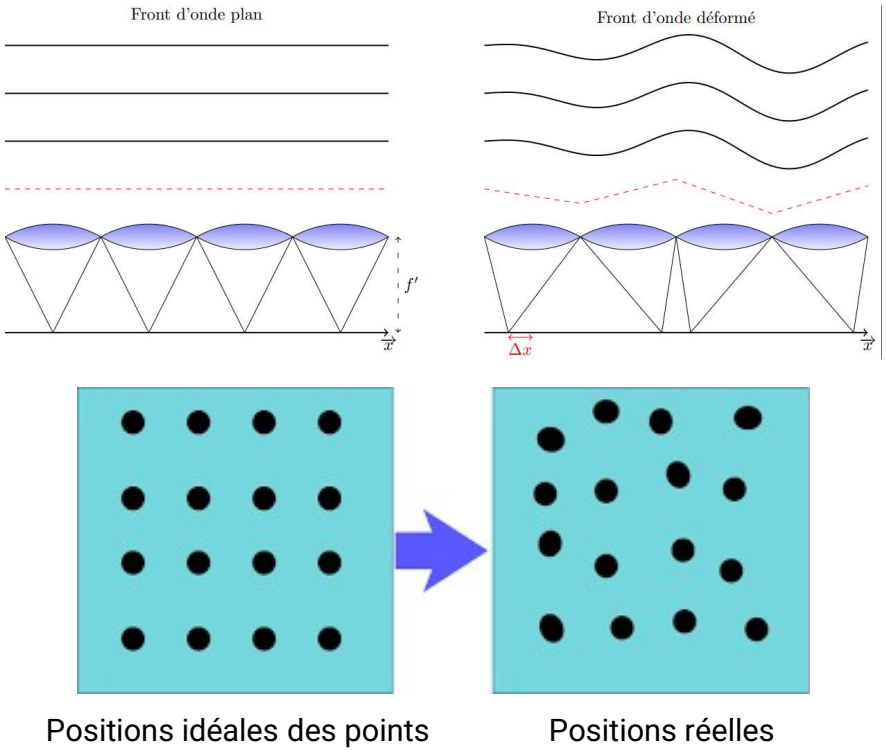


Positions idéales des points

# III. Mesure de la déformation d'un front d'onde avec la méthode inspirée de Shack-Hartmann

La méthode Shack-Hartmann



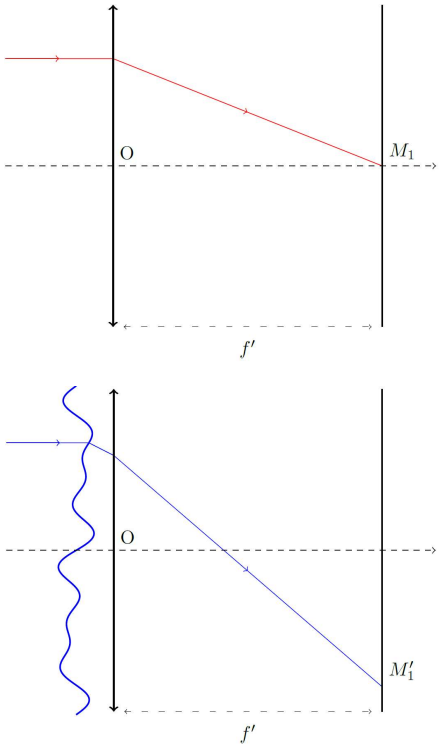

Positions idéales des points

Positions réelles

# III. Mesure de la déformation d'un front d'onde avec la méthode inspirée de Shack-Hartmann

## a) Expérience

**Observation** :
Le Rayon est dévié par le plastique par rapport à une trajectoire idéale

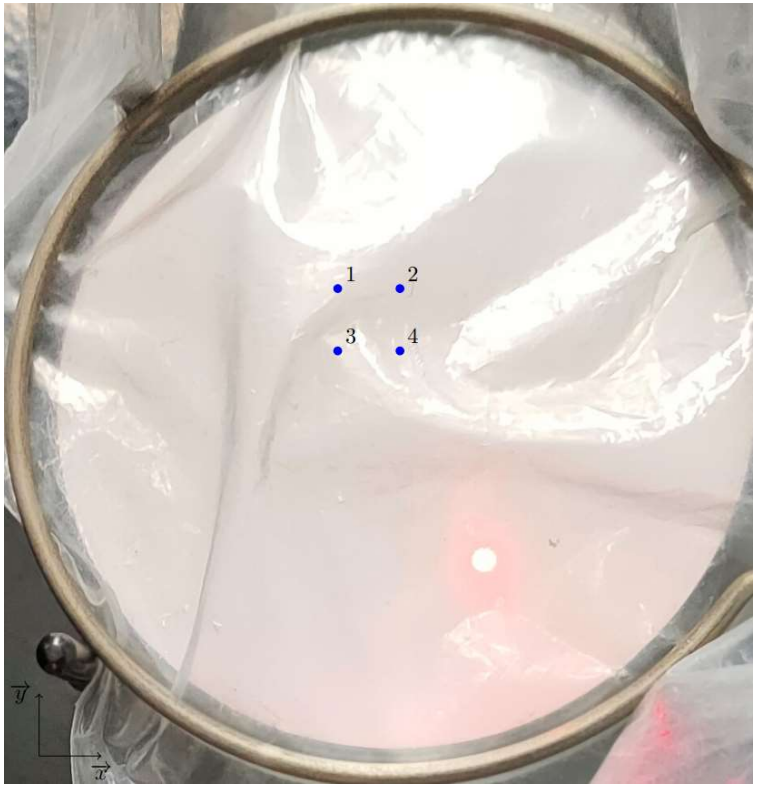



**Montage :**
- Laser
- Lentille convergente
- Plastique : surface déformée

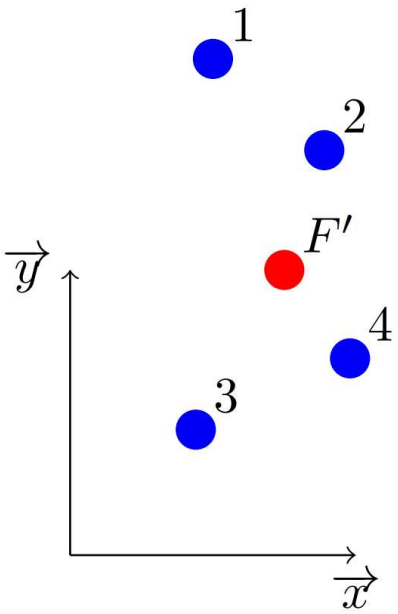
# III. Mesure de la déformation d'un front d'onde avec la méthode inspirée de Shack-Hartmann

## b) Données obtenues

Rayons arrivant sur la lentille




Rayons arrivant dans le plan focal image

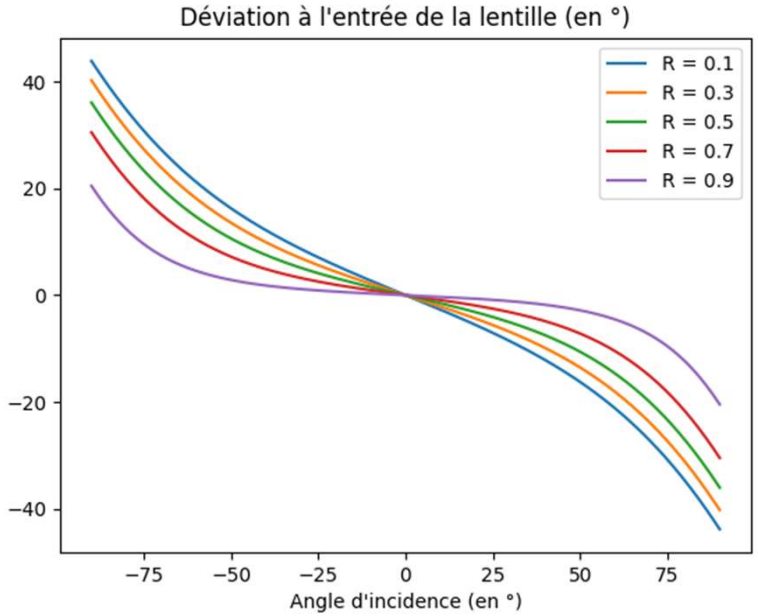


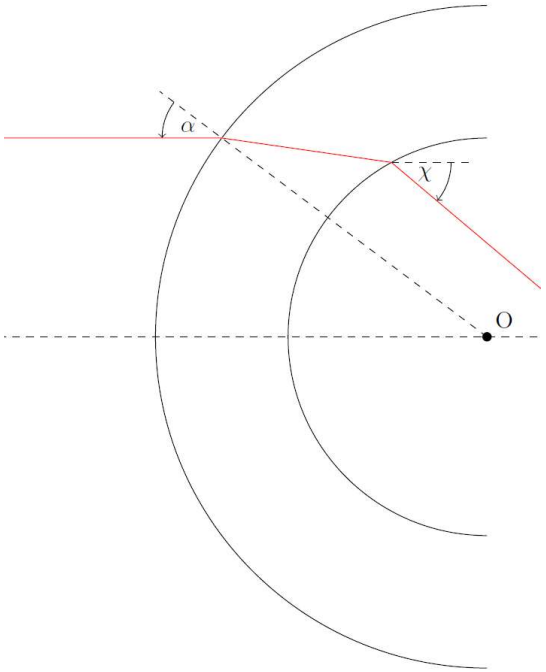
# III. Mesure de la déformation d'un front d'onde avec la méthode inspirée de Shack-Hartmann

## b) Données obtenues

## Modélisation de la déviation du faisceau

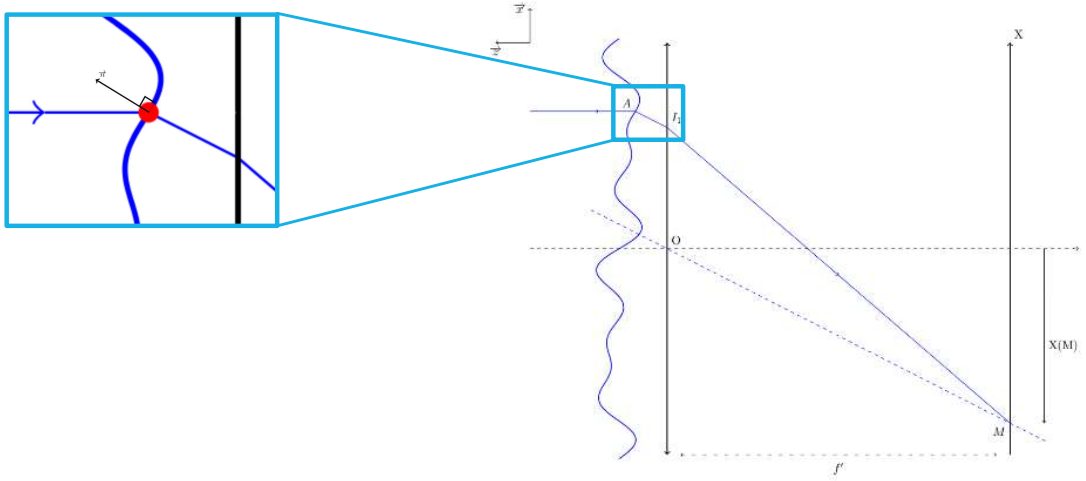
On assimile localement le pochon à deux sphères concentriques de rayon différent



# IV. Traitement informatique du front d'onde reçu

## a) Détermination du front d'onde aux points d'échantillonnage

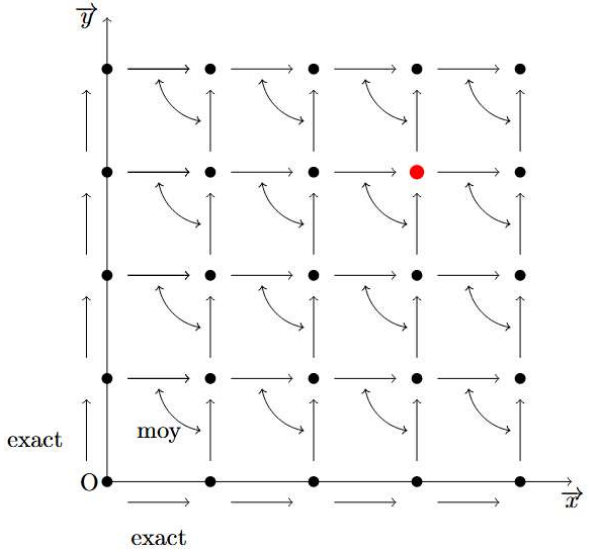



Détermination de la pente de la surface :

$$\frac{\partial z}{\partial x} = \frac{-X(M)}{f'} \qquad \frac{\partial z}{\partial y} = \frac{-Y(M)}{f'}$$

# IV. Traitement informatique du front d'onde reçu

## a) Détermination du front d'onde aux points d'échantillonnage




$$\forall (i,j) \in [\![1, n-1]\!]^2, z(i,j) = \frac{1}{2}\left[z(i-1,j) + \frac{\partial z}{\partial x}(i-1,j)\delta x + z(i,j-1) + \frac{\partial z}{\partial y}(i,j-1)\delta y\right]$$

# IV. Traitement informatique du front d'onde reçu
## b) Interpolation de Lagrange

$$Z(X,Y) = \sum_{i=0}^{n} \sum_{j=0}^{m} z_{i,j} L_i^x(X) L_j^y(Y)$$

$$L_i^x(X) = \prod_{\substack{k=1 \\ k \neq i}}^{n} \frac{X - x_k}{x_i - x_k}$$

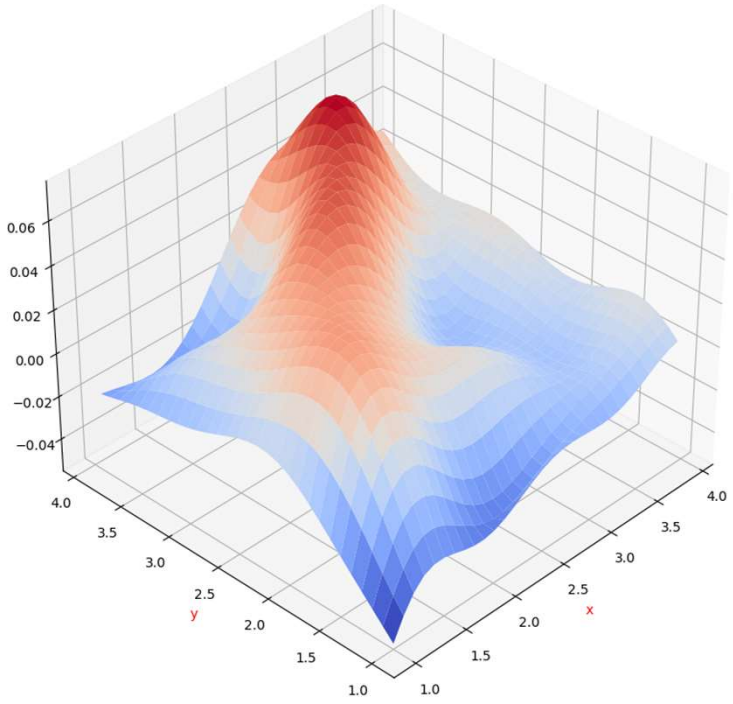
Interpolateur selon x

$$L_j^y(Y) = \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{Y - y_k}{y_j - y_k}$$

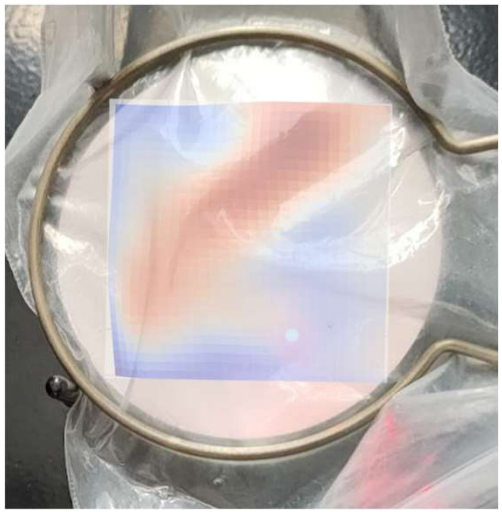
Interpolateur selon y

# IV. Traitement informatique du front d'onde reçu
## b) Interpolation de Lagrange

**Reconstitution du front d'onde**




Le front d'onde interpolé « ressemble » au pochon



# IV. Traitement informatique du front d'onde reçu
## c) Caractérisation de la déformation du front d'onde obtenu

Les polynômes de Zernike sont une base des fonctions polynomiales de $\mathcal{D}$ dans $\mathbb{R}$, elles mêmes denses dans $\mathcal{C}^0(\mathcal{D},\mathbb{R})$ où $\mathcal{D} = \{(x,y) \in \mathbb{R}^2,\ x^2+y^2 \leq 1\}$

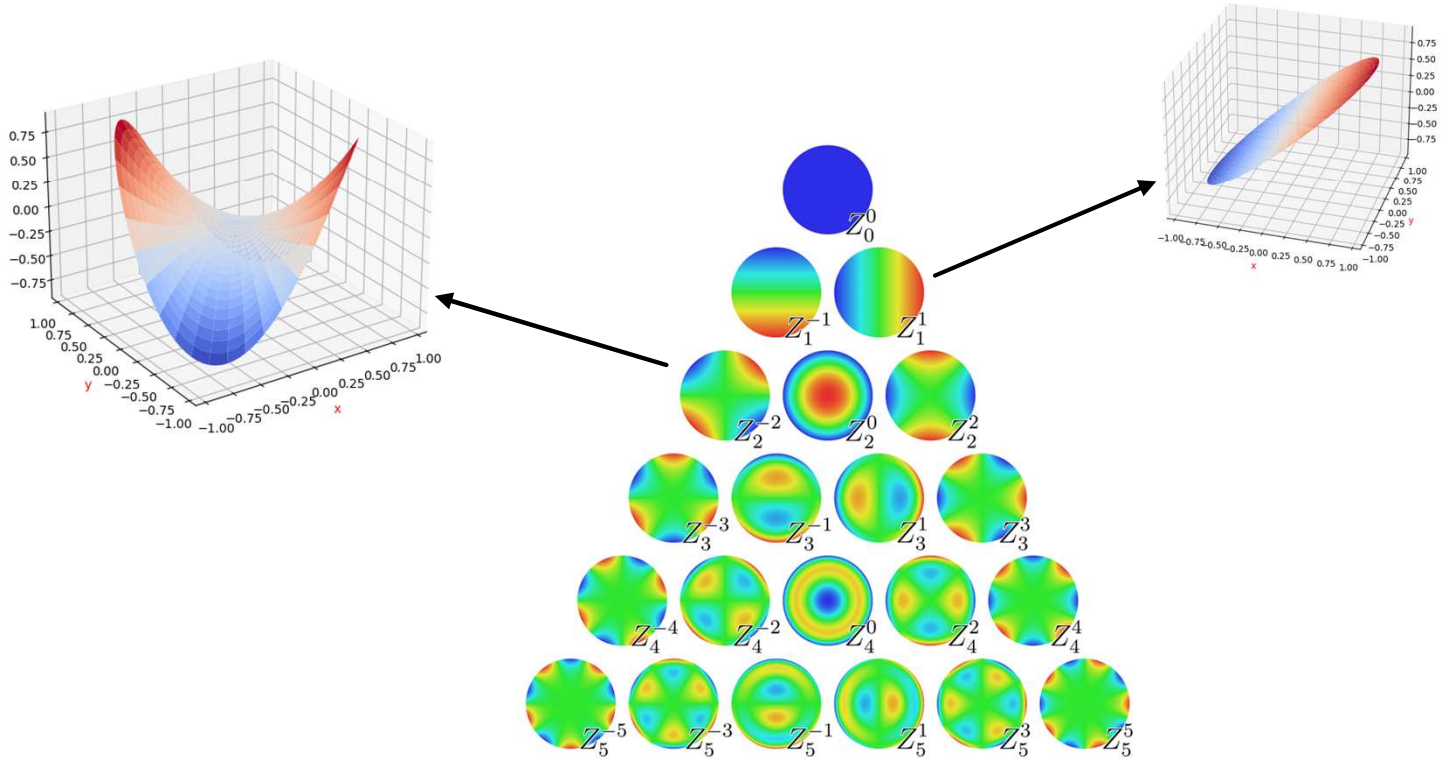
Polynômes de Zernike d'ordre n

$$Z_n^m(\rho,\theta) = \begin{cases} R_n^m(\rho) \times cos(m\theta) & m \geq 0 \\ R_n^m(\rho) \times sin(m\theta) & m < 0 \end{cases} \qquad \mathrm{R}_n^m(\rho) = \sum_{k=0}^{\frac{n-m}{2}} \frac{(-1)^k (n-k)!}{k!(\frac{n+m}{2}-k)!(\frac{n-m}{2}-k)!} \rho^{n-2k}$$

m = nombre de méridiens affectés
n = ordre du polynôme

# IV. Traitement informatique du front d'onde reçu
## c) Caractérisation de la déformation du front d'onde obtenu

**Les polynômes de Zernike en physique**



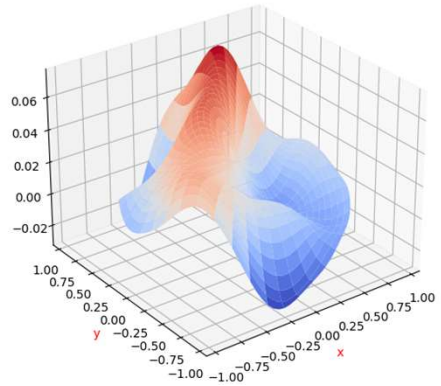

Source : Pinterest.com

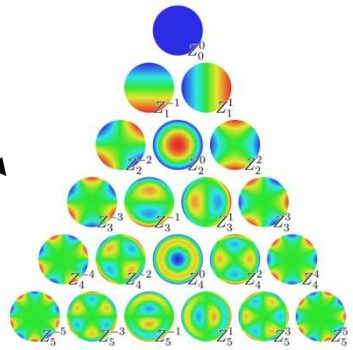
# IV. Traitement informatique du front d'onde reçu
## c) Caractérisation de la déformation du front d'onde obtenu

**Décomposition de la déformation du front d'onde sur la base de Zernike**

Produit scalaire

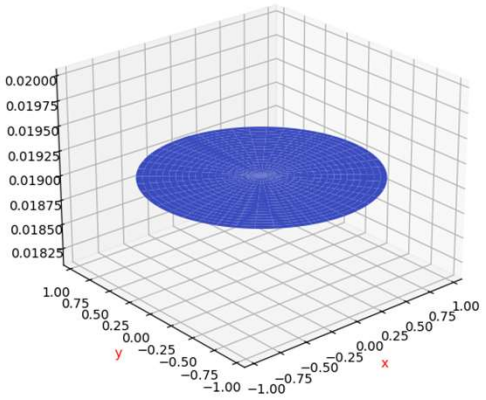
$$< f_1, f_2 >= \frac{1}{\pi} \iint_D f_1(\rho, \Phi) f_2(\rho, \Phi) \rho \, d\rho \, d\Phi$$



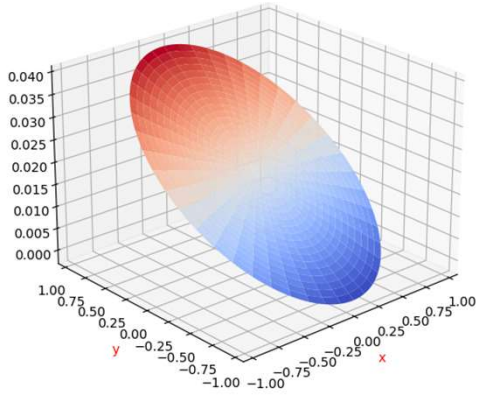
# IV. Traitement informatique du front d'onde reçu
## c) Caractérisation de la déformation du front d'onde obtenu

**Résultat obtenu après projection : ordres 0 et 1**



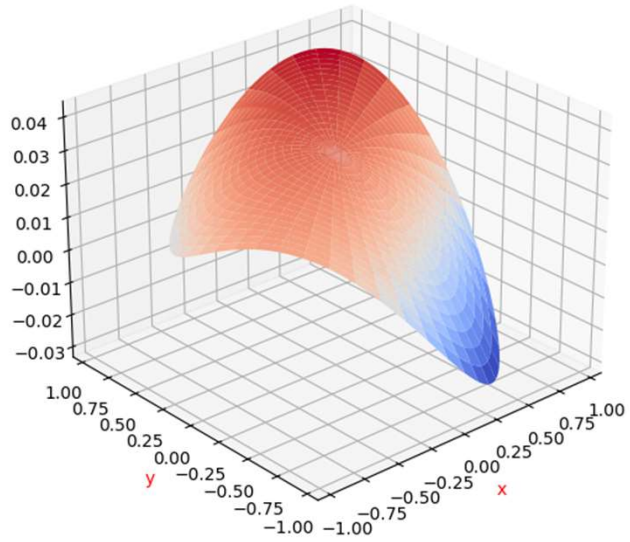

Ordre 0

Ordre 1



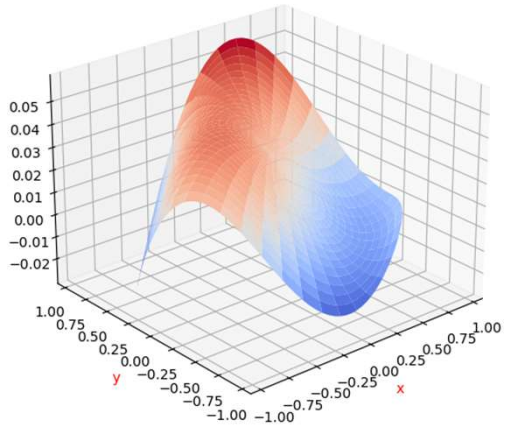
# IV. Traitement informatique du front d'onde reçu
## c) Caractérisation de la déformation du front d'onde obtenu

**Résultat obtenu après projection : ordre 2**



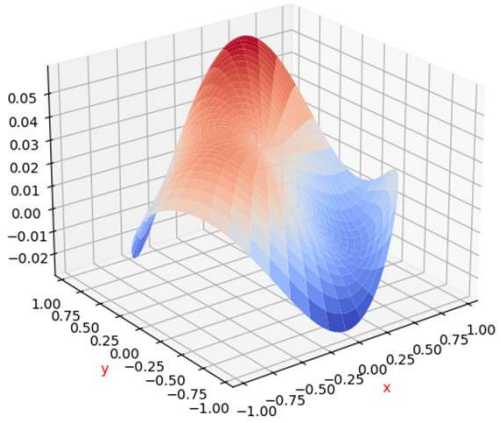
# IV. Traitement informatique du front d'onde reçu
## c) Caractérisation de la déformation du front d'onde obtenu

**Résultat obtenu après projection : ordres 3 et 4**



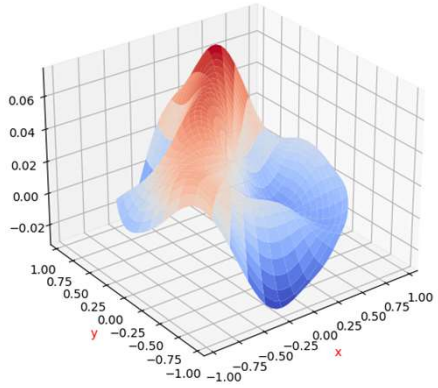

Ordre 3

Ordre 4



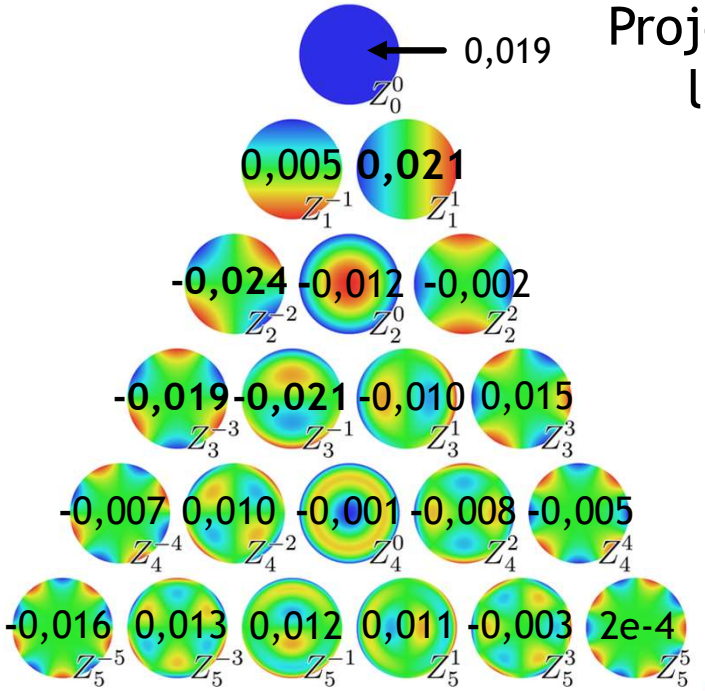
# IV. Traitement informatique du front d'onde reçu
## c) Caractérisation de la déformation du front d'onde obtenu

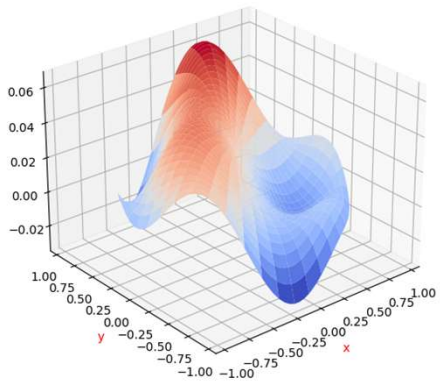
**Projection sur la base des Polynômes de Zernike**



Réel

Résultats



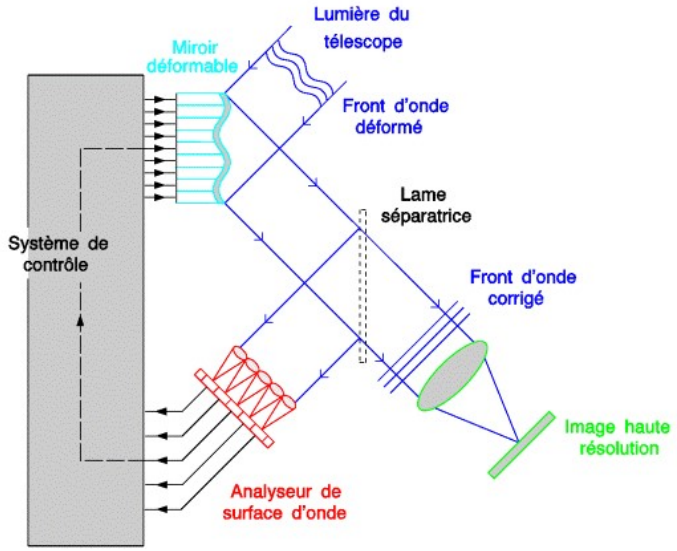



Projeté jusqu'à l'ordre 5

# Conclusion

Mesure précise des défauts :

⇒ Elaboration d'une correction OU changement de la position du rayon




Source : media4.obspm.fr

# Annexes

# Preuve pour la remontée à la surface

**Théorème de Malus :**
Les rayons lumineux sont perpendiculaires aux surfaces d'ondes.

Ainsi, en utilisant le théorème de Malus à l'entrée de la lentille, on en déduit que $\overrightarrow{AI_1}$ est un vecteur normal à la surface d'onde.

Or par construction $\overrightarrow{AI_1} // \overrightarrow{OM}$

De plus, $\overrightarrow{OM} = X(M)\overrightarrow{u_x} + Y(M)\overrightarrow{u_y} - f'\overrightarrow{u_z}$

On en déduit donc $\overrightarrow{n}$ vecteur normal unitaire à la surface d'onde au point de mesure $A$ par

$$\overrightarrow{n} = \frac{X(M)\overrightarrow{u_x} + Y(M)\overrightarrow{u_y} - f'\overrightarrow{u_z}}{\sqrt{X(M)^2 + Y(M)^2 + f'^2}}$$

Où $X(M), Y(M)$ et $f'$ sont connus grâce à l'expérience

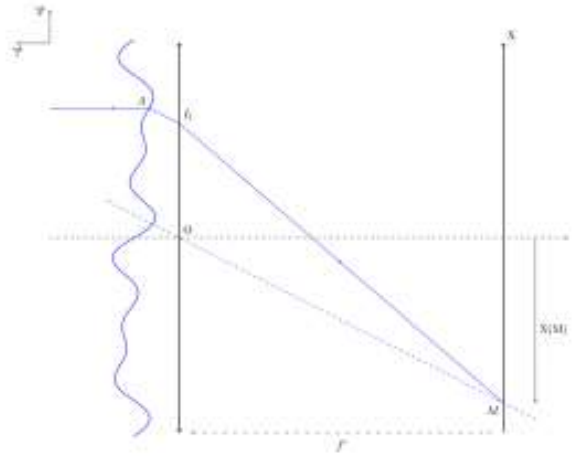


Figure 8: Schéma en vue de coupe selon (xOz)

Soit $h : x \mapsto f(x, y_0)$ alors l'expression de la tangente en $x_0$ de $h$ est

$$T_{x_0} : z = h'(x_0)(x - x_0) + h(x_0) = \frac{\partial z}{\partial x}(M)(x - x_0) + z(M)$$

Ainsi un vecteur directeur de $T_{x_0}$ est : $\overrightarrow{T_{x_0}} = \overrightarrow{u_x} + \frac{\partial z}{\partial x}(M)\overrightarrow{u_z}$

De même selon $\overrightarrow{u_y}$ on obtient : $\overrightarrow{T_{y_0}} = \overrightarrow{u_y} + \frac{\partial z}{\partial y}(M)\overrightarrow{u_z}$

Les connaissances d'une part du vecteur normal et d'autre part des vecteurs tangents à la surface selon $\overrightarrow{u_x}$ et $\overrightarrow{u_y}$ nous permettent de trouver les relations entre les dérivées partielles de $z, X(M)$ et $Y(M)$.

$$\overrightarrow{n} = \frac{X(M)\overrightarrow{u_x} + Y(M)\overrightarrow{u_y} - f'\overrightarrow{u_z}}{\sqrt{X(M)^2 + Y(M)^2 + f'^2}} = \alpha\overrightarrow{u_x} + \beta\overrightarrow{u_y} + \gamma\overrightarrow{u_z}$$

$$\overrightarrow{T_x} = \frac{\overrightarrow{T_{x_0}}}{\|\overrightarrow{T_{x_0}}\|}, \quad \overrightarrow{T_y} = \frac{\overrightarrow{T_{y_0}}}{\|\overrightarrow{T_{y_0}}\|}$$

En exploitant le fait que $\overrightarrow{n} \cdot \overrightarrow{T_x} = 0$ on obtient : $\frac{\partial z}{\partial x}(M) = -\frac{\alpha}{\gamma} = \frac{X(M)}{f'}$

De même avec $\overrightarrow{n} \cdot \overrightarrow{T_y} = 0$ on obtient : $\frac{\partial z}{\partial y}(M) = -\frac{\beta}{\gamma} = \frac{Y(M)}{f'}$

# Preuve pour la remontée à la surface

Il nous faut maintenant reconstituer à partir de la connaissance des dérivées partielles de $z(x, y)$. Tout d'abord, ce qui nous intéresse est la position relative des points dans l'espace ; ainsi, sans pertes de généralités, nous pouvons fixer le point $z(0, 0) = 0$.

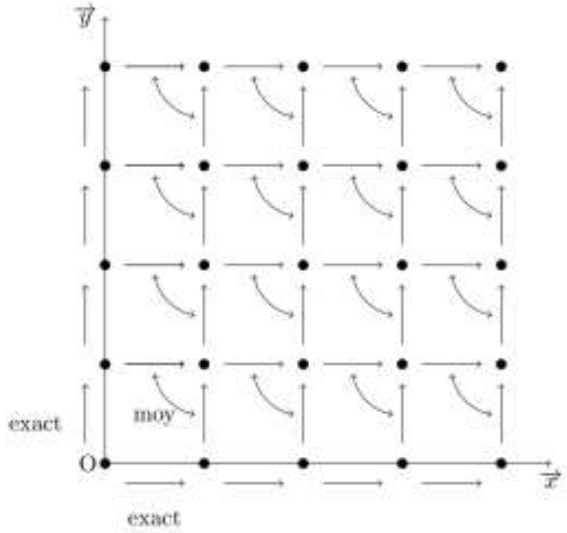



Figure 9: Méthode pour reconstituer la surface

Nous avons élaboré la méthode suivante pour reconstituer la surface d'onde :

1. Calculer à l'aide d'un développement limité à l'ordre 1 les points sur $Ox$ et $Oy$
2. Calculer les autres points à l'aide d'une moyenne des points situés à gauche et en-dessous

Ainsi, pour un maillage de taille $n \times n$ on a:

1. $z(0, 0) = 0$
   Sur $Ox$ : $\forall i \in [\![1, n-1]\!], z(i, 0) = z(i-1, 0) + \frac{\partial z}{\partial x}(i-1, 0)\delta x$
   Sur $Oy$ : $\forall j \in [\![1, n-1]\!], z(0, j) = z(0, j-1) + \frac{\partial z}{\partial y}(0, j-1)\delta y$
2. $\forall (i, j) \in [\![1, n-1]\!]^2, z(i, j) = \frac{1}{2}\left[z(i-1, j) + \frac{\partial z}{\partial x}(i-1, j)\delta x + z(i, j-1) + \frac{\partial z}{\partial y}(i, j-1)\delta y\right]$

Nous obtenons ainsi un nuage de points qui correspond à une version discrète de la surface d'onde (mettre nuage de points)

# Simulation cuve

```
import numpy as np
from PIL import Image
from pylab import *
import matplotlib.pyplot as plt

def tableau(i,j):
    return [[4.44*10**(-13) for a in range(j)] for k in range(i)],[[100 for a in
range(j)] for k in range(i)]
# 4.44*10**(-13) représente le temps de parcours de la case

def dimtab2D(t):
    return len(t), len(t[0])

def bulle(t1,t2): #fonction qui crée une bulle dans la cuve , modifie 2 tableaux un
pour le trajet, l'autre pour l'affichage
    i,j=dimtab2D(t1)
    r=randint(1,50) # rayon de la bulle
    (a,b)=(randint(0,i),randint(0,j)) # place de la bulle
    for p in range(-r,r+1):
        for w in range(-r,r+1):
            if floor(sqrt(p**2+w**2))<=r and -1<a-p<i and -1<b-w<j: #on regarde qu'on
est bien à une distance r du centre
                t1[a-p][b-w]=4.92*10**(-13) # on modifie la valeur du temps de parcours
                t2[a-p][b-w]=200


t1,t2=tableau(3000,5000)
for i in range(1000):
    bulle(t1,t2)

def somme(t,i):
    # somme la ième ligne de t
    s = 0
    for p in t[i]:
        s=s+p
    return s

def moyenne(L):
    s=0
    for i in L:
        s=s+i
    return(s/len(L))

def simulation(t):
    L=[]
    A=[]
    for i in range(len(t)):
        A=A+[i]
    A=np.array(A)
    A=len(t)-A
    print(A) # on définit le nombre de lignes sur lesquelles on va travailler
    for i in range(len(t)):
        L=L+[somme(t,i)/100] # on incorpore dans L la durée de parcours de chaque ligne
    print(moyenne(L))
    return (np.array(L)-moyenne(L))/1000,A

X,Y=simulation(t1)
plt.close()
plt.plot(X,Y)
plt.xlim(-5e-15,5e-15)


plt.ylabel('ligne de la cuve')
plt.xlabel('écart entre les temps de parcours')
plt.show()

a=np.array(t2)
image = Image.fromarray(a)
image.show()
```

# Déviation

```
import matplotlib.pyplot as plt
import numpy as np

N=0.7
x=np.linspace(-np.pi/2,np.pi/2,1000)

## Double dioptre concave vers la droite

for i in range(1,10,2):
    R=i*10**(-1)
    y=(np.arcsin(N*np.sin(x))+np.arcsin(R*np.sin(x))-np.arcsin(N*R*np.sin(x))-
x)*(180/np.pi)
    plt.plot(x*180/np.pi,y,label = f'R = {R:.2}')
    plt.legend()

plt.title("Déviation à l'entrée de la lentille (en °)")
plt.xlabel("Angle d'incidence (en °)")
plt.show()
```

# Polynômes

```
import copy
p1=[[1,2,3],[0,3],[0,3,4,0,0]]
p2=[[2,3],[0,2,0]]

#fonctions à 1 variable utiles
def normalize1v (pol1):
    s=copy.deepcopy (pol1)
    while s[-1]==0 and len(s)>1:
        s.pop(len(s)-1)
    return s

def somme1v (pol1,pol2):
    p=copy.deepcopy (pol1)
    q=copy.deepcopy(pol2)
    p1=normalize1v(p)
    q1=normalize1v(q)
    n,m=len(p1),len(q1)
    if n > m :
        s = p1
        for i in range (m) :
            s[i] += q[i]
    else :
        s = q1
        for i in range (n) :
            s[i] += p[i]
    return s

def mult_scal1v (pol,x):
    p=copy.deepcopy (pol)
    s=normalize1v(p)
    for i in range (len(s)):
        s[i] *= x
    return s

def mult_monom1v (pol,x,i):
    """Multiplie le polynôme par X^i"""
    p=copy.deepcopy (pol)
    m=mult_scal1v (p,x)
    s=[0]*i
    return s+m

def mult1v (pol1,pol2):
    p=copy.deepcopy (pol1)
    q=copy.deepcopy (pol2)
    p1,q1=normalize1v(p),normalize1v(q)
    m=[0]
    for i in range (len(p1)):
        s=mult_monom1v(q1,p1[i],i)
        m=somme1v(m,s)
    return normalize1v(m)

#polynome 2 variables
def  normalize2v (pol):
    p=copy.deepcopy(pol)
    n=len(p)
    for i in range (n):
        p[i]=normalize1v(p[i])
    while p != [[0]] and p[-1] == [0]:
        p.pop(-1)
    m=len(p[0])
    n=len(p)
    for k in range (n):
        m=max (m, len(p[k]))
    for i in range (n):
        l=len(p[i])
        while l<m:
            p[i].append(0)
            l=len(p[i])
    return p

def somme2v (pol1,pol2):
    p=copy.deepcopy(pol1)
    q=copy.deepcopy(pol2)
    n=len(p)
    m=len(q)
    if n<m :
        s=copy.deepcopy(q)
        for i in range (n):
            s[i]=somme1v(p[i],q[i])
    else:
        s=copy.deepcopy(p)
        for i in range (m):
            s[i]=somme1v(p[i],q[i])
    return normalize2v(s)

def mult_scal2v (pol,x):
    p=copy.deepcopy (pol)
    s=normalize2v(p)
    for i in range (len(s)):
        for j in range (len(s[0])):
            s[i][j] *= x
    return s

def mult_monom2v (pol,x,i,j):
    p=copy.deepcopy (pol)
    s=mult_scal2v(p,x)
    n=len(s)
    m=len(s[0])
    z1=[0]*j
    for k in range (n):                #rajoute à chaque sous liste autant de zéro
que le degré du monom en Y
        s[k]=z1+s[k]
    z2=[[0]*(m+j)]*i                   #decale le degré du polynome en X
    return z2+s

def mult2v (pol1,pol2):
    p1=copy.deepcopy(pol1)
    p2=copy.deepcopy(pol2)
    s1,s2=normalize2v(p1),normalize2v(p2)
    n,m = len(s1),len(s1[0])
    f=[[0]]
    for i in range (n):
        for j in range (m):
            f=somme2v(f,mult_monom2v(s2, s1[i][j], i, j))
    return f

##Lagrange 2 variables
def lagrangeY (l,i):
    L=[1]
    for j in range (len(l)):
        if j==i:
            L=L
        else:
            L=mult1v(L,[-l[j]/(l[i]-l[j]),1/(l[i]-l[j])])
    return L

def lagrangeX (l,i):
    L=[[1]]
    for j in range (len(l)):
        if j==i:
            L=L
        else:
            L=mult2v(L,[[-l[j]/(l[i]-l[j])],[1/(l[i]-l[j])]])
    return L

def interpole2v (lx,ly,t):
    Lx=[]
    Ly=[]
    n=len(lx)
    m=len(ly)
    inter=[[0]]
    for k in range (n):
        Lx.append(lagrangeX(lx,k)) # Calcul des Lagrangiens selon x
    for l in range (m):
        Ly.append([lagrangeY(ly,l)]) # Calcul des Lagrangiens selon y
    for i in range (n):
        for j in range (m):
            inter=somme2v(inter,mult_scal2v(mult2v(Lx[i],Ly[j]),t[i][j])) # Somme pour
chaque point
    return inter
```

# Représentation plan

```
from Polynomes import *
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import math


def calc_z_coin (t,f, pasx,pasy) :
    m = len(t)
    n = len(t[0])
    Z = [[0 for _ in range (n)] for _ in range (m)]
    for i in range (1,n):
        X,Y = t[0][i-1]
        Z[0][i]=Z[0][i-1]-X/f*pasx
    for i in range (1,m):
        X,Y = t[i-1][0]
        Z[i][0]=Z[i-1][0]-Y/f*pasy
    for i in range (1,m):
        for j in range (1,n):
            X1 , Y1 = t[i][j-1]
            X2 , Y2 = t[i-1][j]
            Z[i][j] = 0.5*(Z[i][j-1]-X1/f*pasx + Z[i-1][j]-Y2/f*pasy)
    return Z

def calc_z_centre (t,f, pasx,pasy) :
    m = len(t)
    n = len(t[0])
    Z = [[0 for _ in range (n)] for _ in range (m)]
    if m  % 2 ==0 :
        m1=m//2
    else:
        m1=m//2 + 1
    if n%2 == 0:
        n1=n//2
    else :
        n1=n//2 + 1
    for i in range (n1,n):
        X,Y = t[m1][i-1]
        Z[m1][i]=Z[m1][i-1]-X/f*pasx
    for i in range (n1-1,-1,-1):
        X,Y = t[m1][i+1]
        Z[m1][i]=Z[m1][i+1]+X/f*pasx
    for i in range (m1,m):
        X,Y = t[i-1][n1]
        Z[i][n1]=Z[i-1][n1]-Y/f*pasy
    for i in range (m1,-1,-1):
        X,Y = t[i+1][n1]
        Z[i][n1]=Z[i+1][n1]+Y/f*pasy
    for i in range (m1,m):
        for j in range (n1,n):
            X1,Y1=t[i][j-1]
            X2, Y2 = t[i-1][j]
            Z[i][j] = 0.5*(Z[i][j-1]-X1/f*pasx + Z[i-1][j]-Y2/f*pasy)
    for i in range (m1,m):
        for j in range (n1-1,-1,-1):
            X1,Y1=t[i][j+1]
            X2, Y2 = t[i-1][j]
            Z[i][j] = 0.5*(Z[i][j+1]+X1/f*pasx + Z[i-1][j]-Y2/f*pasy)
    for i in range (m1-1,-1,-1):
        for j in range (n1,n):
            X1,Y1=t[i][j-1]
            X2, Y2 = t[i+1][j]
            Z[i][j] = 0.5*(Z[i][j-1]-X1/f*pasx + Z[i+1][j]+Y2/f*pasy)
    for i in range (m1-1,-1,-1):
        for j in range (n1-1,-1,-1):
            X1,Y1=t[i][j+1]
            X2, Y2 = t[i+1][j]
            Z[i][j] = 0.5*(Z[i][j+1]+X1/f*pasx + Z[i+1][j]+Y2/f*pasy)
    return Z

t=[[(0,0),(-1,-1),(0,1),(-1,1),(-1,0),(1,1),(-1,0),(-1,1),(-1,0),(-1,1),(-1,1),(-1,2),
(0,0)],
   [(2,1),(1,1),(0,0),(-1,0),(-2,0),(-1,0),(0,0),(0,0),(0,-1),(-1,1),(-1,1),(-2,1),(-2,
1)],
   [(1,0),(0,0),(0,1),(-2,0),(-2,-1),(-1,2),(0,1),(0,0),(-1,1),(0,-1),(0,1),(-1,1),(-1,
3)],
   [(1,0),(0,0),(0,0),(-2,-1),(0,-1),(1,-1),(0,-1),(0,-1),(0,0),(0,-1),(-2,-1),(-1,0),
(-1,0)],
   [(2,0),(0,-1),(0,0),(-2,0),(-1,0),(0,0),(1,0),(1,0),(-1,-2),(0,0),(0,0),(-1,-1),(-1,
0)],
   [(1,0),(0,0),(1,0),(-1,0),(0,0),(0,0),(0,0),(0,-1),(0,0),(0,0),(2,0),(-1,1),(-1,-
1)],
   [(1,0),(0,0),(2,0),(-1,0),(0,0),(0,0),(-2,2),(0,0),(0,0),(-1,0),(-1,1),(0,0),(-1,-
1)],
   [(2,-2),(0,0),(1,-1),(0,-1),(0,0),(0,0),(0,0),(0,0),(-1,0),(0,0),(0,-1),(-2,0),(-1,-
1)],
   [(1,-1),(-1,1),(1,0),(-1,0),(0,0),(0,0),(0,-1),(0,0),(0,0),(0,0),(-1,0),(-1,-1),(-2,
-1)],
   [(0,0),(1,-1),(-1,0),(2,0),(0,0),(0,-1),(0,0),(0,0),(0,0),(0,0),(0,-1),(0,0),(0,0)],
   [(0,0),(1,-1),(-1,0),(1,-2),(0,-1),(1,-1),(0,0),(0,-1),(0,-1),(-1,-1),(0,0),(0,0),
(0,0)]]


plan_z1=calc_z_centre(t,200,5,5)
n1=len(plan_z1[0])
m1=len(plan_z1)
lx=[i*0.5 for i in range (m1)]
ly=[j*0.5 for j in range (n1)]


z1=interpole2v(lx,ly,plan_z1)

def plan1 (x,y):
    val=0
    n=len(z1)
    m=len(z1[0])
    for i in range (n):
        for j in range (m):
            val += x**i * y**j * z1[i][j]
    return val

def plan1_dec(x,y):
    return plan1(3/2*x+2.5,3/2*y+2.5)

def plan1_polaire_decale(rho,phi):
    return plan1(3/2*rho*math.sin(phi)+2.5,3/2*rho*math.cos(phi)+2.5)

def plan2 (x,y):
    val=0
```

# Représentation plan

```
    n=len(z2)
    m=len(z2[0])
    for i in range (n):
        for j in range (m):
            val += x**i * y**j * z2[i][j]
    return val

ax = Axes3D(plt.figure())
plan1_dec = np.vectorize(plan1_dec)
R = np.arange(0,1,0.05)
Phi = np.arange(0,2*math.pi+0.1,0.05)
R,P = np.meshgrid(R, Phi)
X , Y = R*np.cos(P) , R*np.sin(P)
plan1_polaire_decale = np.vectorize(plan1_polaire_decale)
Z = plan1_dec(X, Y)
plt.xlabel('x',color = 'red')
plt.ylabel('y', color = 'red')
ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
plt.show()

plan_z2 = calc_z_centre(t, 200, 5, 5)
n2 = len(plan_z2[0])
m2 = len(plan_z2)
lx = [i * 0.5 for i in range(m2)]
ly = [j * 0.5 for j in range(n2)]

z2 = interpole2v(lx, ly, plan_z2)

ax = Axes3D(plt.figure())
plan2=np.vectorize(plan2)
X = np.arange(1, 4, 0.1)
Y = np.arange(1, 4, 0.1)
X, Y = np.meshgrid(X, Y)
Z = plan2(X , Y)
ax.plot_surface(X, Y, Z,cmap = cm.coolwarm)
plt.show()
```

# Zernike

```
import math

def Rmn(m,n,rho):
    R = 0
    if (n-abs(m)) % 2 == 0:
        for k in range((n-abs(m))//2+1):
            R += ((-1)**k*math.factorial(n-k)) /
(math.factorial(k)*math.factorial((n+abs(m))/2-k)*math.factorial((n-abs(m))/2-
k))*rho**(n-2*k)
    return R

def Zernike(rho,phi,m=None,n=None):
    if m==None:
        m = Zernike_xy.m
    else:
        Zernike_xy.m = m

    if n==None:
        n = Zernike_xy.n
    else:
        Zernike_xy.n = n
    if m >= 0 :
        return Rmn(m,n,rho)*math.cos(m*phi)
    else:
        return Rmn(m,n,rho)*math.sin(m*phi)

def Zernike_xy(x,y,m=None,n=None):
    if m==None:
        m = Zernike_xy.m
    else:
        Zernike_xy.m = m

    if n==None:
        n = Zernike_xy.n
    else:
        Zernike_xy.n = n
    return Zernike(math.sqrt(x*x+y*y),math.atan2(y,x),Zernike_xy.m,Zernike_xy.n)
```

# Projection Zernike
# sur plusieurs ordres

```
from scipy.integrate import dblquad
from Zernike import *
from Représentation_plan import *


def ps(f,g,domaine_rho,domaine_phi):
    return dblquad(lambda rho, phi : 1/(math.pi)*f(rho,phi)*g(rho,phi)*rho,
domaine_phi[0], domaine_phi[1], lambda rho : domaine_rho[0],
lambda rho : domaine_rho[1], epsabs=1e-5, epsrel=1e-5)[0]

plan_z1=calc_z_centre(t,200,5,5)
lx=[i*0.5 for i in range (m1)]
ly=[j*0.5 for j in range (n1)]

for i in  range (6):

    z1=interpole2v(lx,ly,plan_z1)
    projection_zernike = []
    for n in range(i+1):
        pr = []
        for m in range(-n,n+1,2):
            Zernike_xy(0,0,m,n) #Calcul du polynômes Z(n,m)
            prod = ps(Zernike,plan1_polaire_decale,[0,1],[0,2*math.pi])/ps(Zernike,
Zernike,[0,1],[0,2*math.pi]) #Porjection du front d'onde sur Z(n,m)
            pr.append(prod)
        projection_zernike.append(pr)

    def resultat(x,y):
        res = 0
        for n in range(i+1):
            k = 0
            for m in range(-n,n+1,2):
                Zernike_xy.n, Zernike_xy.m = n,m
                res += (projection_zernike[n][k])*Zernike_xy(x,y)
                k += 1
        return res

    ax = Axes3D(plt.figure())
    resultat = np.vectorize(resultat)
    R = np.arange(0,1,0.05)
    Phi = np.arange(0,2*math.pi+0.1,0.05)
    R,P = np.meshgrid(R, Phi)
    X , Y = R*np.cos(P) , R*np.sin(P)
    Z = resultat(X,Y)
    ax.plot_surface(Y, X, Z, cmap=cm.coolwarm)
    plt.xlabel('x', color = 'red')
    plt.ylabel('y', color = 'red')
    plt.show()
```