Modélisation d'objet et triangulation de Delaunay

Optimisation de l'algorithme , choix des points et application à la reconnaissance faciale

I) **Triangulation de Delaunay**
   1. Définitions et propriétés
   2. Implémentation informatique
      - Algorithme naïf
      - Algorithme par insertion

II) **Application à la reconnaissance faciale**
   1. Principe
   2. Utilisation des coordonnées et des distances
   3. Utilisation des aires des triangles
   4. Résultats

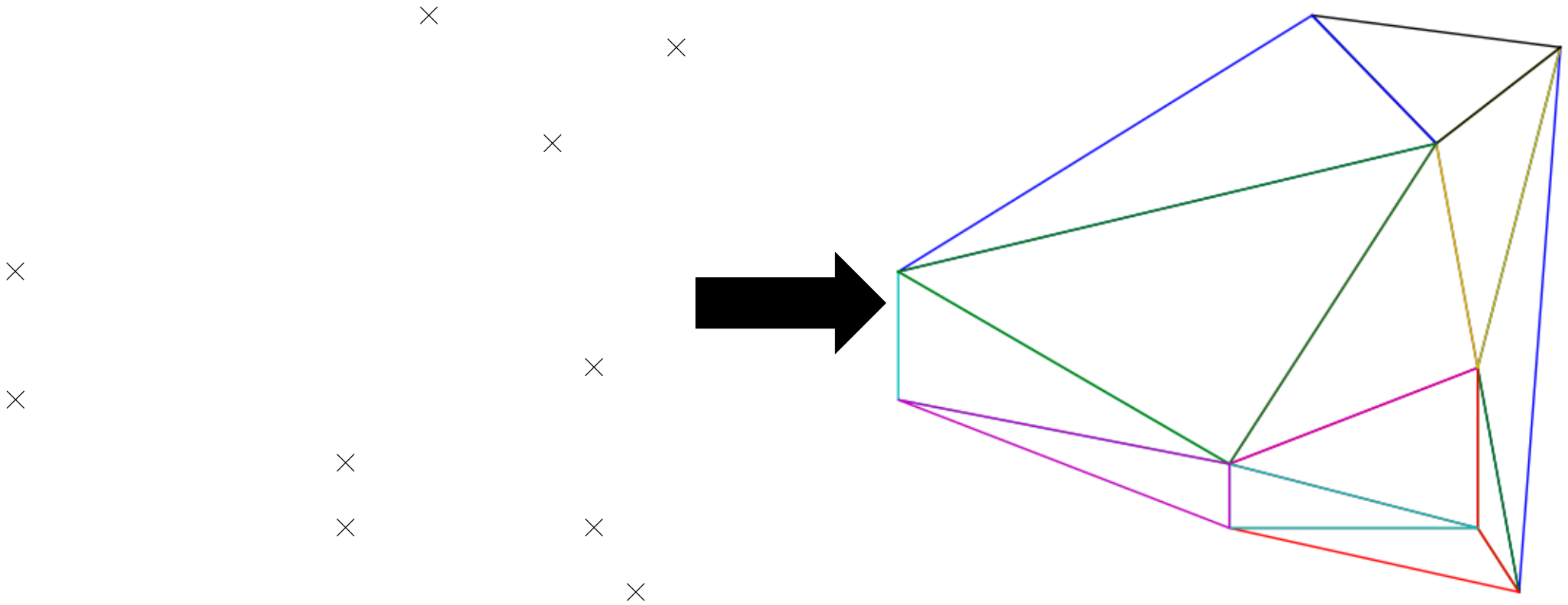
Dans toute la suite on considère P un ensemble fini de points du plan

Définition :

Une triangulation de P est un ensemble de triangles ne se recouvrant pas, dont l'union est l'enveloppe convexe de P.

Définition :

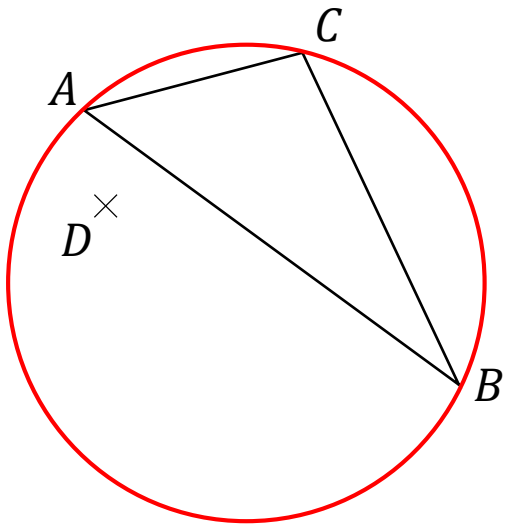
La triangulation de Delaunay de P est une triangulation telle qu'aucun point de P est à l'intérieur du cercle circonscrit d'un des triangles de cette triangulation.



Propriété :

Soit $A(A_x , A_y)$, $B(B_x , B_y)$, $C(C_x , C_y)$ et $D(D_x , D_y)$ quatre points tels que le triangle ABC est direct.

$$D \text{ est à l'intérieur du cercle circonscrit du triangle } ABC \iff \begin{vmatrix} A_x & A_y & A_x^2 + A_y^2 & 1 \\ B_x & B_y & B_x^2 + B_y^2 & 1 \\ C_x & C_y & C_x^2 + D_y^2 & 1 \\ D_x & D_y & D_x^2 + D_y^2 & 1 \end{vmatrix} > 0$$



Algorithme naïf

Principe :

- Créer la liste de tous les triangles possibles avec les points de P
- Tester la propriété de Delaunay pour chaque triangle et chaque point

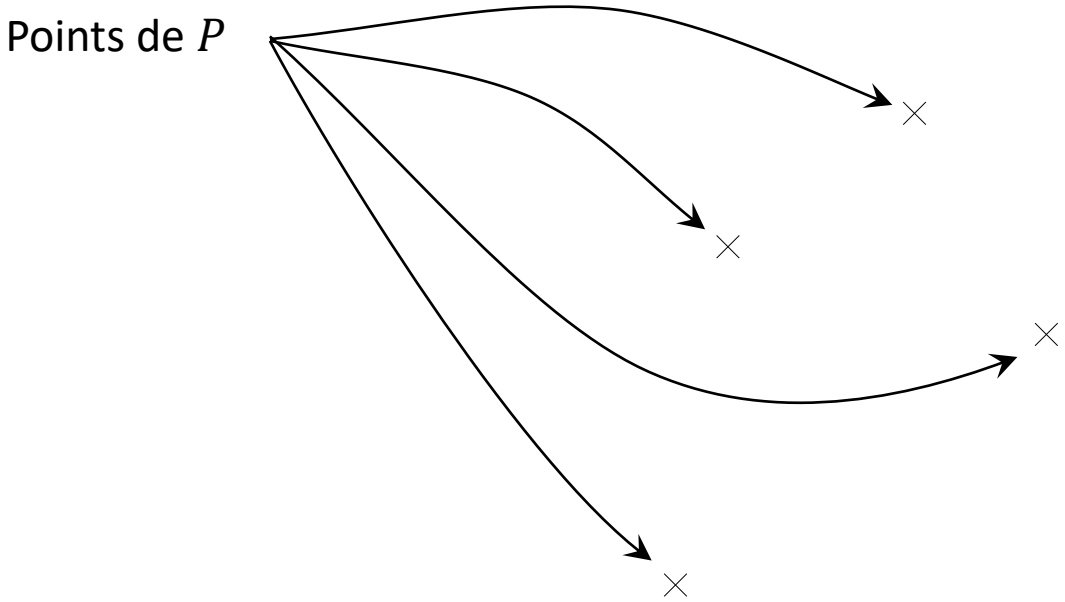
Complexité en $O(n^3)$ où $n = |P|$

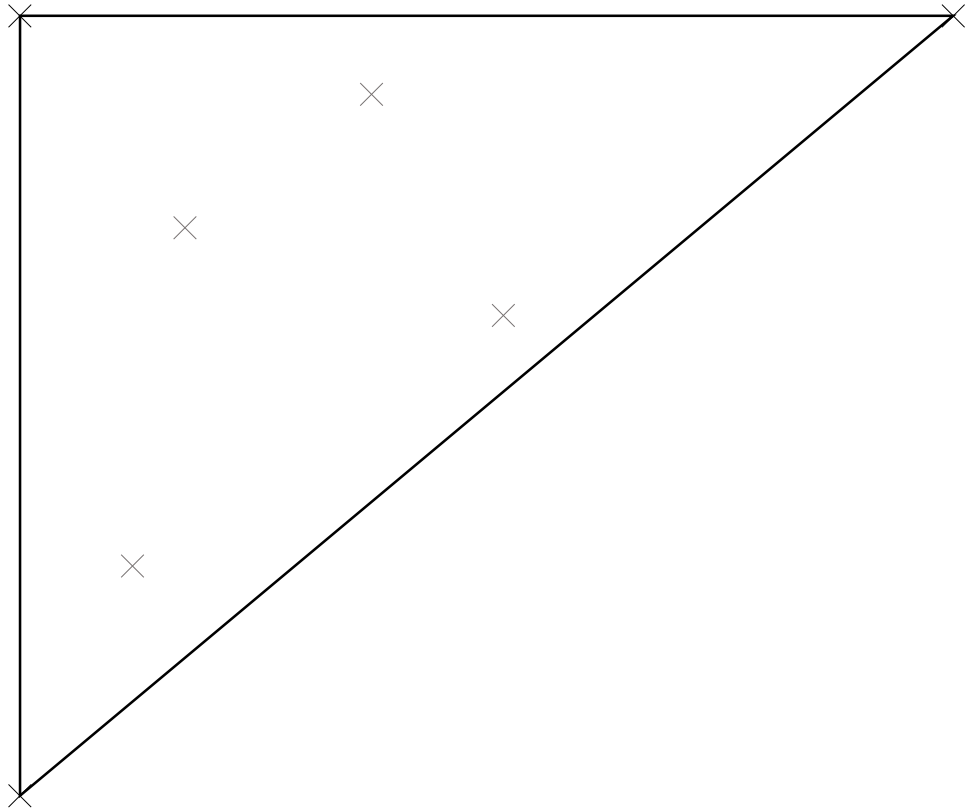
Algorithme par insertion

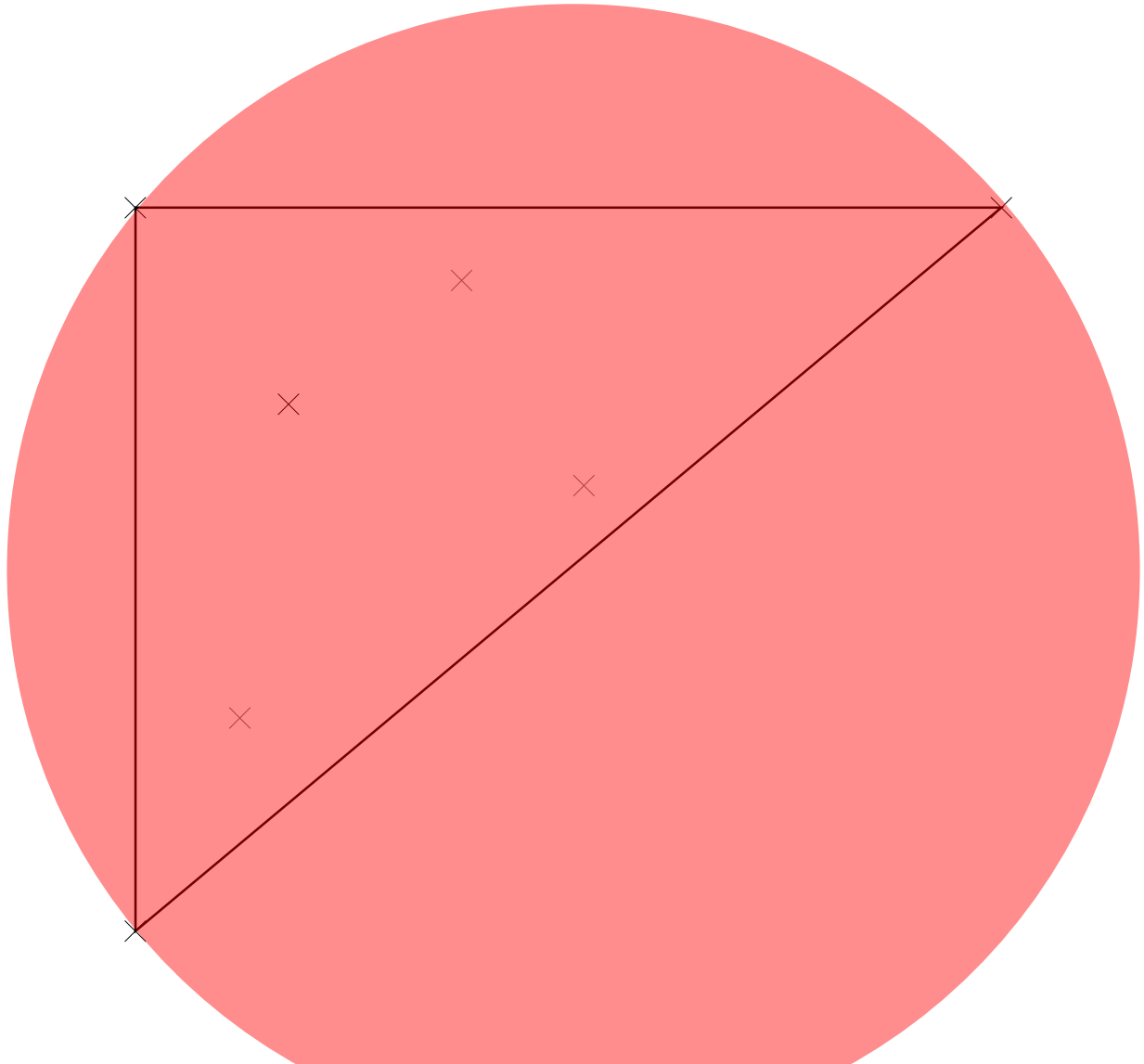
Principe :

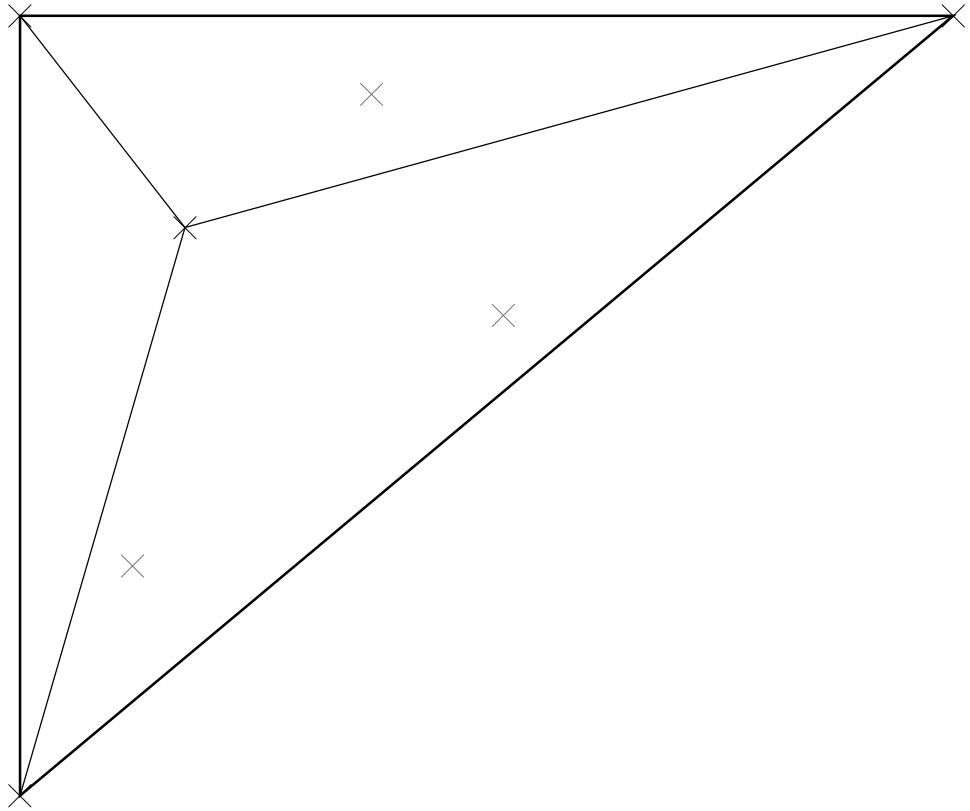
- Poser trois points formant un triangle qui englobe tous les points de P
- Ajouter un point dans la triangulation précédente et en former une nouvelle respectant toujours la propriété de Delaunay
- Répéter l'étape précédente sur tous les points de P
- Retirer tous les triangles utilisant un des points du triangle englobant

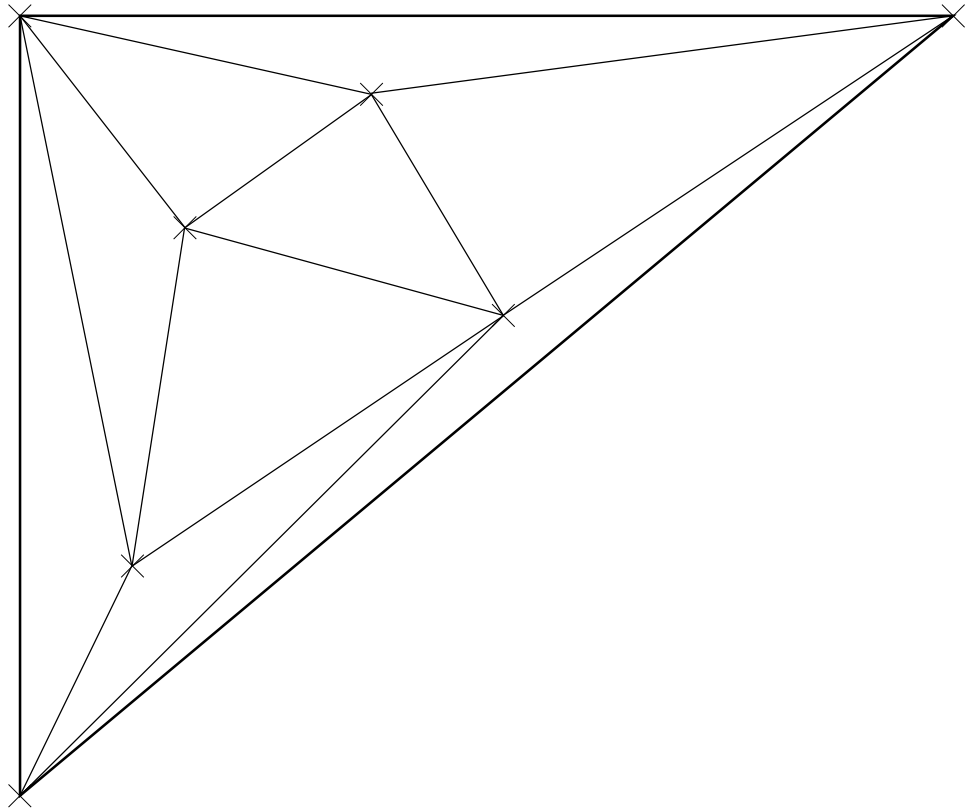
Complexité en $O(n \log(n))$ où $n = |P|$

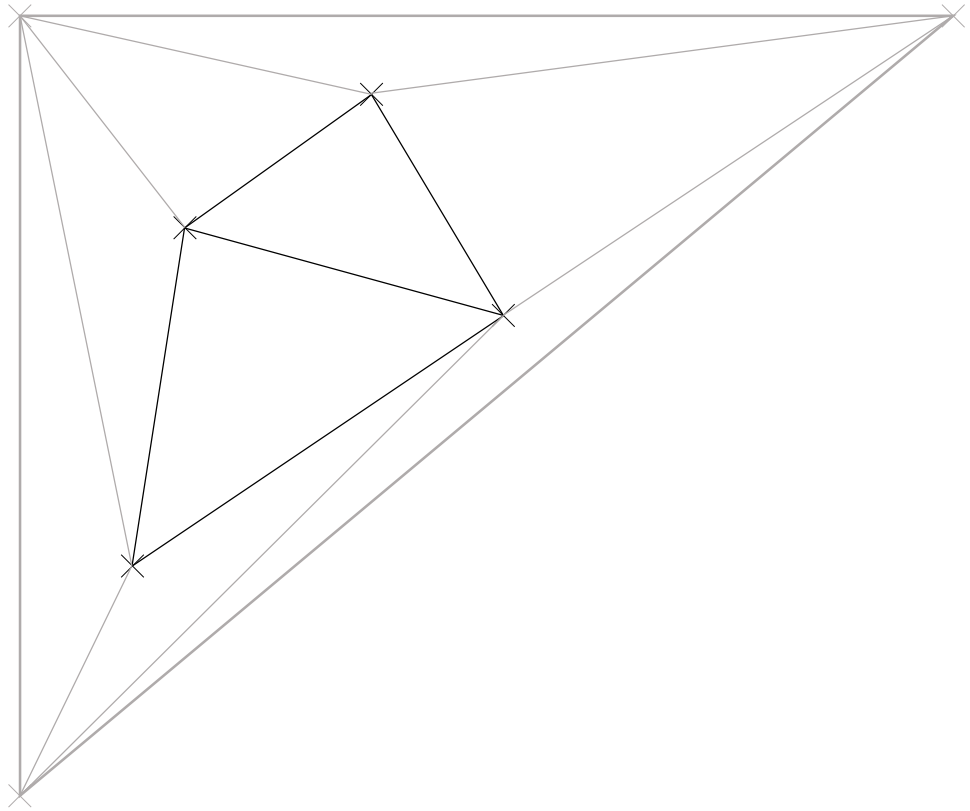

Points de P



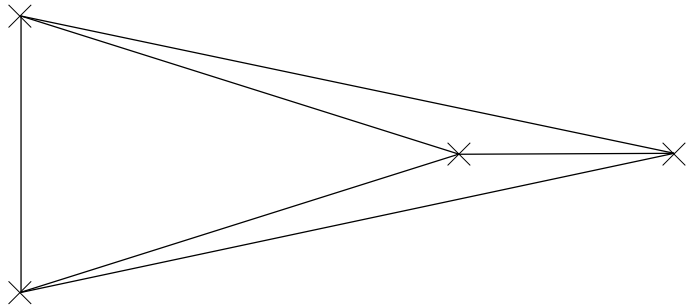


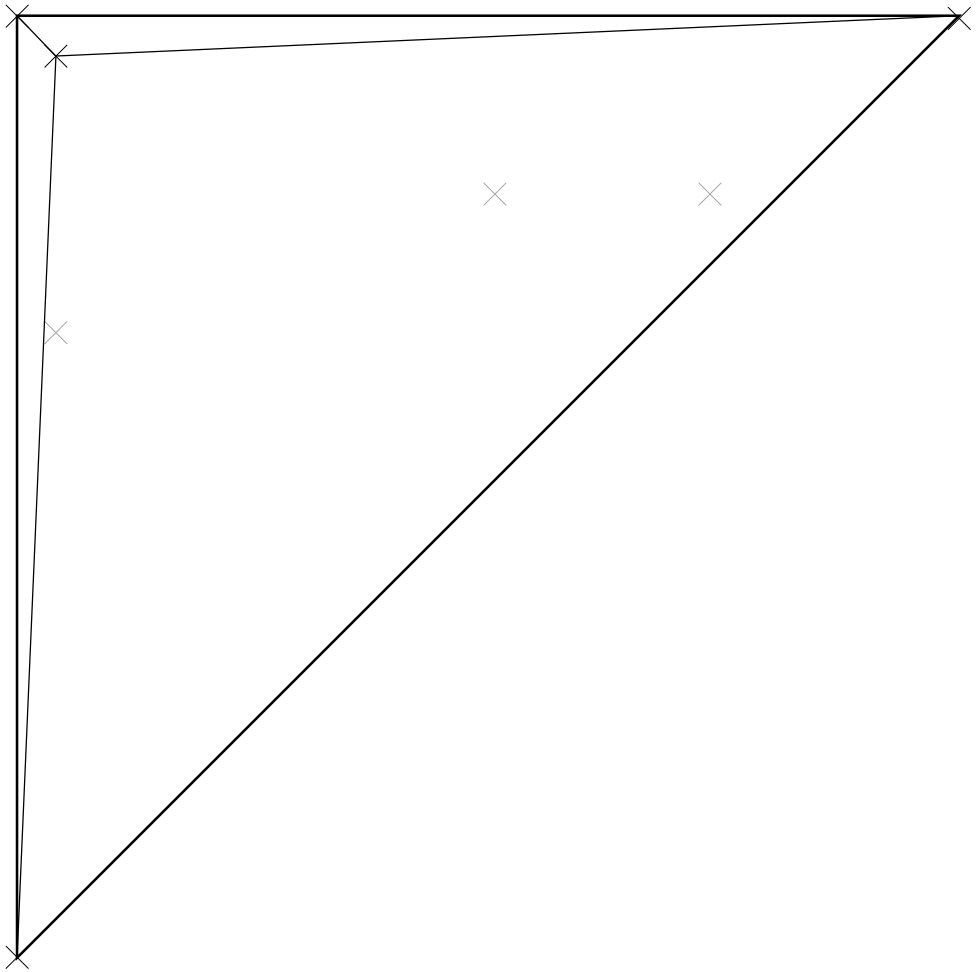


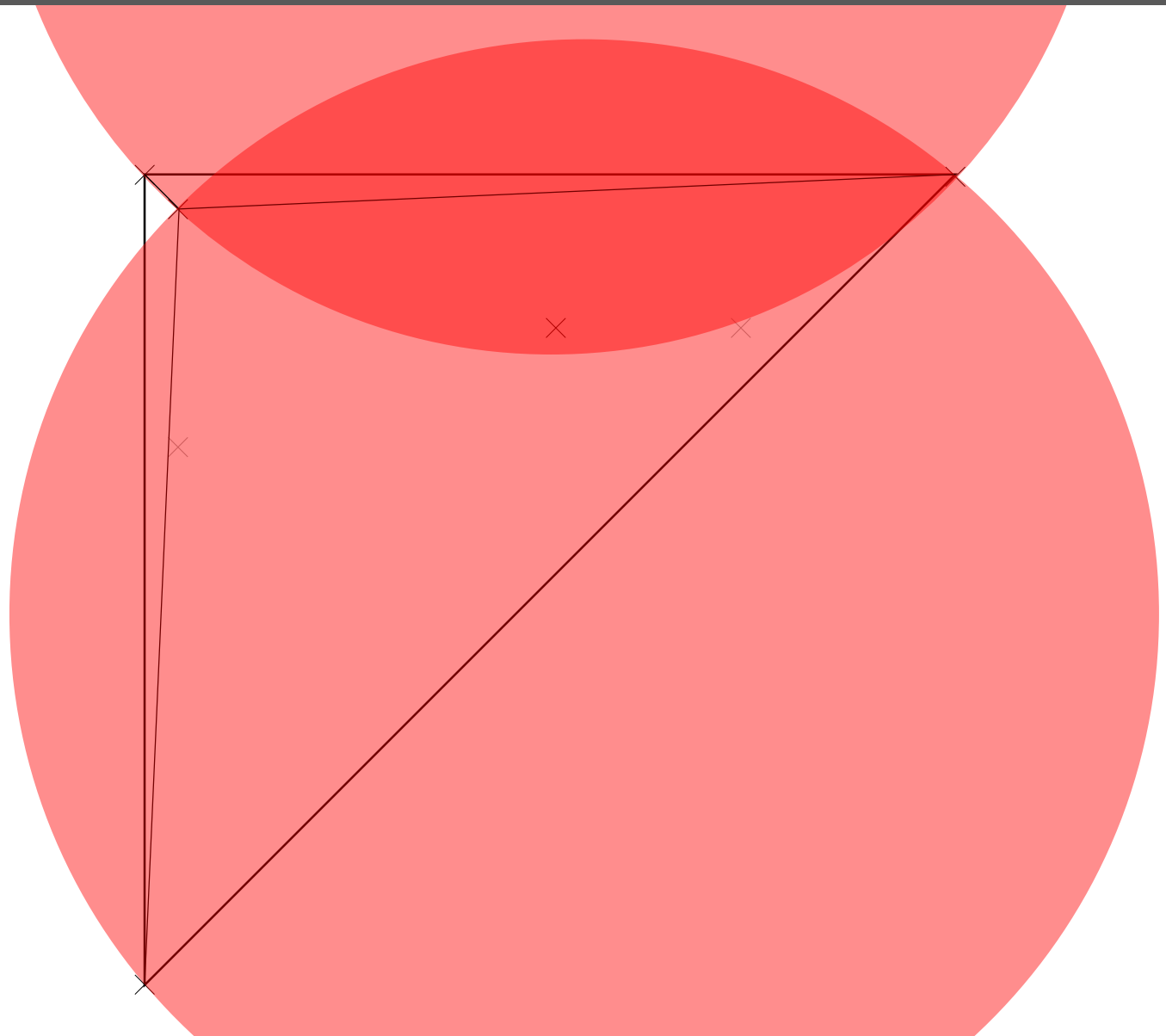


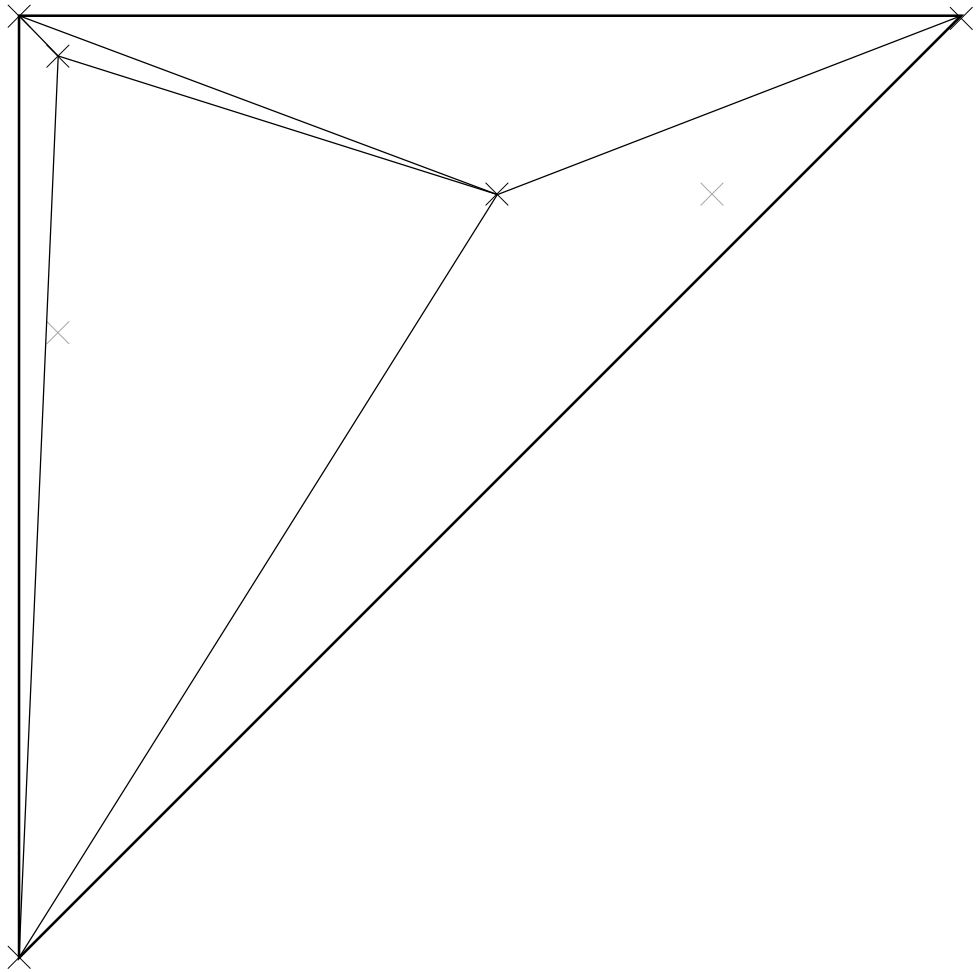


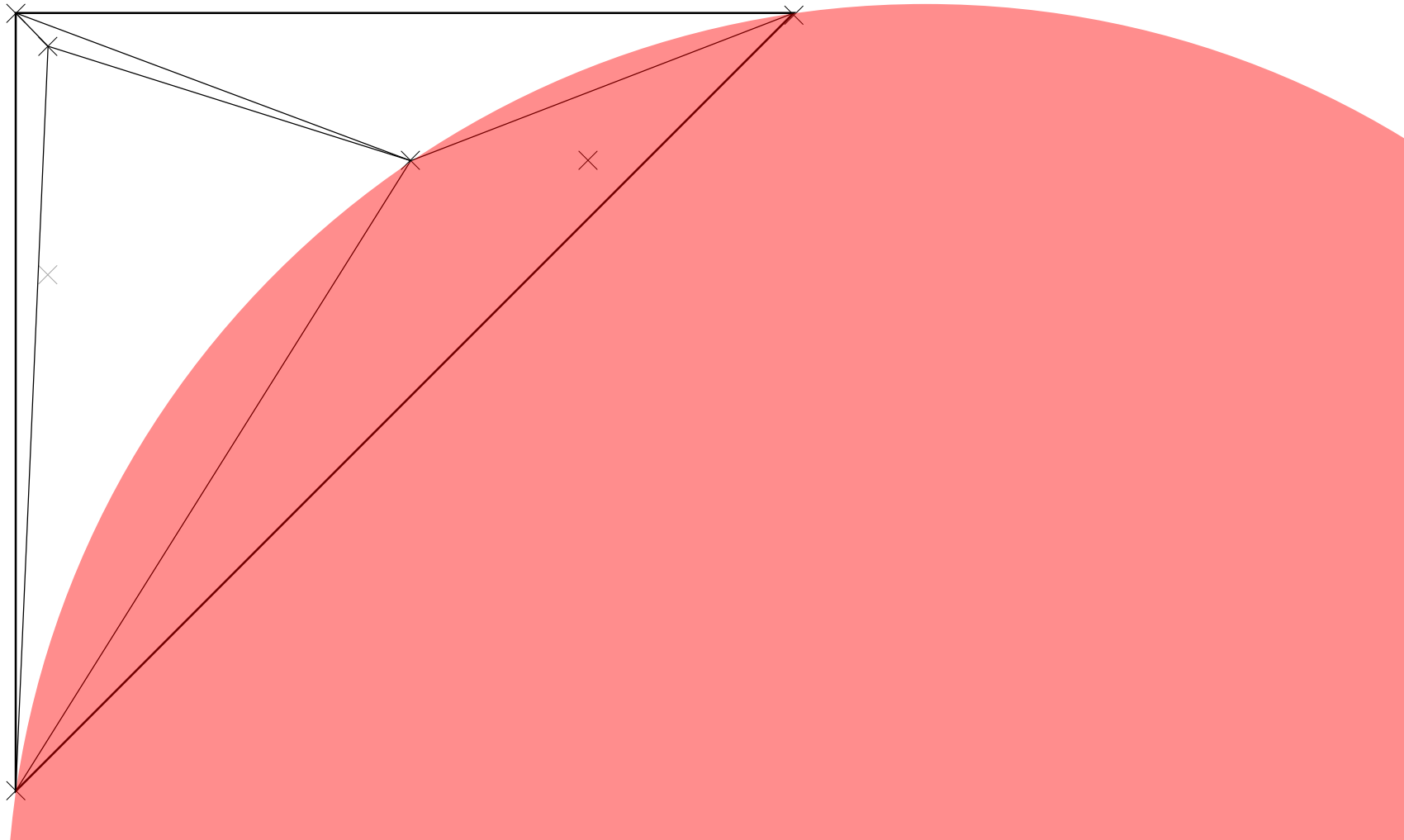
```
def algo_insert (P): #algorithme par insertion utilisant la condition de Delaunay avec le déterminant
  """P : liste des points du plan
  Renvoie la triangulation de Delaunay des points de P"""
  n = len(P)
  mx = min_x(P)
  my = max_y(P)
  pt_hg = (-5 * mx * mx - 100 , 5 * my * my + 100)
  dmax = int(distance_max(P,pt_hg))+1
  pt_hd = (pt_hg[0]+int(np.sqrt(2*dmax*dmax))+1 , pt_hg[1])
  pt_bg = (pt_hg[0] , pt_hg[1]-int(np.sqrt(2*dmax*dmax))-1)
  T = [[pt_hg , pt_bg , pt_hd]]
  for point in P :
    dT = del_triangles(point,T)
    C = contour(dT,T)
    T = retirer(T,dT)
    for segment in C :
      T.append(ajout(segment,point))
  return nettoyage(T,pt_hg,pt_bg,pt_hd)
```

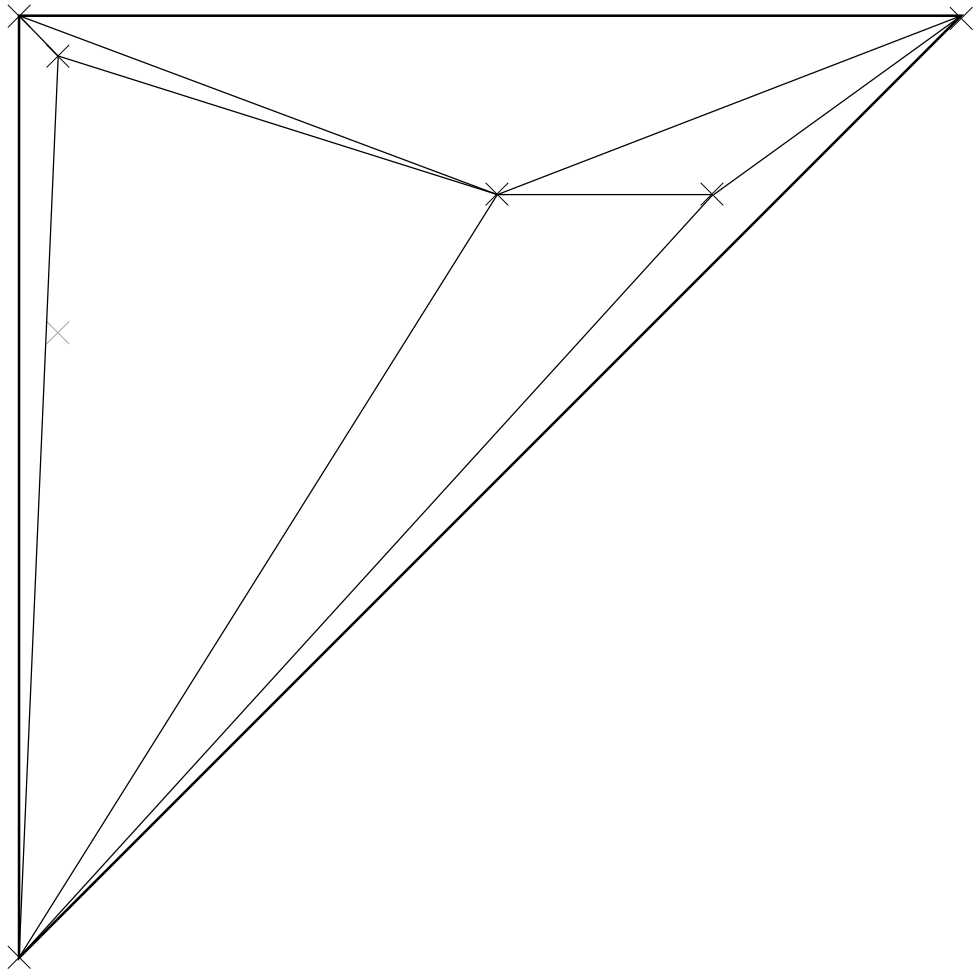


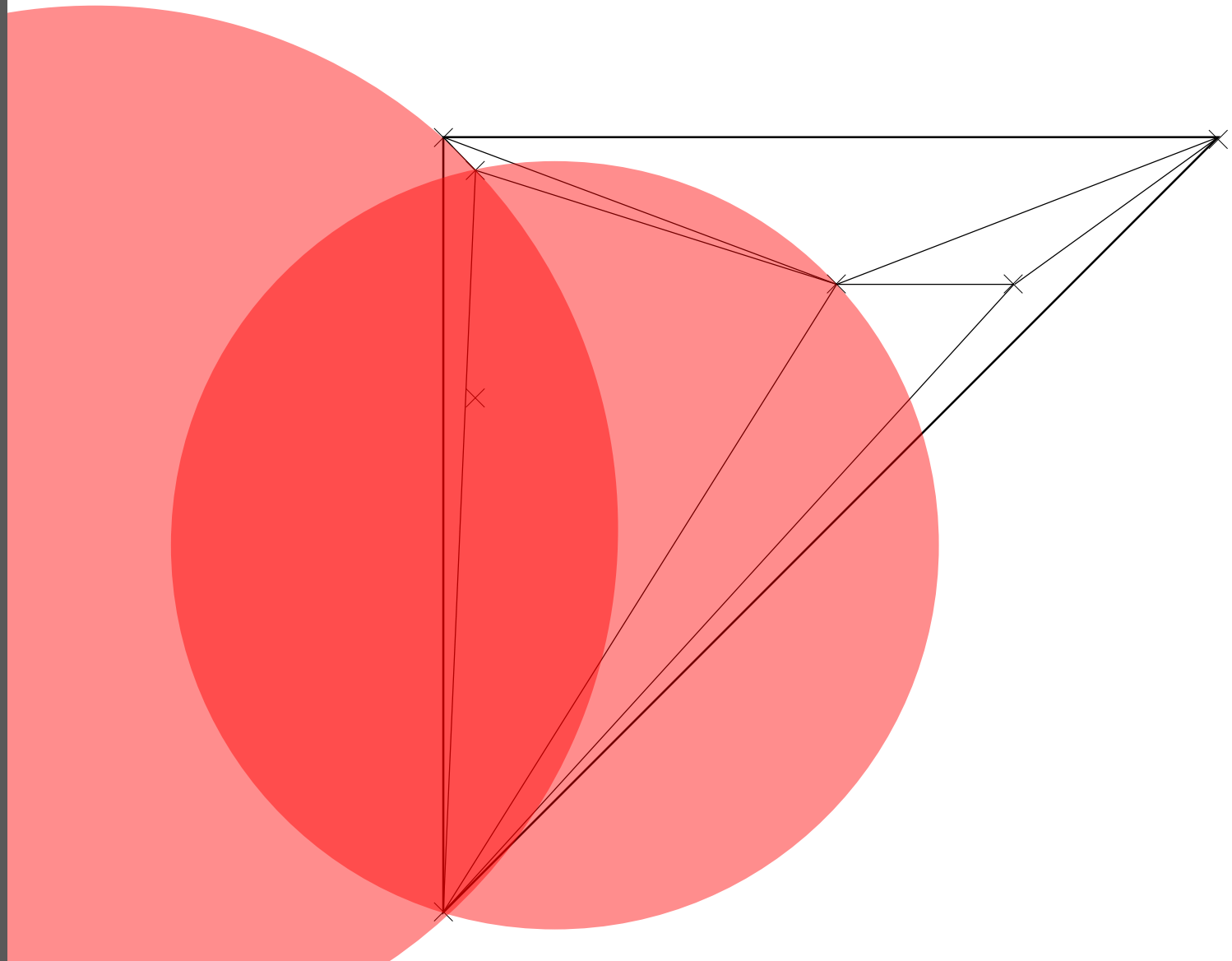


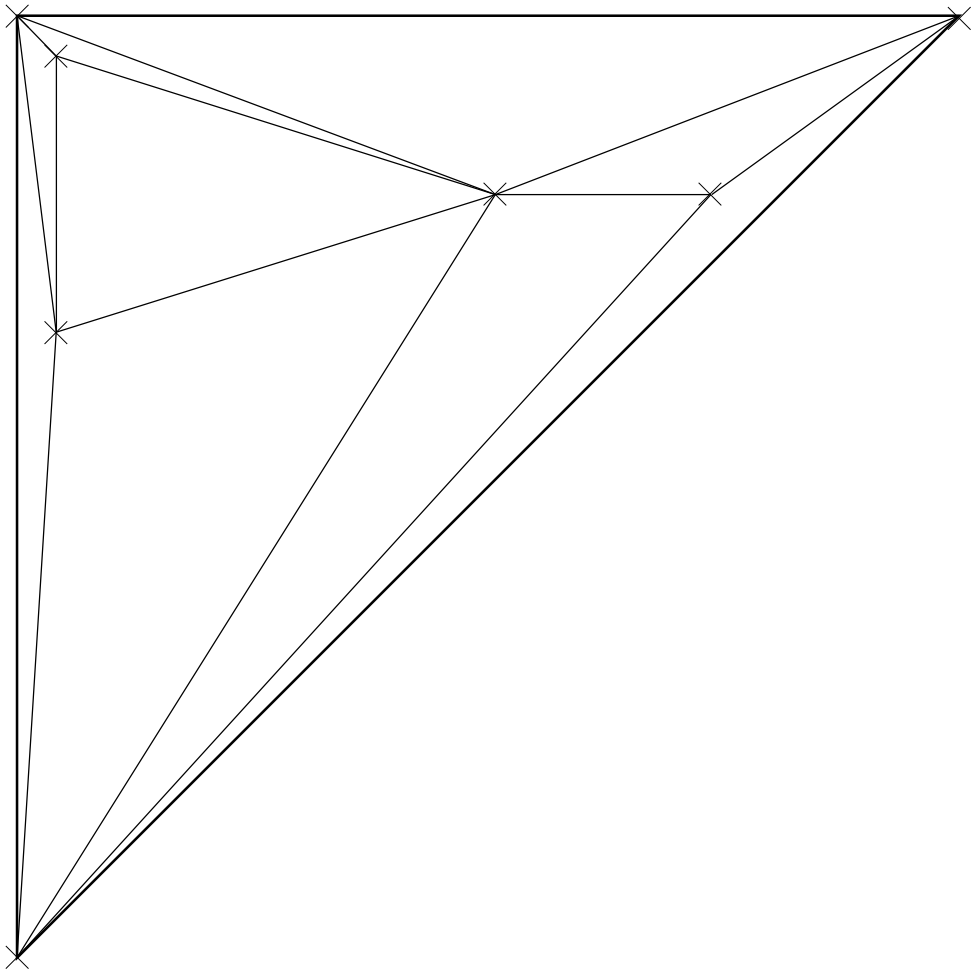


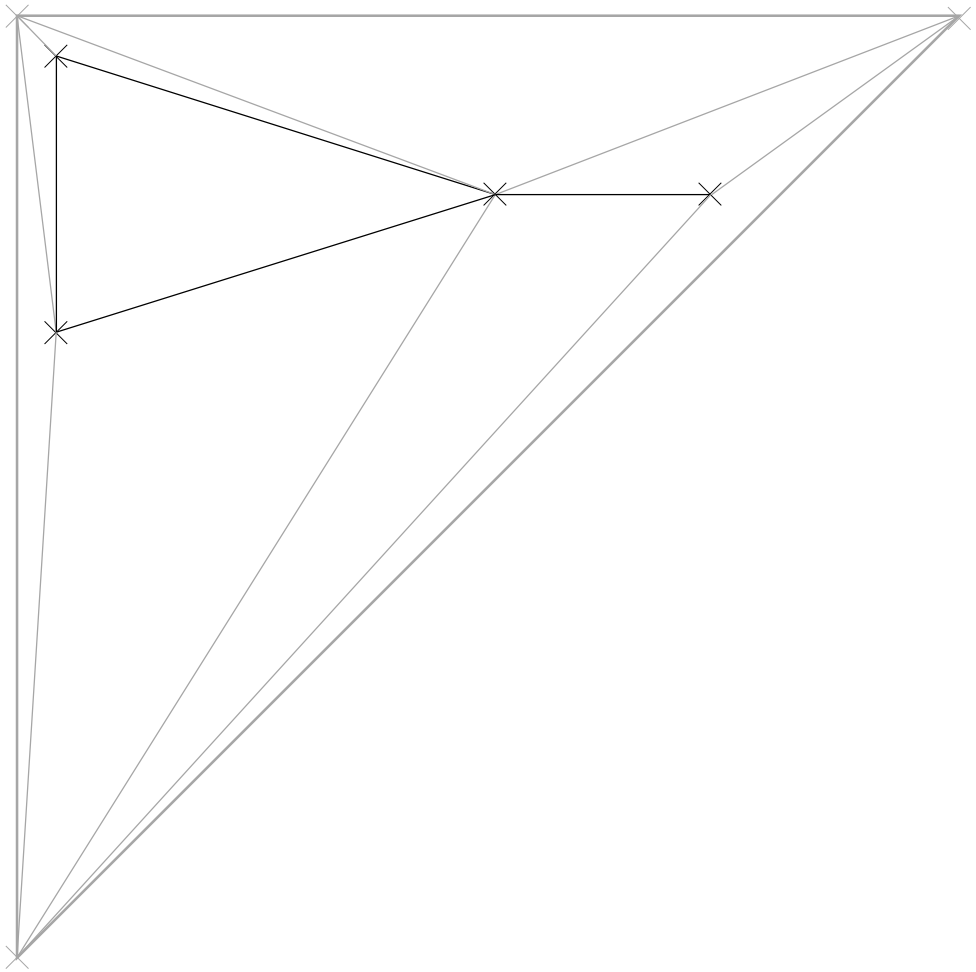










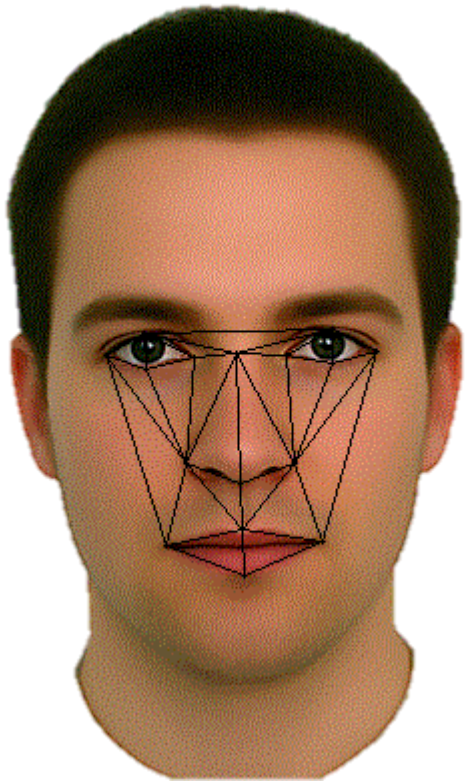


Qu'est-ce que la reconnaissance faciale ?

Reconnaissance faciale = détection + identification du visage

Comment réaliser l'identification ?

Comparer le visage avec une base de données

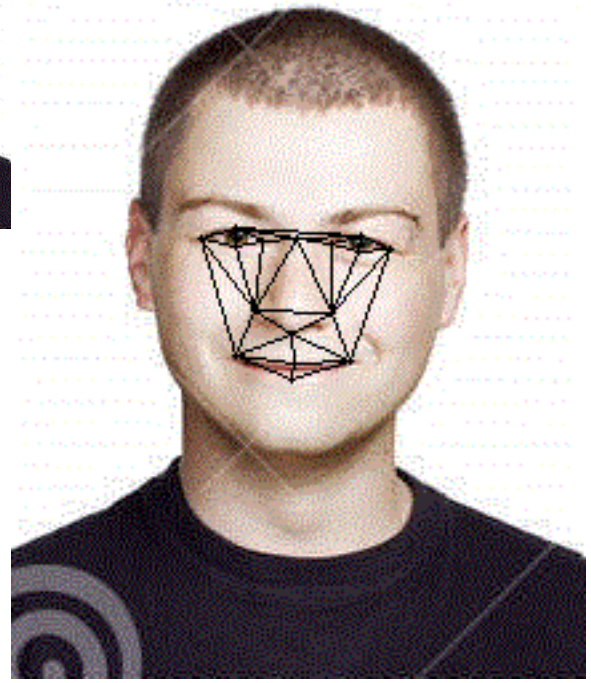


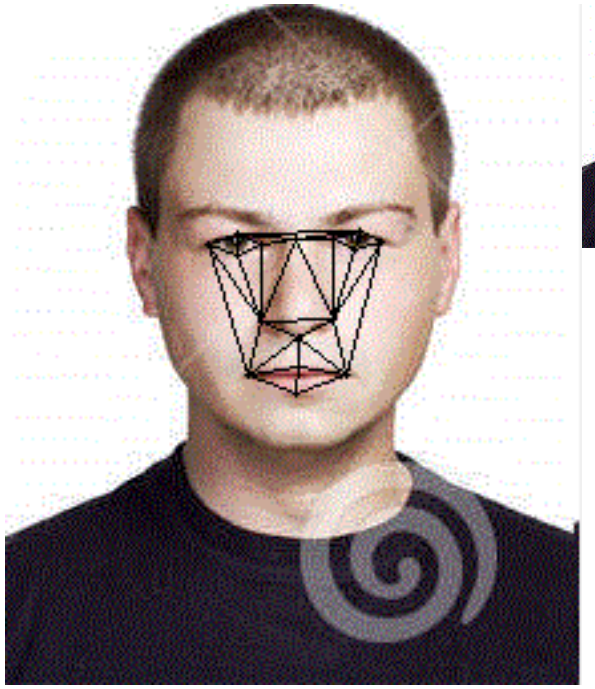
Quelles données compare-t-on ?

- les distance entre des points du visage
- les aires des triangles

Quels points utilise-t-on ?

Contour des différentes structures du visage





```
"""Paramètres de bases"""
#4 pts par oeil
#4 pts nez
#4 pts bouches

def compare_pts(p1,p2):
    """ Compare les deux ensembles de points qui sont supposés comprendre le même nombre d'éléments"""
    n = len(p1)
    L = [0]*n #1 si les points concordent ; 0 sinon
    for k in range(n):
        if (abs(p1[k][0]-p2[k][0]) < dx) and (abs(p1[k][1]-p2[k][1]) < dy):
            L[k] = 1
    return L

def correspondance_naive(p1,p2):
    """ Calcul du pourcentage de correspondance points par points"""
    compt = 0
    L = compare_pts(p1,p2)
    n = len(p1)
    for k in range(n):
        compt += L[k]
    return (compt / n *100) #pourcentage

def correspondance_structure(p1,p2):
    """ Calcul du pourcentage de correspondance avec bonus par structures """
    L = compare_pts(p1,p2)
    n = len(p1)
    bonus = 0
    compt_y = 0
    compt_n = 0
    compt_b = 0
    for k in range(8):
        compt_y += L[k]
    for k in range(8,12):
        compt_n += L[k]
    for k in range(12,n):
        compt_b += L[k]
    corres_y = (compt_y / n *100)
    if corres_y == 100:
        bonus += 10
    corres_n = (compt_n / n *100)
    if corres_n == 100:
        bonus += 10
    corres_b = (compt_b / n *100)
    if corres_b == 100:
        bonus += 10
    return ((compt_y + compt_n + compt_b) / n *100) + bonus
```

```
def distance_carac(p):
    """ Calcul de distances caractéristiques du visage """
    #Liste non-exhaustive
    n = len(p)
    ecart_oeil = ((p[2][0]-p[4][0])**2+(p[2][1]-p[4][1])**2)**(1/2)
    incer_o = (1/2)*(abs(p[2][0]-p[4][0])*dx + abs(p[2][1]-p[4][1])*dy)/ecart_oeil
    larg_nez = ((p[9][0]-p[11][0])**2+(p[9][1]-p[11][1])**2)**(1/2)
    incer_n = (1/2)*(abs(p[9][0]-p[11][0])*dx + abs(p[9][1]-p[11][1])*dy)/larg_nez
    comm_levre = ((p[12][0]-p[14][0])**2+(p[12][1]-p[14][1])**2)**(1/2)
    incer_l = (1/2)*(abs(p[12][0]-p[14][0])*dx + abs(p[12][1]-p[14][1])*dy)/comm_levre
    return ([ecart_oeil , larg_nez , comm_levre],[incer_o,incer_n,incer_l])



def corres_dist(p1,p2):
    """Calcul de la correspondance des distances caractéristiques """
    d1,Dd1 = distance_carac(p1)
    d2,Dd2 = distance_carac(p2)
    nd = len(d1)
    compt = 0
    for k in range(nd):
        if abs(d1[k] - d2[k]) < (1/3*(Dd1[k]+Dd2[k])):
            compt += 1
            print(abs(d1[k] - d2[k]))
    return (compt / nd *100)
```

Propriété :

Soit $\mathcal{A}$ l'aire d'un triangle ABC.

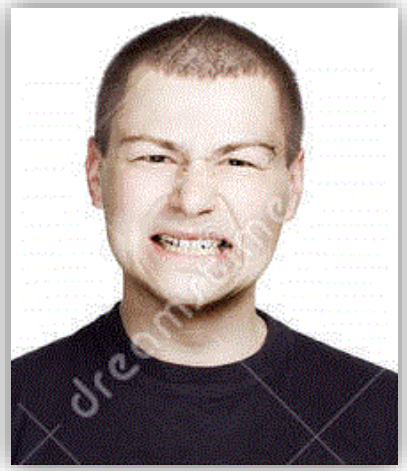
$$\mathcal{A} = \frac{1}{2} \times \left|\det_{\mathcal{B}}\left(\overrightarrow{AB}, \overrightarrow{AC}\right)\right|$$

```
def aire_total(T):
    """Calcul de l'aire totale du polygone"""
    n = len(T)
    A = 0
    for k in range(n):
        A += (1/2)*abs((T[k][1][0]-T[k][0][0])*(T[k][2][1]-T[k][0][1])-(T[k][1][1]-T[k][0][1])*(T[k][2][0]-T[k][0][0]))
    return A

def ecart_aire(t1,t2):
    """Calcul de la différence d'aires des polygones issus des deux triangulations"""
    a1 = aire_total(t1)
    a2 = aire_total(t2)
    return abs(a1-a2)/a1*100
```

Visage 1

Visage 2



Visage 3

Visage 4

Comparaison	Points	Points + bonus	Distances	Aires	
Visages 1/2	94%	94%	33%	95%	Erreur 10 en x et y
Visages 1/3	75%	75%	33%	97%	
Visages 2/3	38%	38%	67%	98%	
Visages 1/4	25%	25%	67%	71%	
Visages 1/2	94%	94%	33%	95%	Erreur 7 en x et y
Visages 1/3	19%	19%	33%	97%	
Visages 2/3	6%	6%	33%	98%	
Visages 1/4	6%	6%	67%	71%	