1 Python correction

```python
from skimage.io import imread # read input image
import numpy as np
import matplotlib.pyplot as plt

from scipy.spatial import Delaunay # Delaunay triangulation
```

1.1 Point pattern

The image is first loaded.

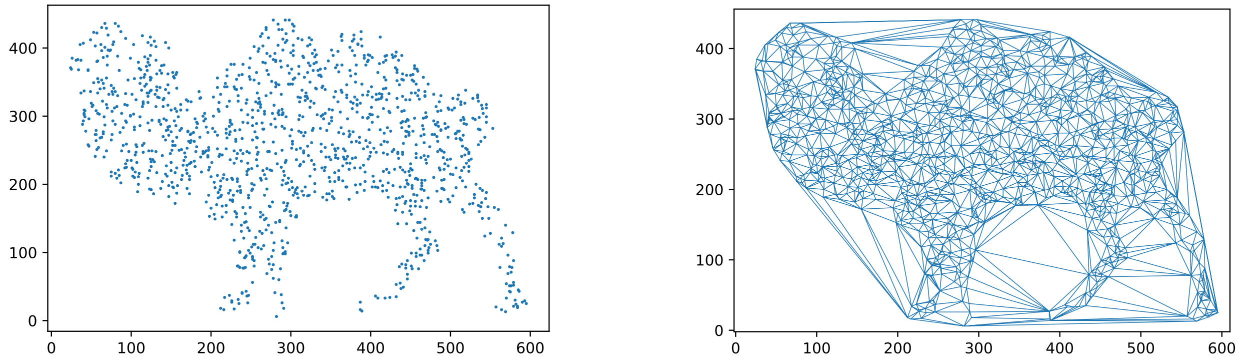
```python
A = imread('camel.png')
m,n = A.shape
```

All coordinates of pixels constituting the shape are extracted. The following code mainly consist of array manipulation.

```python
pts = np.where(A)
pts = np.array(pts).transpose()

indices = np.arange(len(pts))
np.random.shuffle(indices)

# Pay attention to reference: points and image have not the same
    ↪ coordinates
pts = pts[indices]
pts = np.fliplr(pts)
pts[:,1] = m - pts[:,1]
```

Then, given a certain density, points are randomly chosen in the shape. They are displayed in Fig.1.

```python
# Generate points
density =.01
nbPoints = int(len(pts)*density)

points = pts[:nbPoints]
plt.scatter(*zip(*points), s=1)
plt.savefig("points.pdf", bbox_inches='tight')
plt.show()
```



(a) Set of points, uniformely chosen in the shape.

(b) Delaunay triangulation.

Figure 1: Set of points and its Delaunay triangulation.

1.2 Delaunay triangulation

The Delaunay triangulation is simply obtained by the following code. The result is presented in Fig.1.

```python
tri = Delaunay(points)

# Display result
plt.triplot(points[:,0], points[:,1], tri.simplices, lw=.5)
plt.show()
```

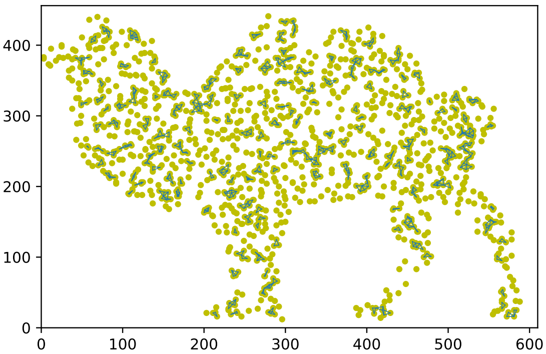
1.3 Alpha-solid

In order to build the alpha-solid, the circum-radii of all triangles should be computed. A rather simple way to do this is to use the class Triangle of sympy.geometry. The use of progressbar displays a progress bar, as the computation might take a long time. The sympy module is a symbolic computation module, and does not an optimal algorithm for this task.

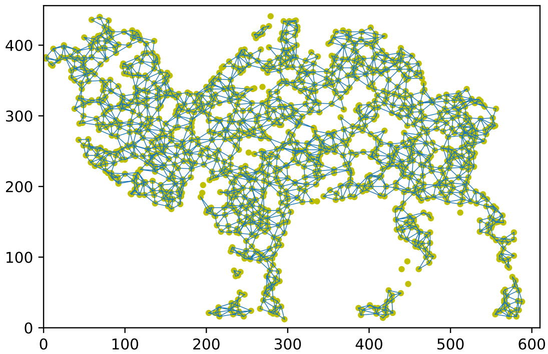
```
from sympy.geometry import Triangle
radius=[]
import progressbar
count=0

# takes a long time because of symbolic computation
with progressbar.ProgressBar(max_value=len(tri.simplices)) as bar:
    for t in tri.simplices:
        count+=1
        tt = Triangle(points[t[0], :], points[t[1], :], points[t[2], :] )
        radius.append(tt.circumradius)
        bar.update(count)
```

Then, given a radius, one can filter the triangles. The results are presented in Fig.2.

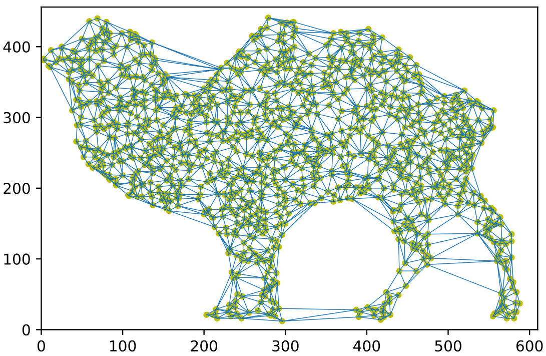
```
 or R in progressbar.progressbar([5,10, 50, 100, 100000]):

    r = np.array(radius)<R
    fig = plt.figure()
    plt.triplot(points[:,0], points[:,1], tri.simplices[r], lw=.5)
    plt.scatter(points[:,0], points[:,1], c='y', s=10)
    plt.show()
```



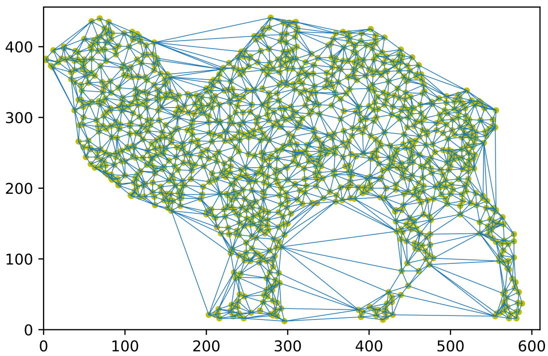
(a) $\alpha = 1/5$.



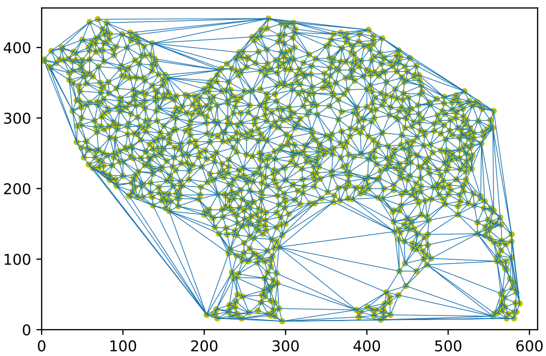
(b) $\alpha = 1/10$.



(c) $\alpha = 1/50$.



(d) $\alpha = 1/100$.



(e) $\alpha = 1/100000$.

Figure 2: Different alpha-solids.