Triangulos2

November 30, 2019

1 Segmentación de triángulos

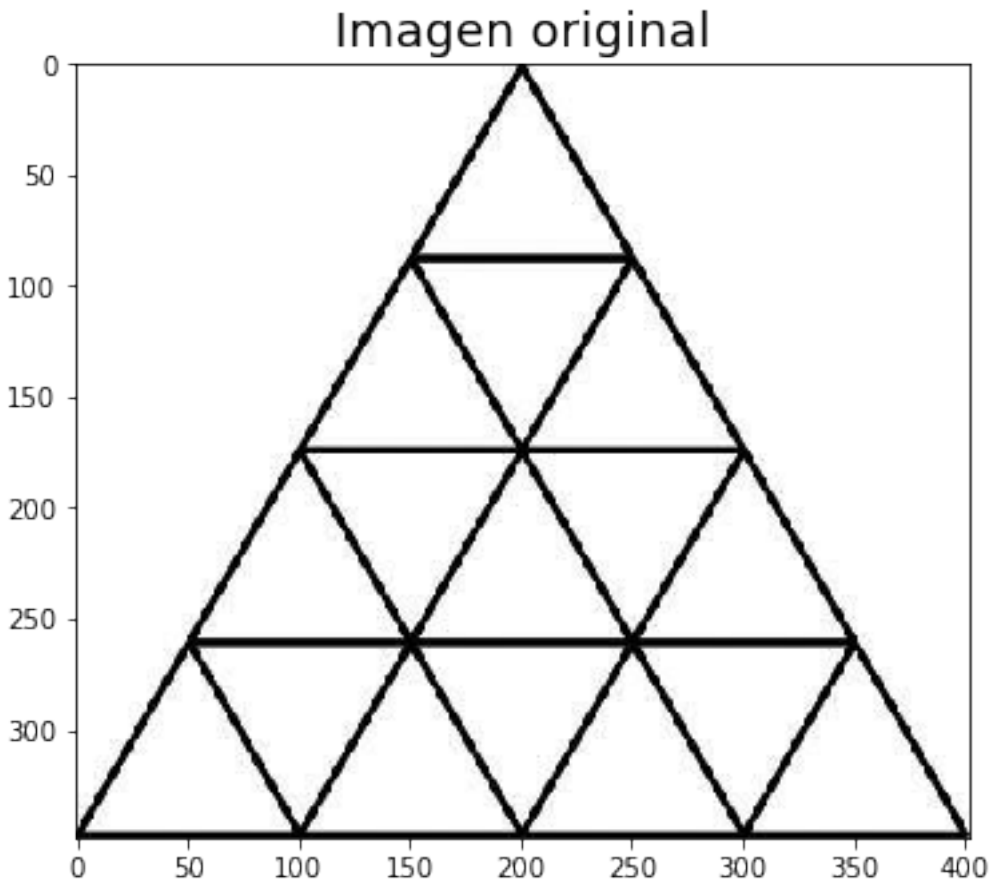
1.0.1 Alumnos: Salma Patricia Gutiérrez Rivera y Gustavo Magaña López

Realice los cálculos para la imagen triangulos2.jpg. Haga una función que calcule cuantos triángulos aparecen en la figura. Haga una imagen para cada triángulo encontrado. **1.** Segmentar las aristas de los triángulos. En los resultados, mostrar la imagen binaria de las aristas.

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv
import math
import skimage
from skimage.measure import label, regionprops
from skimage.color import label2rgb
from itertools import combinations
from skimage.draw import polygon
```

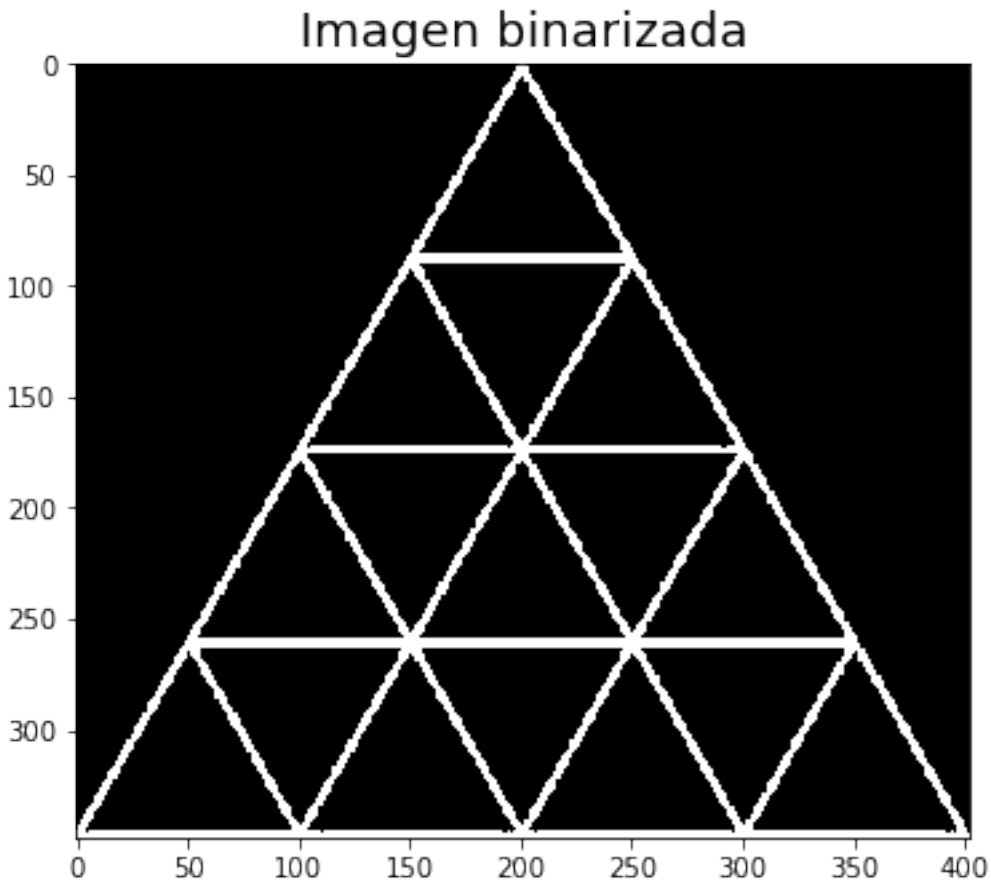
[2]:
```
triangulos = cv.imread("images/triangulos2.jpg", 0)
```

[3]:
```
fig = plt.figure(figsize = (6, 6))
fig.add_subplot(1, 1, 1)
plt.imshow(triangulos, cmap = 'gray');
plt.title("Imagen original", size = 18);
```



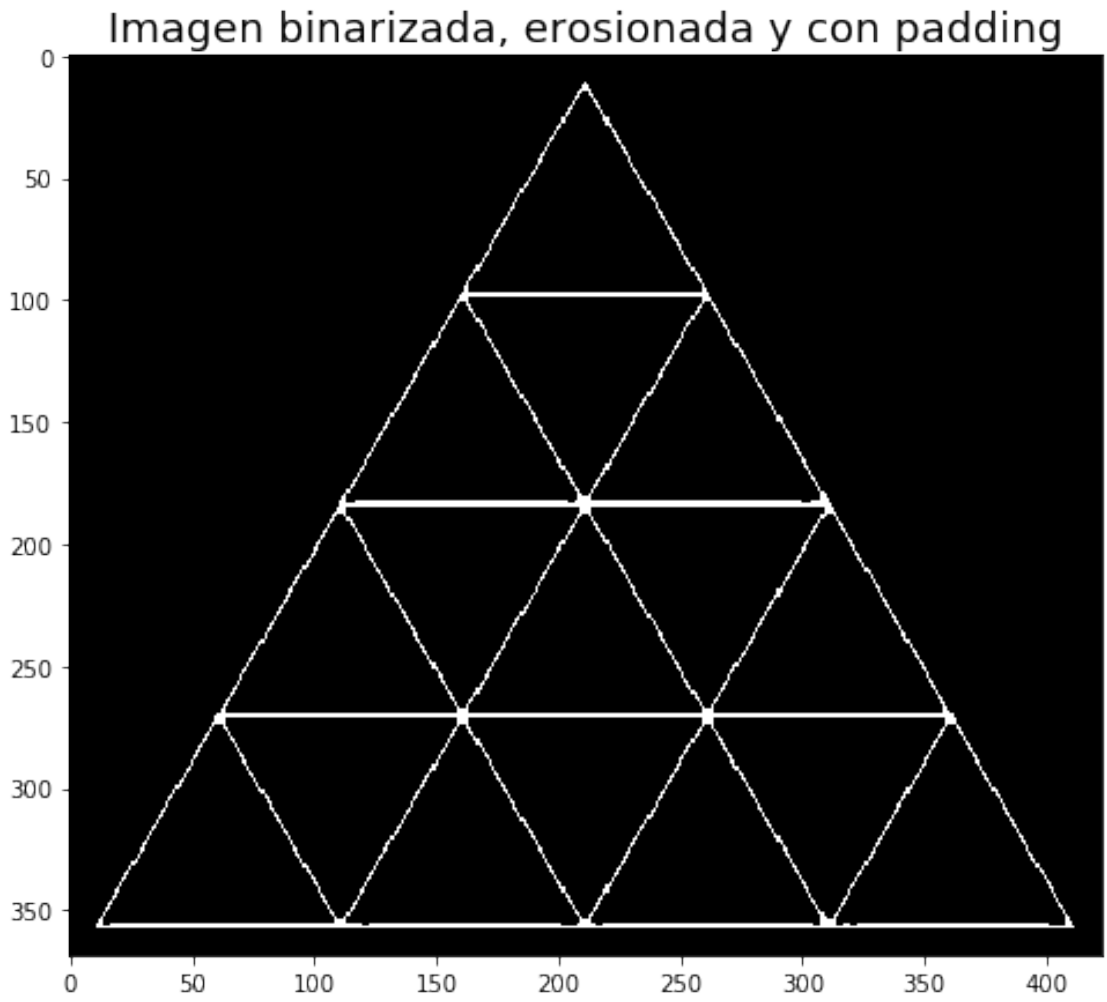
[4]:
```
bin_triang = np.where(triangulos < 170, 255, 0)
```

[5]:
```
fig = plt.figure(figsize = (6, 6))
fig.add_subplot(1, 1, 1)
plt.imshow(bin_triang, cmap = 'gray');
plt.title("Imagen binarizada", size = 18);
```



[6]:
```
padd_bin_triang = cv.copyMakeBorder(bin_triang, 10, 10, 10, 10, cv.
↪BORDER_CONSTANT)
imgu8 = np.uint8(padd_bin_triang)
kernel3 = np.ones((3, 3), np.uint8)
contor = cv.erode(imgu8, kernel3, iterations = 1)
```

[7]:
```
fig = plt.figure(figsize = (8, 8))
fig.add_subplot(1, 1, 1)
plt.imshow(contor, cmap = 'gray');
plt.title("Imagen binarizada, erosionada y con padding", size = 18);
```

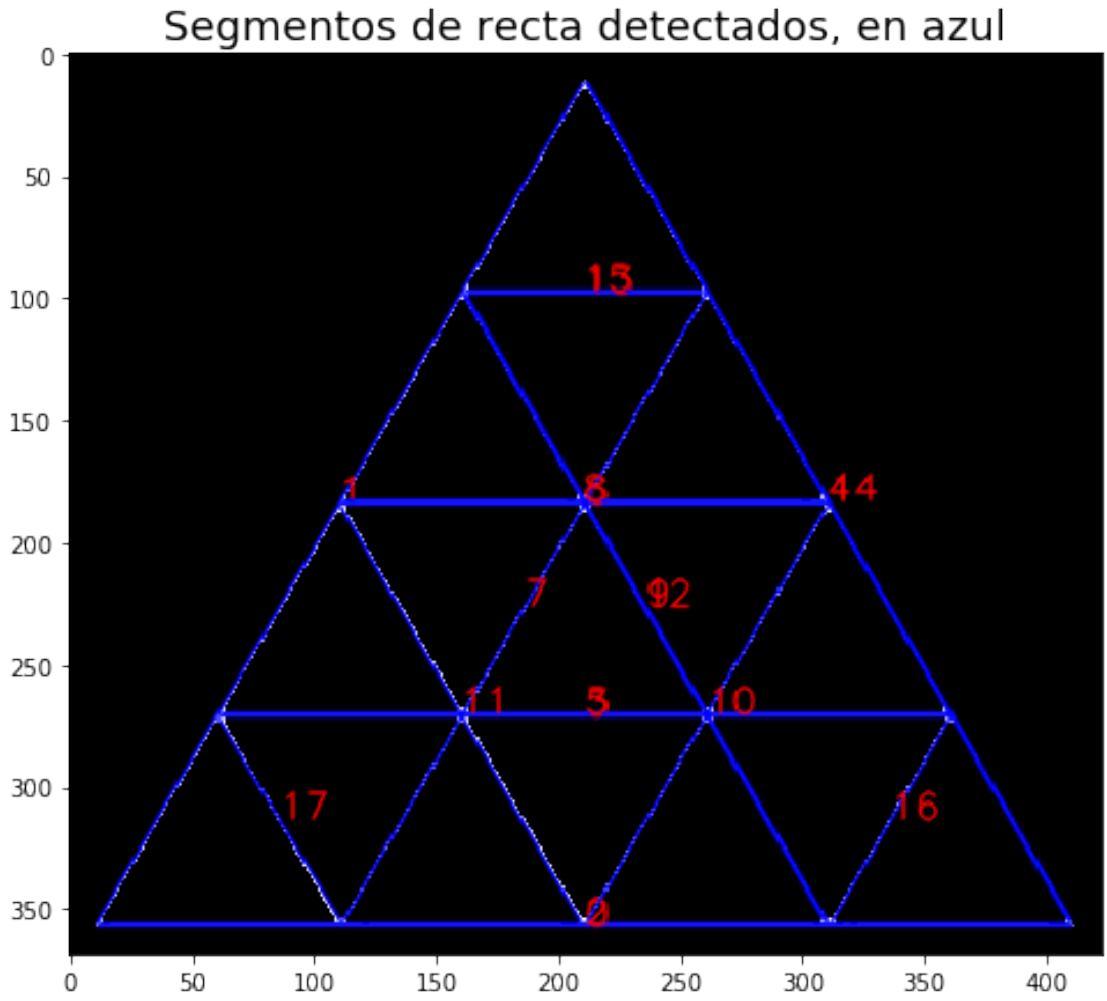


2. Encontrar los modelos de las rectas que forman cada arista. Para esto, solicite un umbral que elimine a las posibles aristas de longitud menor a dicho umbral. Los parámetros que debe encontrar de cada arista son θ, ρ y longitud. Hacer una tabla donde cada renglón sea una arista (identifíquelas con un número) y las columnas sean los parámetros indicados.

[8]:
```
img_con_lineasp = cv.cvtColor(contor, cv.COLOR_GRAY2BGR)
lineas_p = cv.HoughLinesP(contor, 1, np.pi / 180, 45, None, 50, 7)
font = cv.FONT_HERSHEY_SIMPLEX
fontScale = 0.5
color = (255, 0, 0)
thickness = 1

for i in range(len(lineas_p)):
    l = lineas_p[i][0]
    cv.line(img_con_lineasp, (l[0], l[1]), (l[2], l[3]), (0,0,255), 1, cv.
 ↪LINE_AA)
```

```
    org = ((l[0] + l[2]) // 2, (l[1] + l[3]) // 2)
    img_con_lineasp = cv.putText(img_con_lineasp, str(i), org, font, fontScale,␣
↪color, thickness, cv.LINE_AA)
```

[9]:
```
fig = plt.figure(figsize = (8, 8))
fig.add_subplot(1, 1, 1)
plt.imshow(img_con_lineasp);
plt.title("Segmentos de recta detectados, en azul", size = 18);
```



[10]:
```
print("Se detectaron", lineas_p.shape[0], "segmentos de recta")
```

```
Se detectaron 18 segmentos de recta
```

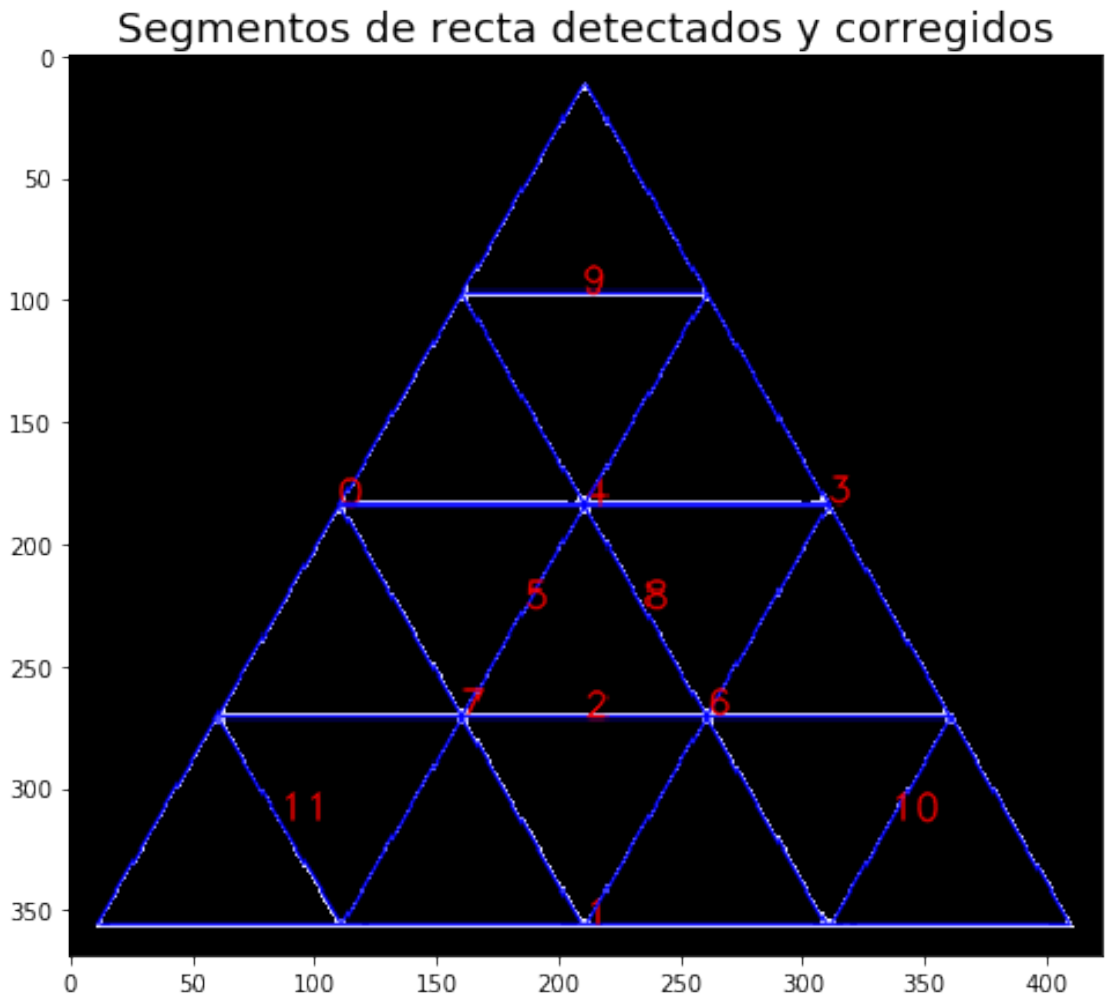
Dibujando recta por recta, encuentro que 0, 5, 8, 9, 13 y 14 son redundantes. Corrijo a mano.

[11]:
```
lineas_p = np.delete(lineas_p, 14, 0);
lineas_p = np.delete(lineas_p, 13, 0);
lineas_p = np.delete(lineas_p, 9, 0);
lineas_p = np.delete(lineas_p, 8, 0);
lineas_p = np.delete(lineas_p, 5, 0);
lineas_p = np.delete(lineas_p, 0, 0);
```

[12]:
```
ep = 1.0e-6

def obtener_houghparams(puntos):
    x1 = puntos[0]
    y1 = puntos[1]
    x2 = puntos[2]
    y2 = puntos[3]

    theta = math.atan((x1 - x2) / (y2 - y1 + ep))
    rho = x1 * math.cos(theta) + y1 * math.sin(theta)
    longitud = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

    return [round(theta * 180.0 / math.pi), round(rho), round(longitud)]
```

[13]:
```
img_con_lineasp = cv.cvtColor(contor, cv.COLOR_GRAY2BGR)
hough_params = []

for i in range(len(lineas_p)):
    l = lineas_p[i][0]
    cv.line(img_con_lineasp, (l[0], l[1]), (l[2], l[3]), (0,0,255), 1, cv.
 ↪LINE_AA)
    params = obtener_houghparams(l)
    hough_params.append(params)
    org = ((l[0] + l[2]) // 2, (l[1] + l[3]) // 2)
    img_con_lineasp = cv.putText(img_con_lineasp, str(i), org, font, fontScale,␣
 ↪color, thickness, cv.LINE_AA)
```

[14]:
```
fig = plt.figure(figsize = (8, 8))
fig.add_subplot(1, 1, 1)
plt.imshow(img_con_lineasp);
plt.title("Segmentos de recta detectados y corregidos", size = 18);
```



[15]:
```
print("segmento   theta    rho      longitud")

for i in range(len(hough_params)):
    set_ = hough_params[i]
    print(i, "\t  ", set_[0], "\t   ", set_[1], "\t", set_[2])
```

```
segmento   theta    rho      longitud
0          30       188.0        397
1          -90      -356.0       398
2          -90      -271.0       300
3          -30      177.0        397
4          -90      -184.0       200
5          30       275.0        298
6          30       361.0        196
7          -30      4.0          196
8          -30      91.0         297
```

```
9           -90       -97.0         97
10          30        448.0         96
11          -31       -88.0         97
```

3. Encontrar las aristas que son paralelas. Reporte cuántos grupos de aristas paralelas encontró y cuáles son paralelas entre ellas.

La rectas paralelas son las que tienen mismo ángulo y distinto radio.

[16]:
```
mp = dict()

for i in range(len(hough_params)):
    if hough_params[i][0] in mp.keys():
        mp[hough_params[i][0]][0].append(i)
        mp[hough_params[i][0]][1] += 1
    else:
        mp[hough_params[i][0]] = [[i], 1]

ans = 0

for it in mp:
    count = mp[it][1]
    if count != 1:
        ans += 1

print("Hay", ans, "conjuntos de líneas paralelas.")

for trash, notrash in mp.items():
    if len(notrash[0]) > 1:
        print('Las líneas', notrash[0], 'son paralelas entre sí.')
```

```
Hay 3 conjuntos de líneas paralelas.
Las líneas [0, 5, 6, 10] son paralelas entre sí.
Las líneas [1, 2, 4, 9] son paralelas entre sí.
Las líneas [3, 7, 8] son paralelas entre sí.
```

La recta 11 debería aparecer junto con 3, 7 y 8, añadiendo tres pares más de líneas paralelas, pero la diferencia en el ángulo es de un grado. Lo cambio a mano.

[17]:
```
hough_params[11][0] = -30
```

4. De la lista de aristas obtenidas en el objetivo 2, agrupe aquellas que formen los triángulos observados en la imagen. Haga una tabla donde los renglones identifiquen al triángulo y las columnas (3) sean los números que identifiquen a las aristas que lo forman.

Primero, voy a guardar todas las combinaciones de tres de mis líneas y eliminar los conjuntos que incluyan líneas paralelas.

[18]:
```
lineas_set = np.arange(12)
```

[19]:
```
conjuntosd3 = combinations(lineas_set, 3)
conjuntosd3 = list(conjuntosd3)
indices_a_borrar = []
i = 0

for conjunto in conjuntosd3:
    l1 = conjunto[0]
    l2 = conjunto[1]
    l3 = conjunto[2]
    if hough_params[l1][0] == hough_params[l2][0] or \
    hough_params[l1][0] == hough_params[l3][0] or \
    hough_params[l3][0] == hough_params[l2][0]:
        indices_a_borrar.append(i)
    i += 1

for i in range(len(indices_a_borrar) - 1, 0, -1):
    j = indices_a_borrar[i]
    conjuntosd3 = np.delete(conjuntosd3, j, 0)
```

Ahora voy a encontrar las intersecciones entre las rectas y, si están mínimo a 90 pixeles de distancia, lo cuento como triángulo.

[20]:
```
def interseccion(l1, l2):
    m1 = ((l1[3] - l1[1]) / (l1[2] - l1[0] + ep))
    m2 = ((l2[3] - l2[1]) / (l2[2] - l2[0] + ep))
    x_cruce = (l2[1] - l1[1] + m1 * l1[0] - m2 * l2[0]) / (m1 - m2 + ep)
    y_cruce = m1 * (x_cruce - l1[0]) + l1[1]
    tol = 2.0
    es_valido = True

    r0 = lineas_p[0][0]
    r3 = lineas_p[3][0]
    r1 = lineas_p[1][0]
    ec0 = ((r0[3] - r0[1]) / (r0[2] - r0[0] + ep)) * (x_cruce - r0[0]) + r0[1]
    ec3 = ((r3[3] - r3[1]) / (r3[2] - r3[0] + ep)) * (x_cruce - r3[0]) + r3[1]
    ec1 = r1[1]
    if y_cruce < ec0 - tol or y_cruce < ec3 - tol or y_cruce > ec1 + tol:
        es_valido = False

    return es_valido, x_cruce, y_cruce
```

[22]:
```
print("triángulo\trectas")
t = 0
min_dist = 90.0

for conjunto in conjuntosd3:
    i1 = conjunto[0]
```

```
    i2 = conjunto[1]
    i3 = conjunto[2]
    l1 = lineas_p[i1][0]
    l2 = lineas_p[i2][0]
    l3 = lineas_p[i3][0]

    val1, v1x, v1y = interseccion(l1, l2)
    val2, v2x, v2y = interseccion(l1, l3)
    val3, v3x, v3y = interseccion(l3, l2)

    dist1 = math.sqrt((v1x - v2x)**2 + (v1y - v2y)**2)
    dist2 = math.sqrt((v1x - v3x)**2 + (v1y - v3y)**2)
    dist3 = math.sqrt((v3x - v2x)**2 + (v3y - v2y)**2)

    if (val1 and val2 and val3) and \
    (dist1 > min_dist and dist2 > min_dist and dist3 > min_dist):
        mi_trian = contor.copy()
        c = np.array([v1x, v2x, v3x])
        r = np.array([v1y, v2y, v3y])
        rr, cc = polygon(r, c)
        mi_trian[rr, cc] = 100
        cv.imwrite('triang_segmen/t{0}.jpg'.format(t), mi_trian)
        print(t, "\t", conjunto)
        t += 1

print("En total hay", t, "triángulos en la imagen")
```

```
triángulo       rectas
0         [0 1 3]
1         [0 1 7]
2         [0 1 8]
3         [ 0  1 11]
4         [0 2 3]
5         [0 2 7]
6         [0 2 8]
7         [0 3 4]
8         [0 3 9]
9         [0 4 8]
10        [1 3 5]
11        [1 3 6]
12        [ 1  3 10]
13        [1 5 7]
14        [1 5 8]
15        [1 6 8]
16        [2 3 5]
17        [2 3 6]
18        [2 5 8]
```

```
19        [ 2  5 11]
20        [2 6 7]
21        [ 2  8 10]
22        [3 4 5]
23        [4 5 7]
24        [4 6 7]
25        [4 6 8]
26        [5 8 9]
En total hay 27 triángulos en la imagen
```