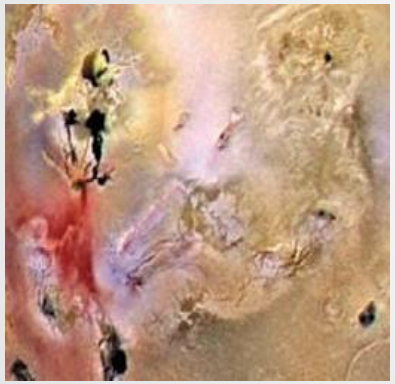






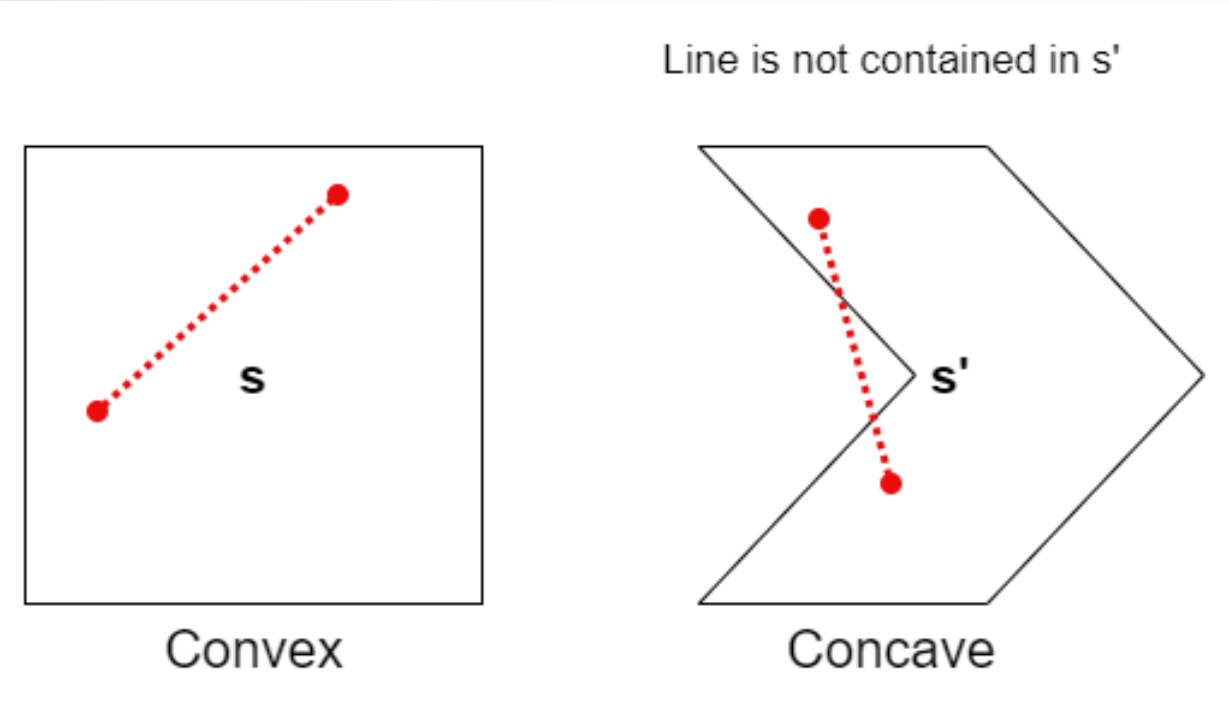

CSC 3141

IMAGE PROCESSING LABORATORY

09 – Convex Hull and the Watershed Algorithm

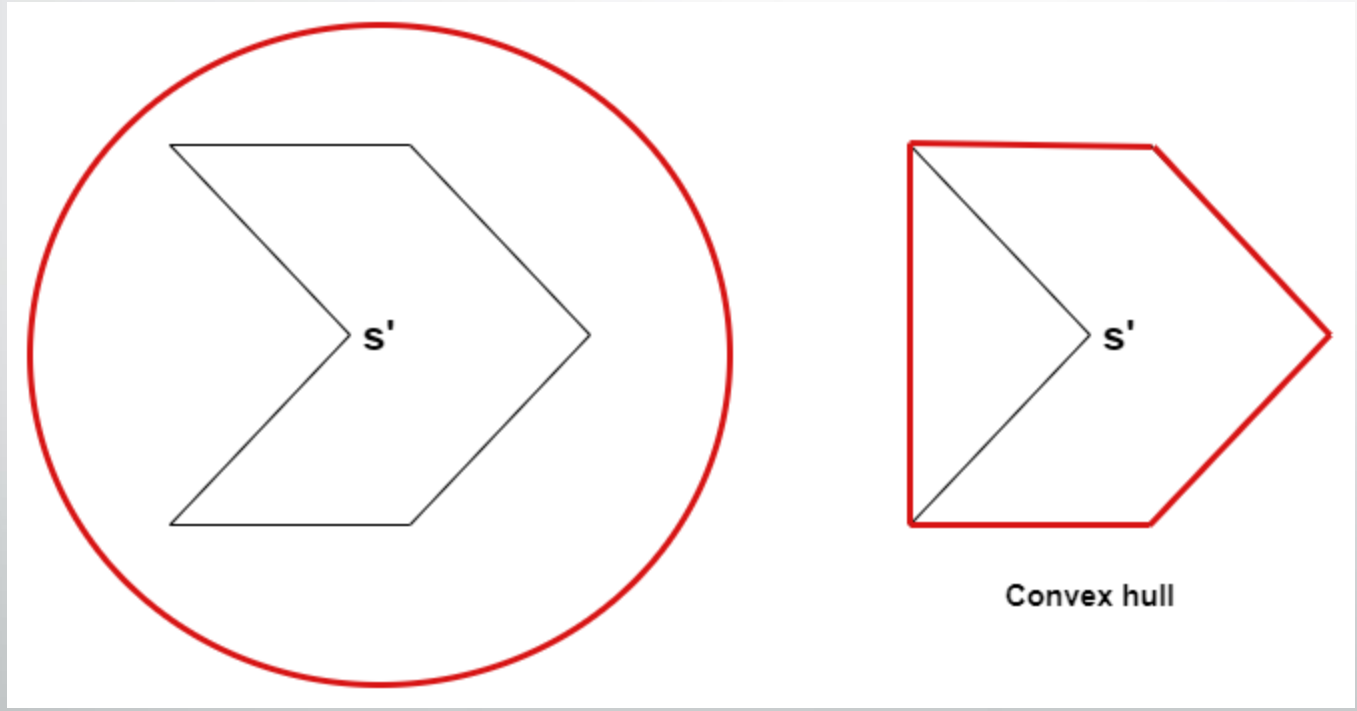
Convex Hull

Any region/shape is said to be **convex** if the line joining any two points within the region is contained entirely in that region.



Convex Hull

The smallest or the convex boundary enclosing a given shape or set of points is known as the convex hull.



Convex Hull

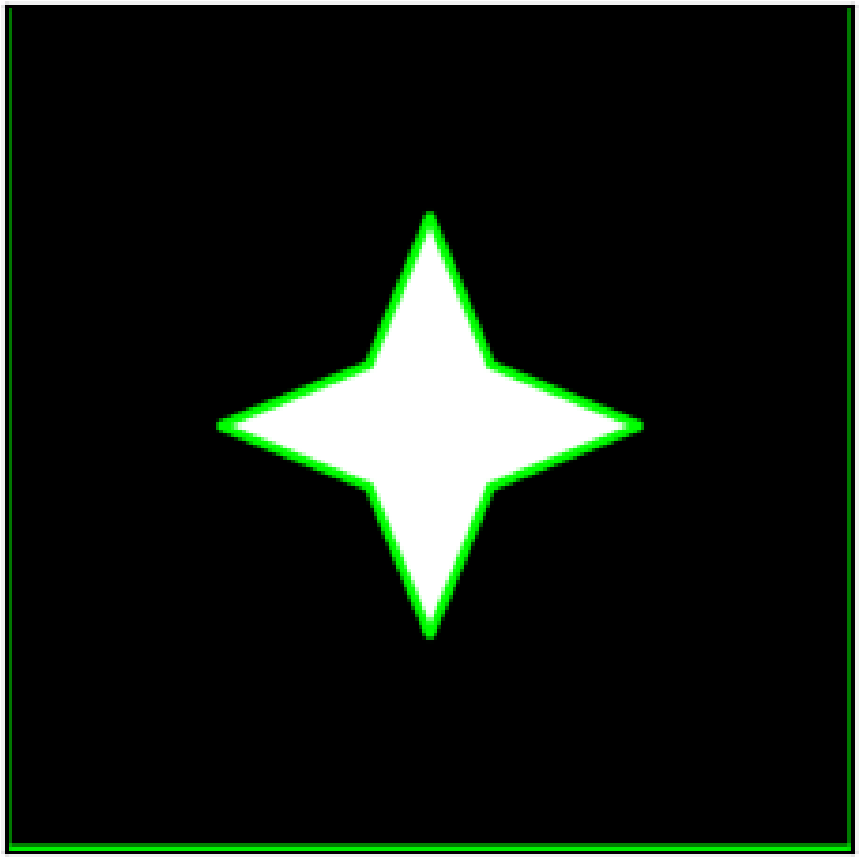
hull = cv.convexHull(points[, hull[, clockwise[, returnPoints]]])

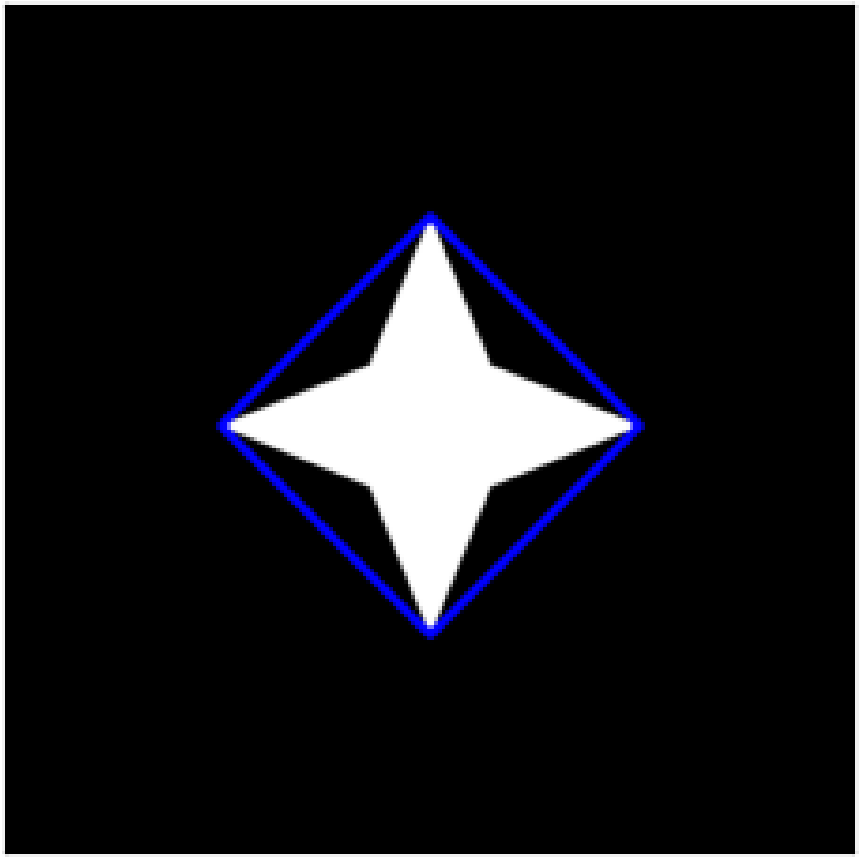
- **points** : the contours
- **hull** : the output
- **clockwise** : Orientation flag. If it is True, the output convex hull is oriented clockwise. Otherwise, it is oriented counter-clockwise
- **returnPoints** : By default, True. Then it returns the coordinates of the hull points. If False, it returns the indices of contour points corresponding to the hull points

Convex Hull

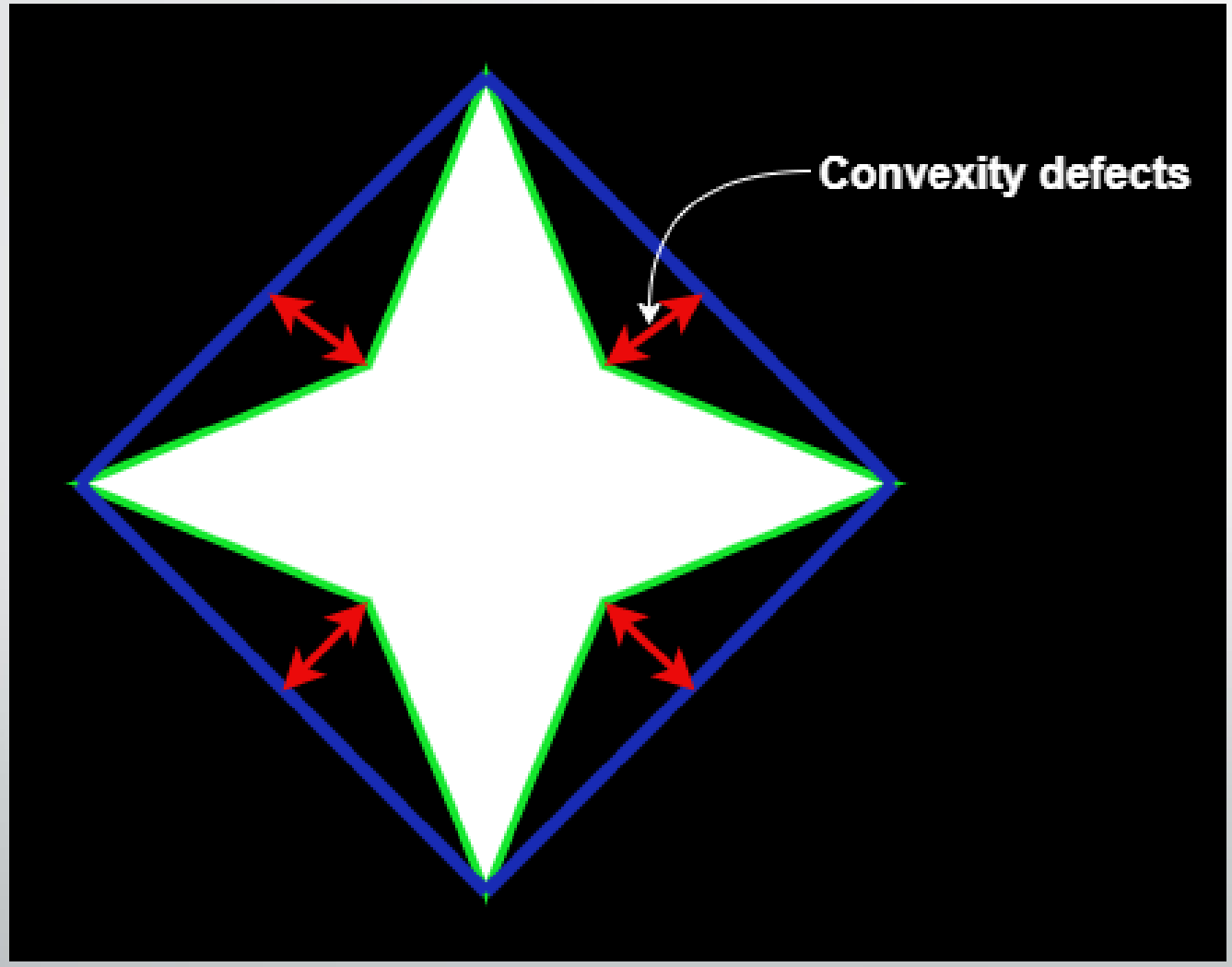
```
img = cv2.imread(r'star.png')
img_copy = img.copy()
# Convert to grayscale
img_gray = cv2.cvtColor(img_copy,cv2.COLOR_BGR2GRAY)
# Create a binary thresholded image
_, binary = cv2.threshold(img_gray, 230, 255, cv2.THRESH_BINARY_INV)
# Find all contours in the image
contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
# Draw all contour
cv2.drawContours(img_copy, contours, -1, (0,255,0), 5)
# Select a contour
cnt = contours[1]
# Get the convex hull
hull = cv2.convexHull(cnt)
# draw the convex hull
hull_img = img.copy()
cv2.drawContours(hull_img, [hull], 0 , (0,0,255), 5)
# Display the result
plt.subplot(121)
plt.tick_params(left = False, right = False , labelleft = False ,
                labelbottom = False, bottom = False)
plt.imshow(img_copy)
plt.subplot(122)
plt.tick_params(left = False, right = False , labelleft = False ,
                labelbottom = False, bottom = False)
plt.imshow(hull_img)

plt.show()
```

Convex Hull



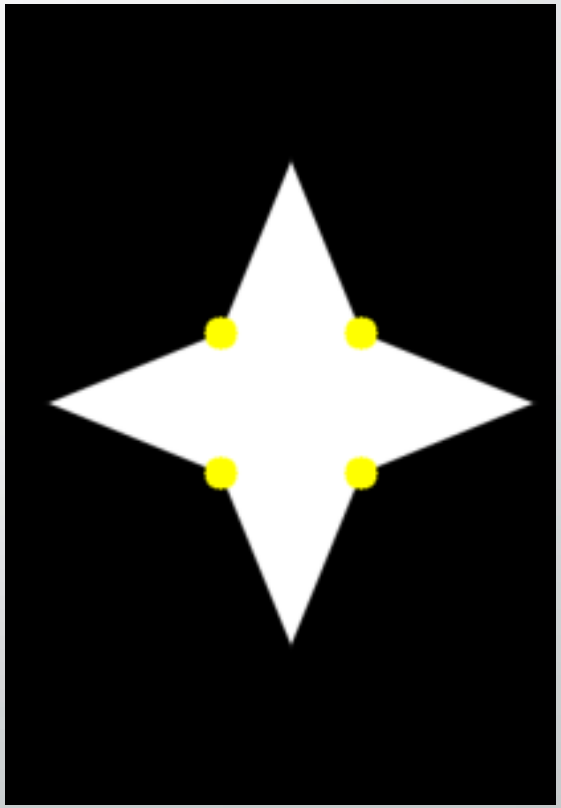


Convexity Defects



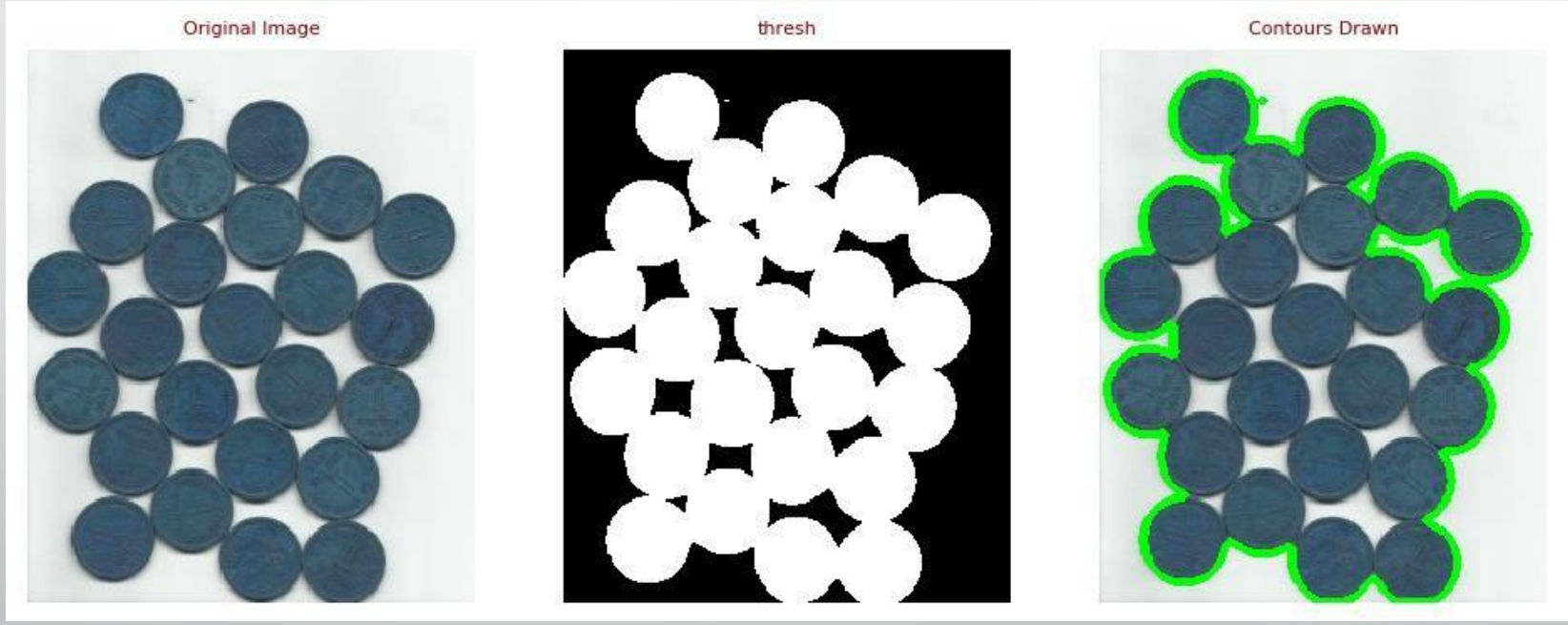
Convexity Defects

```
# Get the convex hull
hull = cv2.convexHull(cnt,returnPoints=False) # mind the 'returnPoints=False'
# Get a copy to draw convexity defects
defects_img = img.copy()
# Find convexity defects
defects = cv2.convexityDefects(cnt,hull)
# Draw circles on convexity defects
for i in range(defects.shape[0]):
    s,e,f,d = defects[i,0]
    f_point = tuple(cnt[f][0])
    cv2.circle(defects_img,f_point,10,[255,255,0],-1)

# Display the result
plt.subplot(111)
plt.tick_params(left = False, right = False , labelleft = False ,
                labelbottom = False, bottom = False)
plt.imshow(defects_img)
plt.show()
```



Contours limitations in instance segmentation

```
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
_, cnts, heir = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
img2 = img.copy()
cv2.drawContours(img2,cnts,-1,(0,255,0),3)
```



Watershed Algorithm

1. Finding the sure background using morphological operation like opening and dilation.

2. Finding the sure foreground using distance transform.

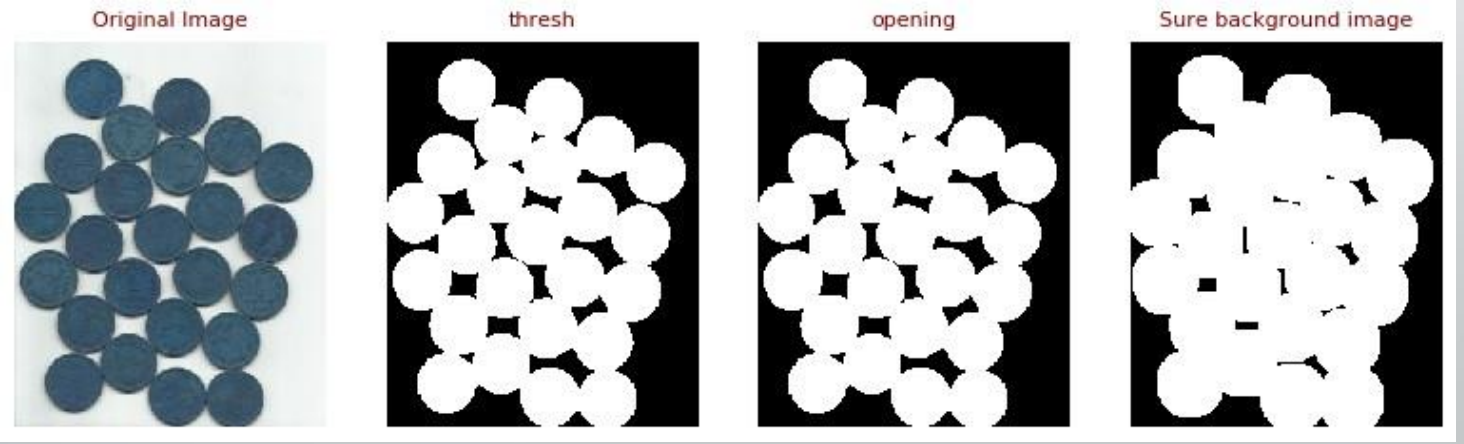
3. Unknown area is the area neither lies in foreground and background and used it as a marker for watershed algorithm.

4. Apply watershed algorithm

Ref: https://datahacker.rs/007-opencv-projects-image-segmentation-with-watershed-algorithm/
Ref: https://people.cmm.minesparis.psl.eu/users/beucher/wtshed.html
Ref: https://pyimagesearch.com/2015/11/02/watershed-opencv/

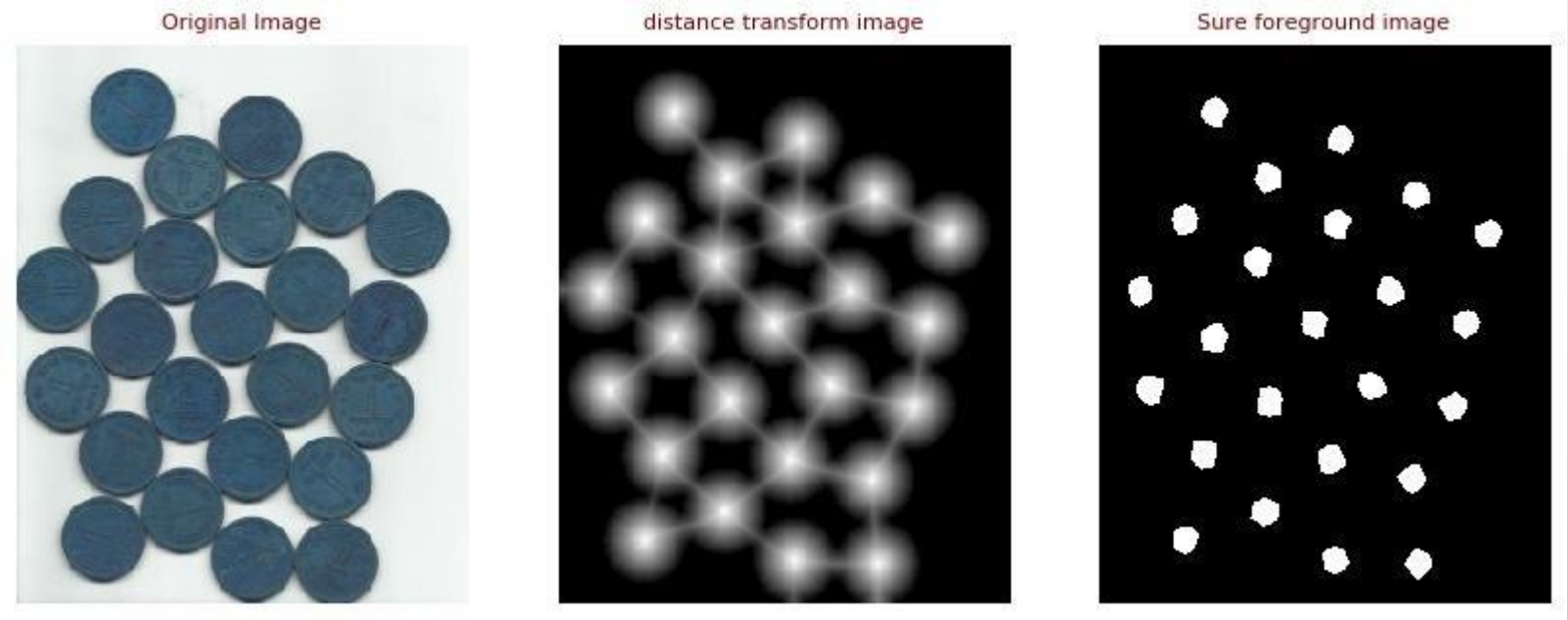
Step 01 : Background Extraction

```
# 1. Background Extraction

img = cv2.imread('images/water_coins.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

## Applying dilation for sure_bg detection
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
## Defining kernel for opening operation
kernel = np.ones((3,3), np.uint8)
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)

## After opening, will perform dilation
sure_bg = cv2.dilate(opening, kernel, iterations=3)
```



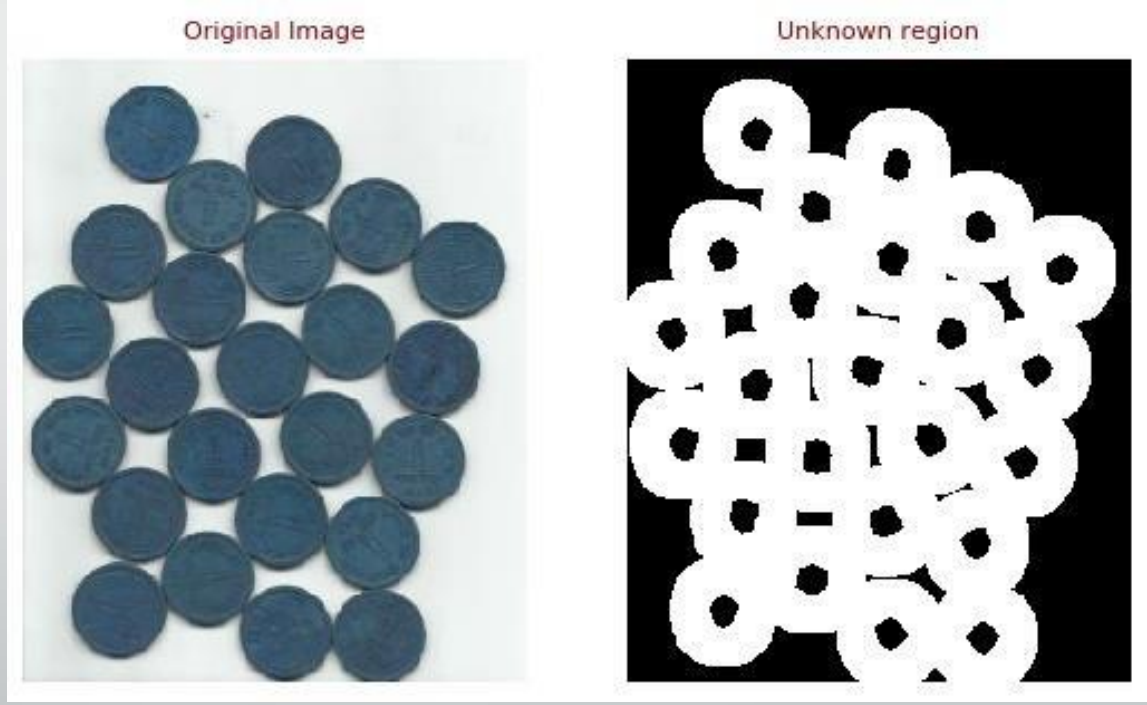
Step 02 : Foreground Extraction

```
# 2. Foreground Extraction

dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)
sure_fg = np.uint8(sure_fg)
```



Step 03 : Finding the Unknown Area

```
# 3. Finding the Unknown Area (Neither sure Foreground Nor for Background)

unknown = np.subtract(sure_bg, sure_fg)
cv2.imshow('unknown', unknown)
```



Step 04 : Applying Watershed Algorithm

```
# 4. Applying Watershed Algorithm

ret, markers = cv2.connectedComponents(sure_fg)
print(markers)
## Add one so that sure background is not 0
markers = markers +1
## Making the unknown area as 0
markers[unknown == 255] = 0

markers = cv2.watershed(img, markers)

## boundary region is marked with -1
img[markers == -1] = (255, 0, 0)
```

Original Image

markers

Watershed region

– END –