Vision 2.0
Workshop 2
Computer Vision

```
contours,hierarchy =
cv2.findContours(mask,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

- Arguments:
  - First one is source image.
  - Second is contour retrieval mode.
  - Third is contour approximation method.

  CONTOUR APPROXIMATION METHOD: contours are the boundaries of a shape. It stores the (x, y) coordinates of the boundary of a shape. But does it store all the coordinates? For eg, if we have to find the contour of a straight line, we need just 2 end points. This is what cv2.CHAIN_APPROX_SIMPLE does. It removes all the redundant points ang compress the contour.

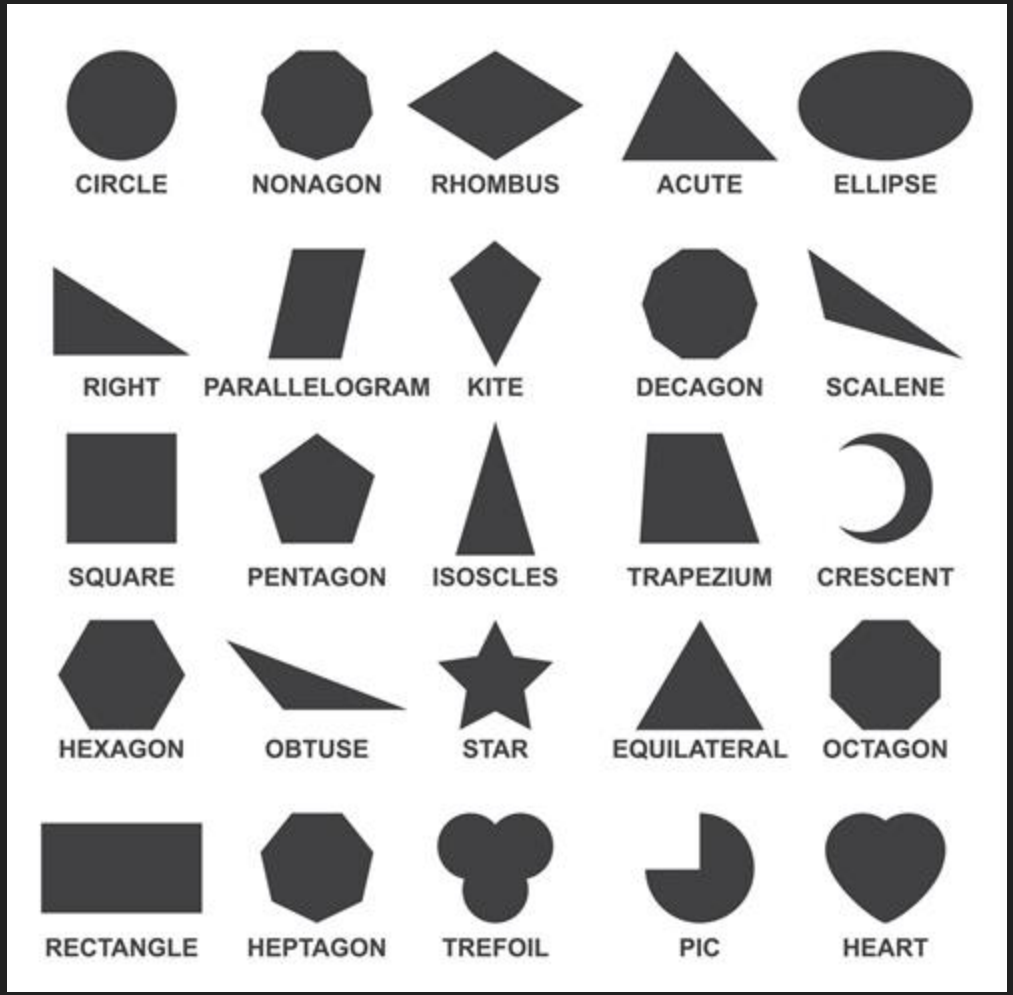
DOUBTS

Drawing on images:

- Line: cv2.line(image, start_point, end_point, color, thickness)
- Rectangle: cv2.rectangle(image, start_point, end_point, color,thickness)
- Circle: cv2.circle(image, center_coordinates, radius, color, thickness)

Shape detection:

- Shape detection is one of the basic tools in image processing.
- We will leverage contour properties to actually label and identify some simple shapes in an image.



Contours:

- Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity.
- The contours are a useful tool for shape analysis and object detection and recognition.
- For better accuracy, use binary images.

```
contours,hierarchy =
cv2.findContours(mask,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

- Outputs:
  - **Contours** : Contours is a Python List of all the contours found by the findContours function. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.
  - **Hierarchy** : Optional output vector, containing information about the image topology.

Drawing Contours:

```
cv.drawContours(img, contours, -1, (0,255,0), 3)
```

- To draw the contours, **cv2.drawContours**() function is used. It can also be used to draw any shape provided you have its boundary points.
- Its first argument is source image.
- Second argument is the contours which should be passed as a Python list.
- Third argument is index of contours (useful when drawing individual contour. To draw all contours, pass -1).
- Remaining arguments are color and thickness.

Drawing Contours:

- To draw all the contours in an image:

```
cv.drawContours(img, contours, -1, (0,255,0), 3)
```

- To draw an individual contour, say 4th contour:

```
cv.drawContours(img, contours, 3, (0,255,0), 3)
```

- But most of the time, below method will be useful:

```
cnt = contours[4]
cv.drawContours(img, [cnt], 0, (0,255,0), 3)
```

Contour Features

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$

- Moments:
  - What are moments in physics?
  - It's a weighted average of image pixel intensities and we can get the centroid or spatial information from the image moment
  - From this moments, you can extract useful data like area, centroid etc.
  - We will use it to calculate centroid.
  - Centroid :
    - `cx = int(M['m10']/M['m00'])`
    - `cy = int(M['m01']/M['m00'])`

```
M=cv2.moments(contours[0])
```

- What will M00 give?

Contour Features

- Area : Contour area is given by the function cv2.contourArea() or from moments, M[‘m00’].

```
Area = cv2.contourArea(contours[index])
```

- Perimeter : Contour perimeter is given by the function cv2.arcLength().

```
Perimeter = cv2.arcLength(contours[index], True)
```

Second argument specifies whether shape is a closed contour (if passed True), or just a curve.

Contour Features(contd)

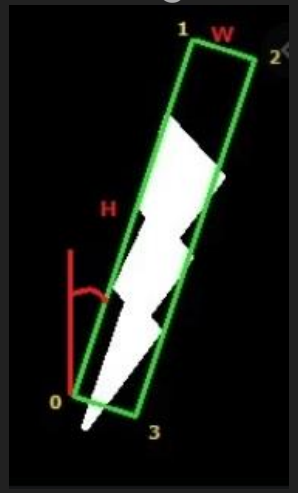
- Bounding Rectangle : It is the rectangle that will enclose the contour. It is of two types:
  - Straight Bounding rectangle: It is a straight rectangle, it doesn't consider the rotation of the object. So area of the bounding rectangle won't be minimum. It is found by the function cv2.boundingRect().
  - Let (x,y) be the top-left coordinate of the rectangle and (w,h) be its width and height.

```
x,y,w,h = cv2.boundingRect(cnt)
img = cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
```

Contour Features(contd)

- Minimum Area Rectangle: Here, bounding rectangle is drawn with minimum area, so it considers the rotation also.
- The function used is cv2.minAreaRect(). It returns a Box2D structure which contains following details - (top-left corner(x,y), (width, height), angle of rotation).
- But to draw this rectangle, we need 4 corners of the rectangle. It is obtained by the function cv2.boxPoints()

```
rect = cv2.minAreaRect(cnt)
box = cv2.boxPoints(rect)
box = np.int0(box)
im = cv2.drawContours(im,[box],0,(0,0,255),2)
```



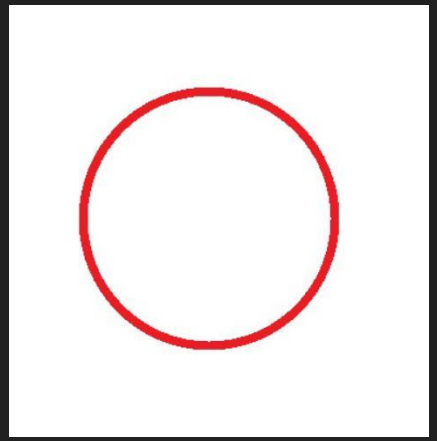
Contour Approximation:

- It approximates a contour shape to another shape with less number of vertices depending upon the precision we specify.
- To understand this, suppose you are trying to find a square in an image, but due to some problems in the image, you didn’t get a perfect square, but a “bad shape”.
- Now you can use this function to approximate the shape. In this, second argument is called epsilon, which is maximum distance from contour to approximated contour. It is an accuracy parameter. A wise selection of epsilon is needed to get the correct output.
- Third argument specifies whether curve is closed or not.

Contour Approximation(contd):

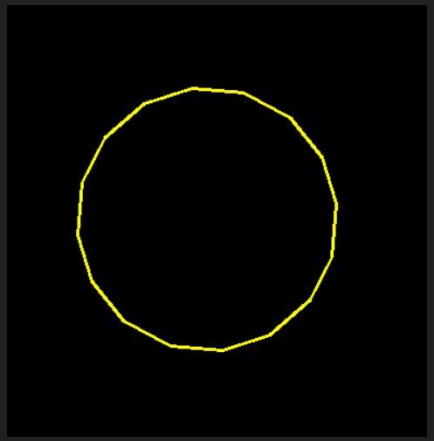
```
epsilon = 0.1*cv2.arcLength(contours[index],True)
approx = cv2.approxPolyDP(contours[index],epsilon,True)
```

- Epsilon sets approximation accuracy.
- Below, in second image, green line shows the approximated curve for epsilon = 10% of arc length. Third image shows the same for epsilon = 1% of the arc length.



ORIGINAL IMAGE

Epsilon is 20



Epsilon is 5

Detecting shapes using contour approximation:

- We can use the fact that triangle has three vertices, rectangle has four vertices and so the remaining ones will be circle.

```
img = cv2.imread("shapes.jpg", cv2.IMREAD_GRAYSCALE)

_, threshold = cv2.threshold(img, 240, 255, cv2.THRESH_BINARY)

_, contours, _ = cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

for cnt in contours:
    approx = cv2.approxPolyDP(cnt, 0.01*cv2.arcLength(cnt, True), True)
    if len(approx) == 3:
        print("Triangle it is!")
    elif len(approx) == 4:
        print("Rectangle it is!")
    else:
        print("Circle it is!")
```

Detecting shapes using some other methods:

- Most of the simple geometrical shapes have some properties associated with them.
- Let’s take an example:
  - For a circle, $(\text{area}/(\text{perimeter})^2)$ is a constant equal to $(1/(4*\pi))$.
  - Do the same thing for squares.