Cairo University

Faculty of Engineering

Systems and Biomedical Department

Spring 2024

Computer Vision(SBE 3230)

Assignment 2
"Edge and Boundary Detection"

Abdulrahman Emad

Mariam Ahmed Saied

Mourad Magdy

Youssef Ashraf

Ziad ElMeligy

Under the supervision of

Prof/Ahmed Badawy

Eng/Layla Abbas

Eng/Omar Hesham

Table of Contents

Detection of Lines, Circles, and Ellipses (Hough Transform)

In order to identify lines, circles, and ellipses in the provided images, we applied the Hough transform. And it's a method for identifying basic shapes in photographs, such circles, lines, and ellipses. For visual aids, we then superimposed the identified shapes on the original photographs.

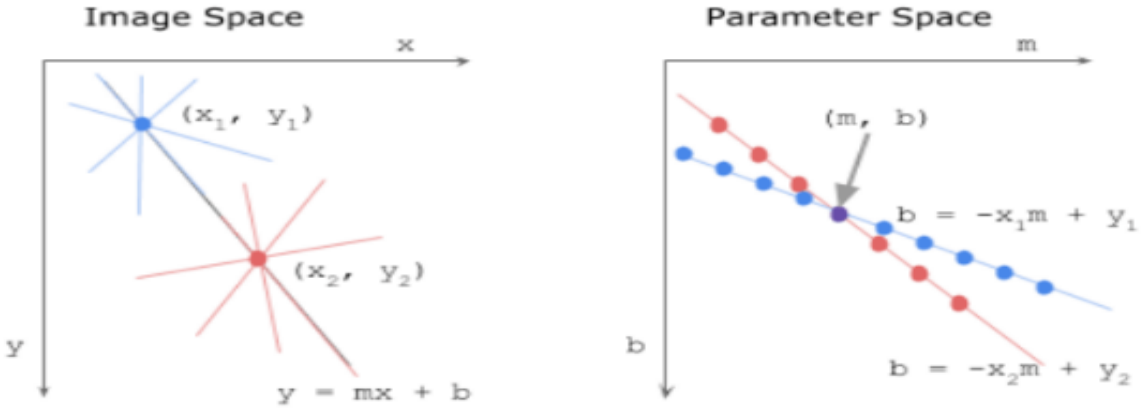
First: Lines

Original Hough transform (Cartesian Coordinates)

In image space line is defined by the slope m and the y-intercept b :

"Y = mx + b"

Therefore, we must define these characteristics in order to detect the line in the picture space, even though they are not relevant in the image domain. Lines from the picture domain represent a point in the other domain with m and b coordinates. Lines in the Hough domain will be mapped to points on the same line in the picture domain. At a place where these lines cross, there are certain values m and b.



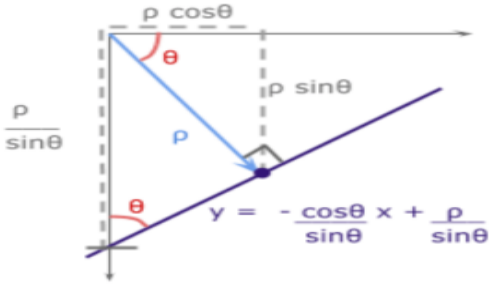
Alternative Parameter Space (Polar Coordinates)

We must switch to polar coordinates since the slope of vertical lines in cartesian coordinates is not determined. A line in polar coordinates is defined by two numbers, ρ and θ, where ρ is the line's norm distance from the origin. θ is the angle formed by the horizontal x axis and the norm.

$$y = \frac{-cos(\theta)}{sin(\theta)}x + \frac{\rho}{sin(\theta)}$$

and

$$\rho = xcos(\theta) + ysin(\theta)$$



The Range of values of ρ and θ:

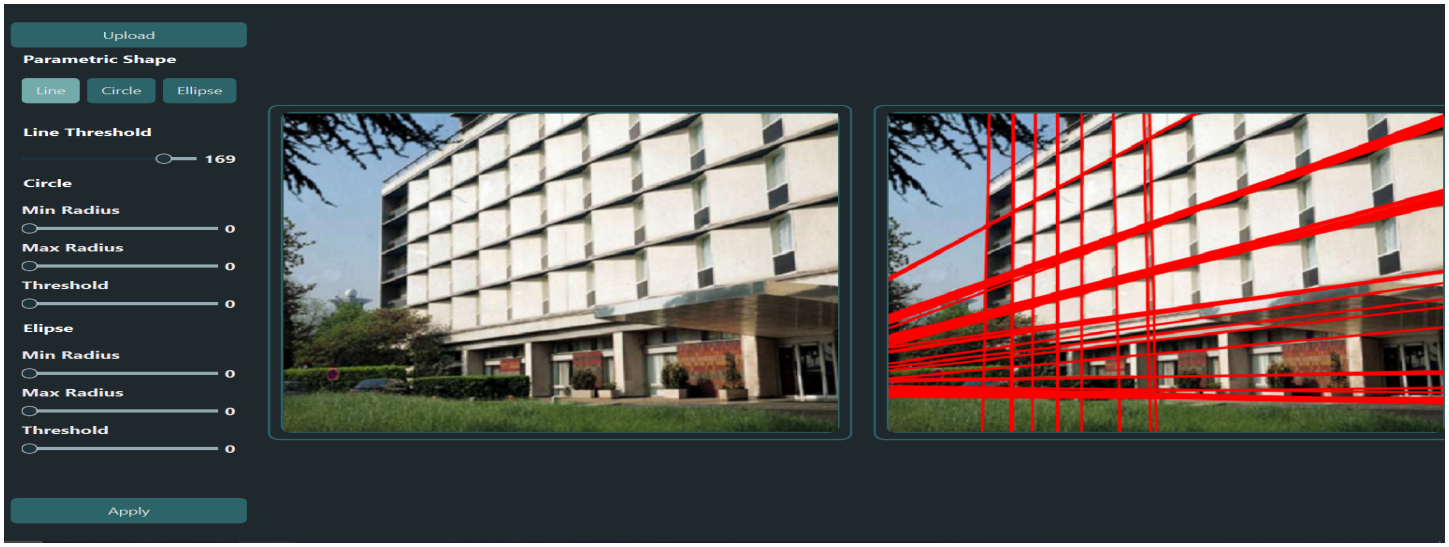
- θ in polar coordinate takes value in range of -90 to 90
- The maximum norm distance is given by diagonal distance which is

$$\rho_{max} = \sqrt{x^2 + y^2}$$

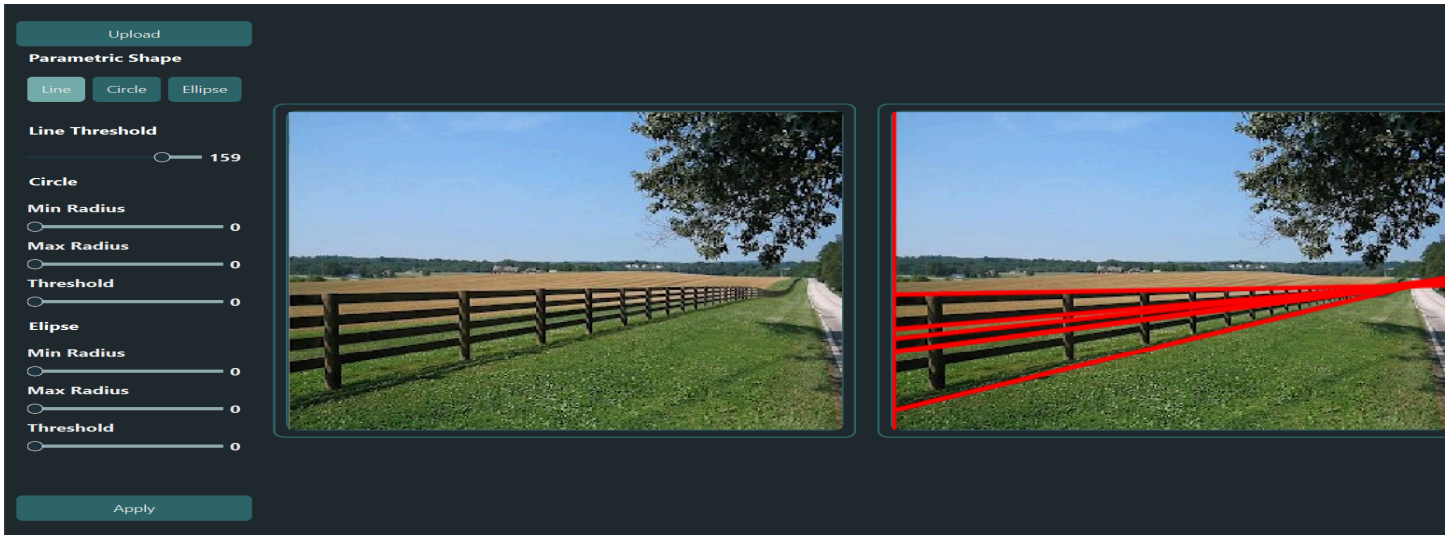
So ρ has values in range from -ρmax to ρmax

Results of our Qt c++ application

Trying Hough Transform on the building image



Trying Hough Transform on the fence image



Second: Circles

In the context of circle detection, the Hough Transform works by representing circles as points in a parameter space. For each point in the input image that is considered an edge point (usually obtained through an edge detection algorithm like Canny), the Hough Transform algorithm iterates through a range of possible circle parameters (center coordinates and radius) and votes in an accumulator array. The peaks in this accumulator array correspond to the parameters of circles in the image.

Circle detection using the Hough Transform involves representing circles in a parameter space, typically (a, b, r), where (a, b) represents the center coordinates and r is the radius of the circle. The following contains the hough transform algorithm steps.

Hough Transform Algorithm

1- Parameter Initialization:

- We initialize the parameters necessary for the Hough Transform. Each circle in the image space is parameterized by its center coordinates (a, b) and radius r.
- Preprocessing: The input image is converted to grayscale to simplify processing.
- Edge detection is performed using the Canny edge detector to identify potential circle boundaries.

2- Accumulator Array Construction:

- We create an accumulator array to accumulate votes for circle parameters (a, b, r). For each edge pixel detected in the Canny edge image, we create center points in the accumulator corresponding to potential circle centers.
- We loop over the edge image and accumulate votes in the accumulator for pixels with intensities greater than a specified threshold, indicating potential circle centers.

3- Local Maxima Detection:

- We identify local maxima in the accumulator array, which correspond to potential circle centers in the original image. These local maxima represent the (a, b) coordinates of the detected circles.

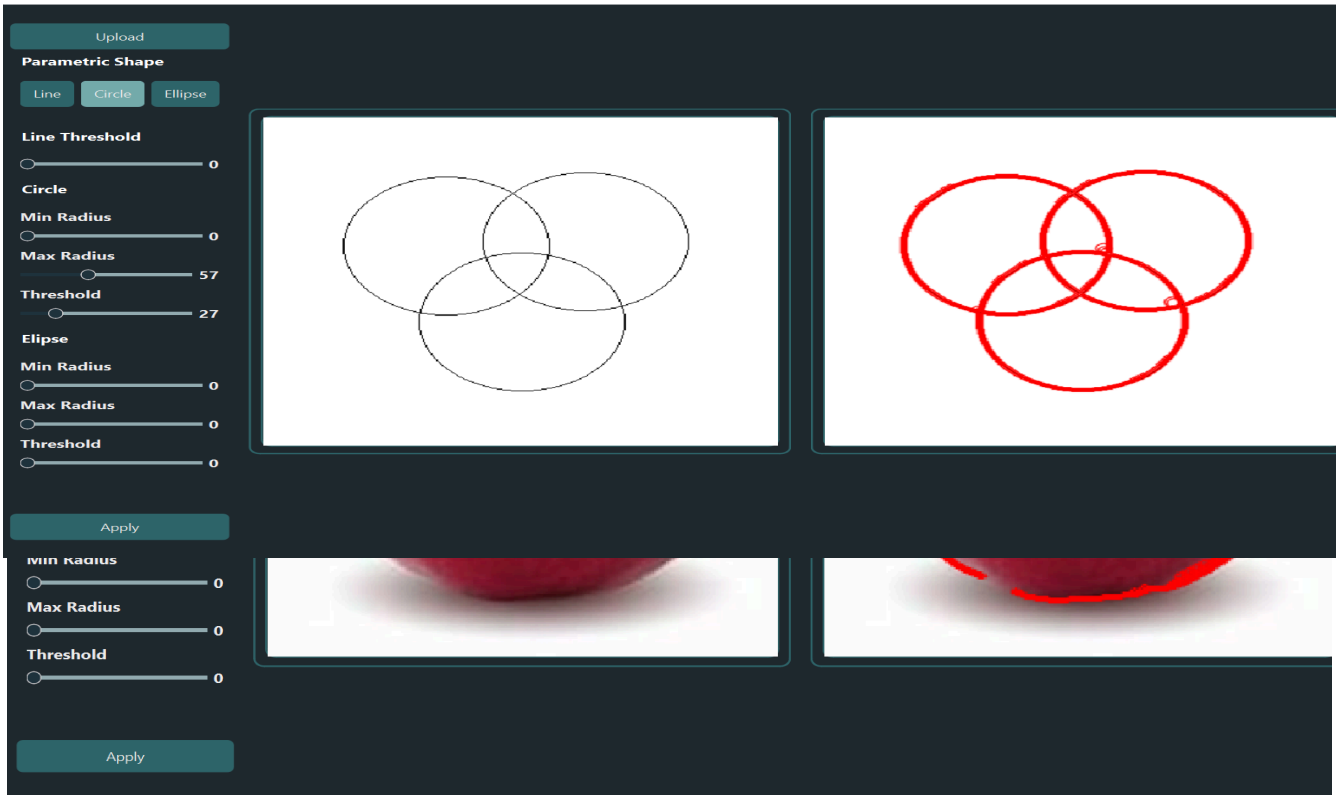
4- Circle Parameter Estimation:

- We iterate over a range of radii values (minRadius to maxRadius) to detect circles of varying sizes. Parameters such as minRadius and maxRadius can be estimated based on the expected number of circles in the image and their spatial distribution.
- Additionally, parameters like minDist (minimum distance between circle centers) can be adjusted based on the spatial arrangement of circles in the image.

5- For the parameters included in the algorithm

- We can roughly estimate minRadius and maxRadius by calculating the number of circles that can fit on the horizontal.
- If our circles are next to each other touching each other, we can give minDist twice the value of minRadius.
- If our circles are far apart, we need to give higher values.
- If we detected fewer circles than we should, we can try to lower the param2

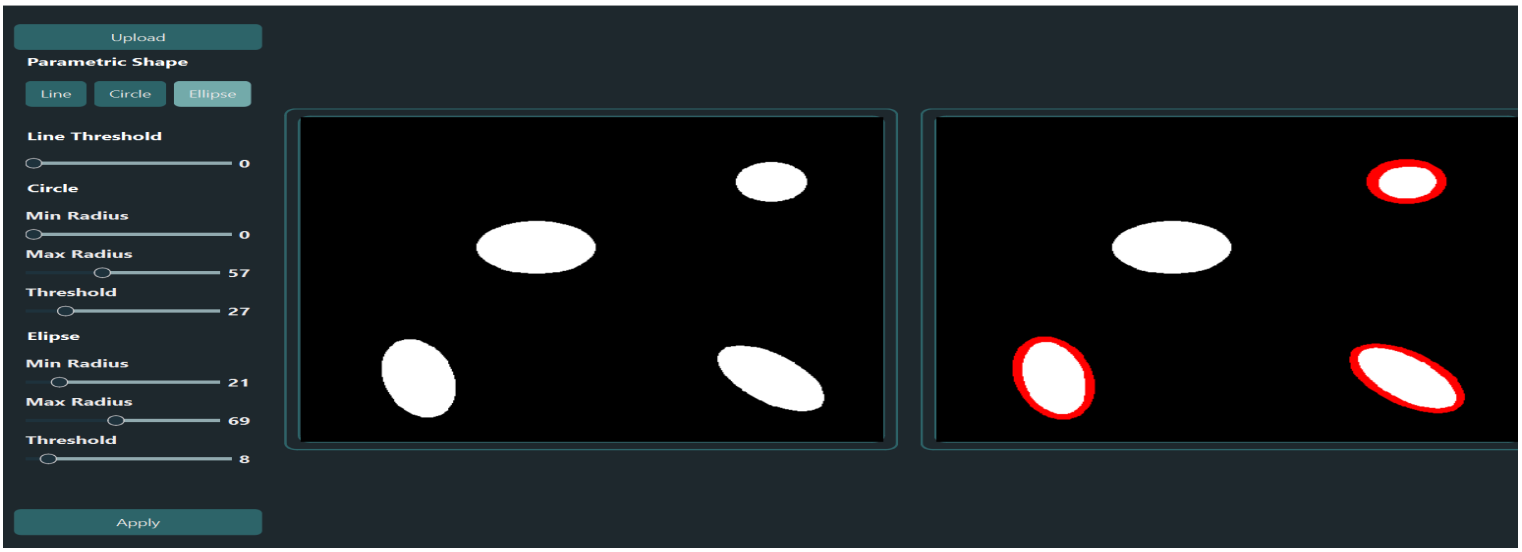
Results of our Qt c++ application



Third: Ellipses

The HoughEllipse method diverges from conventional ellipse detection techniques due to the complexity associated with a 5-dimensional array and extensive computations. Instead, we adopt the principles outlined in "A New Efficient Ellipse Detection Method" (linked here), employing a simplified approach utilizing a 1-dimensional array as an accumulator. Our method assumes pairs of points as potential major axes for ellipses, subsequently testing additional points to verify their validity. To enhance efficiency, we implement multi-threading to parallelize the process, resulting in significantly reduced runtime, with the example image processing in under 10 seconds

Results of our Qt c++ application



Canny Edge Detection

The Canny edge detection algorithm is a widely used technique for detecting edges in images, it is one of the most popular edge detection methods due to its effectiveness and robustness.

Steps of Canny Edge Detection

1- Gaussian Smoothing:

The first step in the Canny algorithm is to apply Gaussian smoothing to the input image. This helps reduce noise and suppress minor variations in pixel intensity.

2- Gradient Calculation:

The gradient of the image is calculated using derivatives (typically the Sobel operators) in both the horizontal and vertical directions. This step highlights regions of rapid intensity change, which often correspond to edges in the image.

3- Gradient Magnitude and Direction:

The gradient magnitude and direction are computed from the horizontal and vertical gradients. The magnitude represents the strength of the edge, while the direction indicates the orientation of the edge.

4- Non-Maximum Suppression:

Non-maximum suppression is performed to thin out the detected edges. For each pixel in the gradient magnitude image, only the local maximum gradient value is preserved along the direction of the gradient.

5- Double Thresholding:

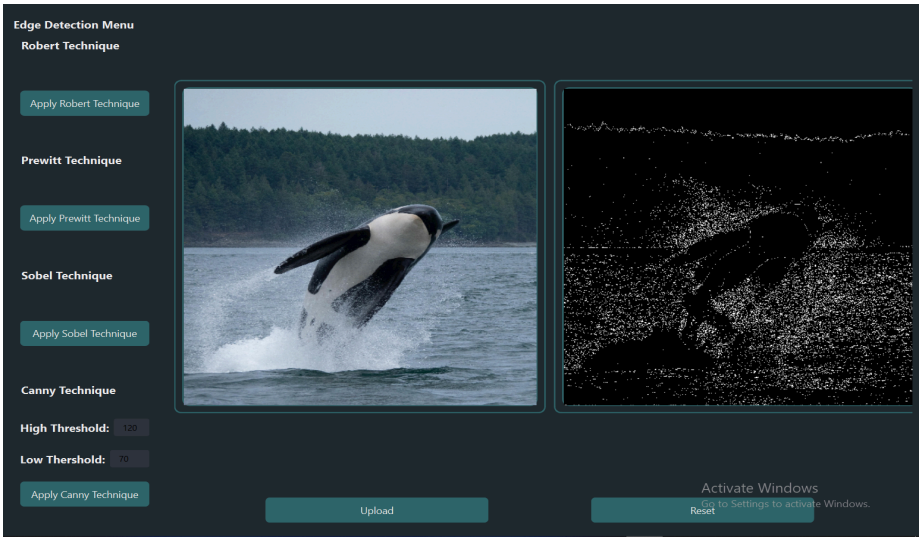
Double thresholding is used to classify pixels as strong edges, weak edges, or non-edges based on their gradient magnitude. Pixels with gradient magnitudes above a high threshold are classified as strong edges, while those below a low threshold are classified as non-edges. Pixels with magnitudes between the two thresholds are classified as weak edges.

6- Edge Tracking by Hysteresis:

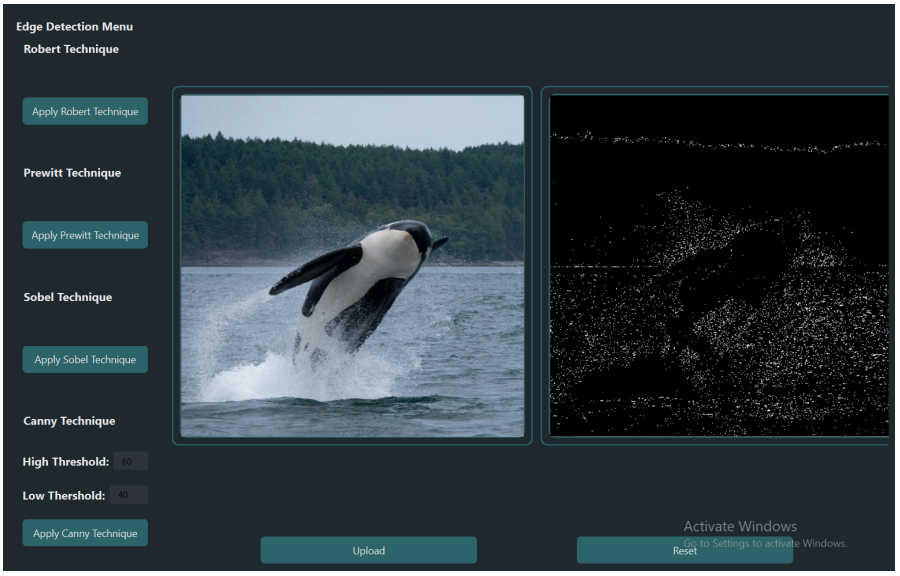
Finally, edge tracking by hysteresis is applied to connect weak edges to strong edges. Weak edges that are connected to strong edges are promoted to strong edges, while isolated weak edges are discarded.

Results Comparison

Opencv implementation for canny



Our implementation for canny



As we can see there is a slight difference between the two images, but actually the difference became this small because **we decreased the thresholding values in our implementation**, we tried to search for the reason why there was a little difference between the two images, there were a lot of reasons like:

1. **Algorithmic difference:**
   OpenCV's Canny edge detection implementation may use a different algorithm or have optimizations that result in different edge detection outcomes compared to our custom implementation.
2. **Parameter Tuning:**
   OpenCV's Canny implementation may use default parameters that are tuned differently from the parameters we have chosen for our custom implementation. It's possible that the default parameters in OpenCV's implementation are more suitable for detecting edges in specific images.
3. **Implementation Details:**
   Differences in how the gradient magnitude, gradient direction, non-maximum suppression, and hysteresis thresholding are computed and applied can lead to variations in the detected edges. Small discrepancies in implementation details can accumulate and result in noticeable differences in edge detection results.

Snake Active Contour

Active contours play a crucial role in various tasks such as image segmentation and object tracking, highlighting their significance in computer vision applications. It discusses the importance of contour-based image segmentation and object tracking, emphasizing their utility in tasks such as medical imaging, surveillance, and autonomous navigation. The report outlines the objectives and structure, setting the stage for an in-depth exploration of active contours in the subsequent sections.

Active Contours

Active contours, also referred to as snakes, represent a powerful technique in computer vision for delineating object boundaries in images.

Contours are fundamental components in computer vision, serving as versatile tools for analyzing and understanding the structure and features within images. Contours represent the boundaries or outlines of objects or regions of interest in an image. Their importance lies in their ability to encapsulate essential spatial information, such as shape, size, and spatial arrangement, enabling various image processing tasks like object detection, recognition, segmentation, and tracking.

How Active Contours work

Active contours, also known as snakes, were developed to address the limitations of traditional image segmentation methods in computer vision. These methods often struggle with complex images, noisy backgrounds, and irregular object shapes, making accurate delineation challenging.

Active contours work based on the principle of energy minimization. They are defined by an initial contour or curve, which is iteratively deformed to minimize an energy function. This energy function consists of two components: internal energy, representing the smoothness and shape of the contour, and external energy, which attracts the contour towards object boundaries or features of interest in the image.

Fig 1. *shows when contours are initially set and then slowly by optimization it relapses and latches onto the edges of the object*

The evolution of active contours is typically driven by the greedy algorithm, where we work on finding the optimum solution for each point rather than the global optimum that takes in considerations all points, each point towards the direction of energy minimization, the contour adjusts its shape and position to minimize the energy functional. This process enables active contours to accurately delineate object boundaries, even in the presence of noise, clutter, or occlusions in the image.

The motivation behind the development of active contours stems from the need for robust and efficient techniques for image segmentation and object delineation. Active contours offer a powerful solution to the challenges faced by traditional segmentation methods. They have found widespread use in various computer vision applications, including medical image analysis, object tracking, and shape modeling, where accurate object localization and segmentation are crucial.

The energy functional

The energy functional used is a sum of several terms, each corresponding to some force acting on the contour.

A suitable energy functional is the sum the following three terms:

$$E = \int (a(s)Econt + b(s)Ecurv + g(s)Eimage)ds$$

Where :
a = alpha, b= beta, g = gamma

In the energy functional for active contours, the terms Econt, Ecurvature(Ecurv and Econtinuous are called the internal energy), and Eimage(External Energy) represent different forces acting on the contour, namely the contour energy, curvature energy, and image energy, respectively. serve as weighting factors that adjust the influence of each energy term. These parameters are crucial as they determine the contour's behavior during evolution and are often adjusted to achieve the best segmentation results.

Econt (Contour Energy)

This term encourages smoothness and continuity along the contour. It penalizes deviations from a smooth curve and helps maintain the contour's coherence during evolution. Minimize the first derivative it works on minimizing the first derivative, which prevents the points from being too far apart from each other thus preventing abrupt edges or irregularities

Ecurv (Curvature Energy)

This term penalizes regions of high curvature along the contour. It encourages the contour to have a smooth and regular shape, preventing it from excessively bending or curving to avoid oscillations of the snake by penalizing high contour.

That’s why we need to minimize the second derivative (curvature)

E image (Image Energy)

This term attracts the contour towards object boundaries or regions of interest in the image. It measures the image gradient or other image features along the contour and guides its evolution towards areas with significant intensity changes or texture. To calculate Image Energy, we need to apply some processing on Images, there are many ways to achieve this. Blurring and Edge detection techniques (Sobel mostly) work very well, achieving good results when calculating magnitude changes.

- Econt and Ecurv are called internal energy terms.
- Eimage is called an external energy term.

Chain Code:

Chain code is a technique used in computer vision and image processing to represent the boundary of an object or region in a binary image. It is particularly relevant in the context of active contours because it provides a compact and efficient way to encode the contour information, making it suitable for contour-based algorithms such as active contours.

In chain code representation, the contour of an object is traced by following its boundary pixels sequentially. Each pixel on the boundary is represented by a code that indicates the direction from the current pixel to the next pixel on the contour. This direction can be encoded using different schemes, but one common convention is to use 8-connected chain code, where each pixel is connected to its eight neighbors (horizontal, vertical, and diagonal).

The eight possible directions are typically represented using numbers from 0 to 7, where each number corresponds to a specific direction relative to the current pixel. For example, in a clockwise manner starting from the east direction (0).

Chain Code Representation

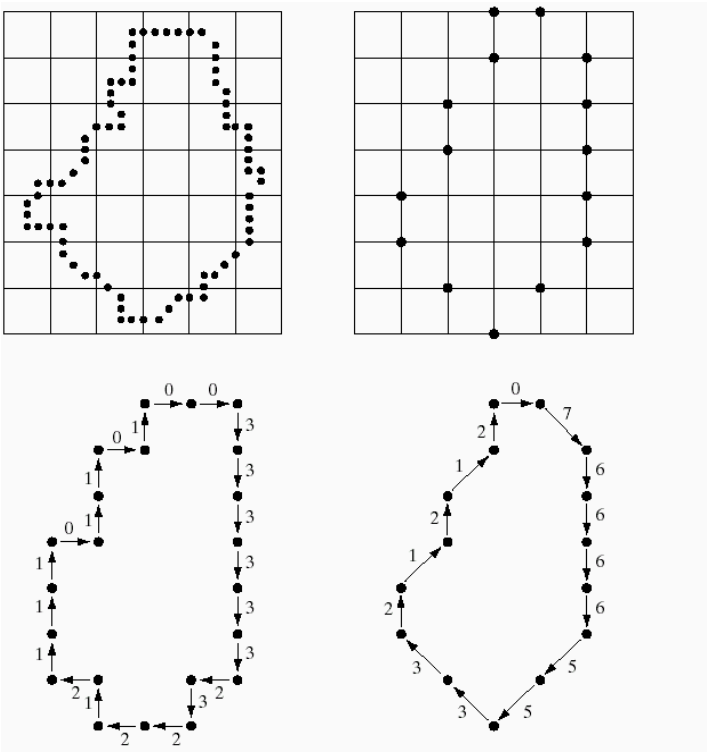


Fig2. This figure represents chain coding on a segmented contour

Advantages of Chain Code

- **Compact Representation**

  Chain code requires minimal memory to store the contour information, as it only stores the sequence of direction codes rather than the coordinates of all contour pixels.

- **Efficient Processing**

  Chain code facilitates efficient contour processing and manipulation, as it enables fast traversal of the contour and easy computation of geometric properties such as length, area, and curvature.

- **Robustness**

  Chain code is inherently robust to noise and small-scale variations in the contour, as it represents the contour's general shape and topology rather than specific pixel locations.

In the following section, the results of the snake algorithm are shown visually, that shows how the snake contour works as an elastic band, so that it bends around the object contour but doesn’t break, also, tweaking the parameters will cause different penalization to the contour, thus leading to different results.

Results of the Snake Active Contour