A COMPUTATIONAL GEOMETRY APPROACH TO DIGITAL IMAGE CONTOUR EXTRACTION

by

Dharma Teja (201207571)
Falak Chhaya (201232523)
Kumar Bipin (201232672)

*A project report submitted in partial fulfillment
of the requirement for the course work
of*

Digital Image Processing

**International Institute of Information Technology
Hyderabad**

Contents

Chapter 1

Introduction

Contours[1] are the boundary lines of geometric shapes within digital images; see Fig. 1.1. Since the identification of contours is crucial for analyzing the contents of an image, contour extraction is one of the most important problems in Image Processing and pattern recognition. Once object contours have been extracted, several shape features that are useful for identifying and classifying objects can be determined. These features include perimeter length, irregularity, width, height, aspect ratio, and area. However, this problem is especially difficult for images with complex shapes and with noise.

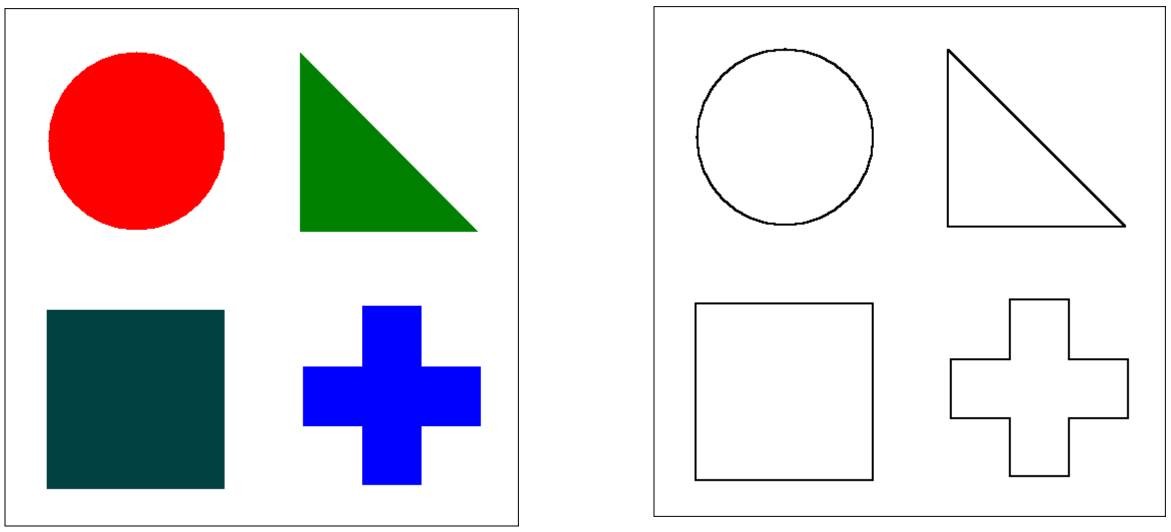


Fig 1.1 Digital image of shapes and corresponding contours

In the mentioned reference paper, a method for automatically extracting contours of arbitrary shapes from digital images using techniques from computational geometry is presented. The aim is to show how a combination of simple geometric algorithms can be used effectively to extract contours even from images with a moderate amount of noise.

[1] We use the term contour as used in image processing to refer to lines and object boundaries. In mathematics, a contour line of a function of two variables is a curve along which the function has a constant value. In cartography, a contour line joins points of equal elevation (height) above a given level, such as mean sea level. For detailed information see *http://en.wikipedia.org/wiki/Contour line.*

1.1 Standard Definitions

A *digital image* is a discrete approximation of an image obtained by *sampling* points with discrete coordinates and *quantizing* the values of each sample. It is formed by a finite number of sample elements equally spaced over a square grid with a rectangular shape. Each element is called a *pixel* and has an *intensity* value. The rows and columns of elements determine the *spatial* coordinates (x, y) of the pixel;

And the intensity determines its gray-scale or color value. In a *gray-scale* image, all pixels have shades of gray ranging from black to white. In the case of *color* images, the intensity determines the color of each pixel according to some color model, such as RGB. A digital image is stored as a two-dimensional array (usually of integers), wherein each element and its value correspond to a pixel and its intensity, respectively. When a digital image is displayed, each pixel is represented by a small square with a unique color determined by its intensity. See Fig. 1.2 for examples of continuous and digital images.

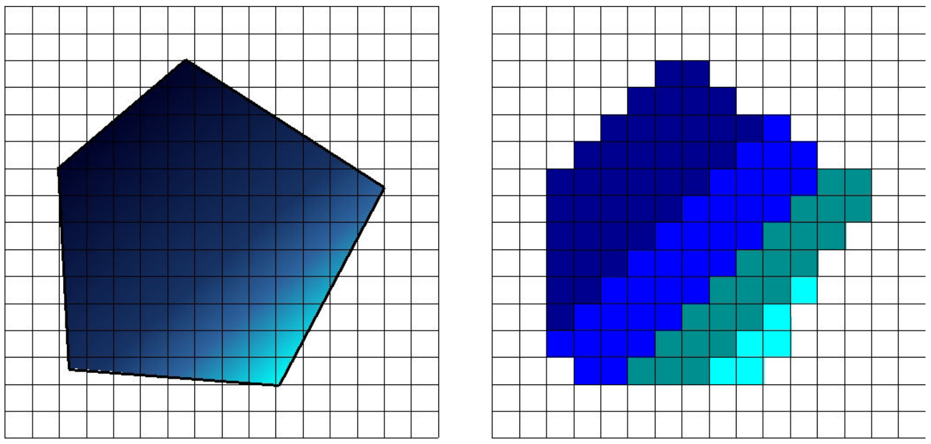


Fig 1.2 Continuous image (Left) and corresponding digital image

Contours are lines, straight or curved, that define forms or shapes. They can be open lines, or they can be closed boundaries. Contours can be represented by pixels, for example, with black pixels over a white background or vice versa. However, even when they can visually represent the contours of an image, these pixels alone do not provide a complete characterization of the contours. Contours are lines, not independent pixels; so in order to have contours defined by pixels, more information is needed. A sequence of consecutive adjacent pixels is a possible representation. However, this kind of representation has some disadvantages: it requires a lot of space because many pixels are needed; it is not scale independent since the number of pixels required changes with the size; and it is discrete, so lines are not generally smooth unless numerous pixels are used. Given these drawbacks, a geometric representation provides a better way to represent contours, for example, with line segments defined by pairs of points. This much simpler representation does not have the problems associated with the discrete counterpart. See Fig. 1.3 for examples of pixel-based and geometric-based contour representations.

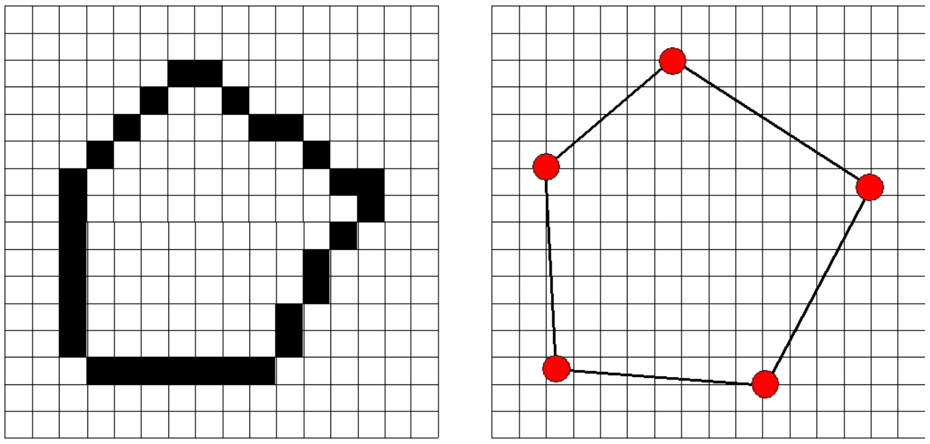


Fig 1.3 Pixel based contour(Left) and Geometry based Contour

1.2 Nature of Work

We have implemented a thesis [1] partially and proposed an innovation to existing work. Apart from that, we experimented for different parameters and came up with optimum values of tuning parameters.

1.3 Previous Work

Traditional methods for finding contours can be classified by scope depending on whether they do local, regional, or global processing. Local methods analyze a small neighbourhood associated with every pixel and link adjacent pixels if they satisfy some criteria. Regional methods use different techniques to connect pixels which are previously known to be part of the same region or contour. In such cases, geometric algorithms, such as polygonal fitting, can be used to efficiently find approximations of contours; however, the knowledge required to apply such algorithms is not always available, so they are not generally applicable. Global methods, such as the Hough transform, do not rely on any kind of prior knowledge, and try to find sets of pixels which lie on curves of specific shapes.

These three methods all present some drawbacks: local methods ignore valuable global information about the geometric proximity of pixels, since they only look at a very small neighbourhood; regional methods require prior knowledge about which pixels are part of which contour; and global methods such as the Hough transform can only be used to find certain types of shapes. Our method exploits the global information about the geometric proximity of pixels, requires no prior knowledge about the regional membership of pixels, and is not restricted to any particular shapes.

Another possible classification for contour extraction algorithms is by the way they work. The extraction of line segments from images is an important problem related to contour extraction: straight lines are a subset of all possible contours and since any curve can be approximated by small segments some contour extraction algorithms are actually line extraction algorithms. According to, several models have been reported in the literature for the extraction of line segments from images, and these are broadly classified into four categories: statistical based, gradient based, pixel connectivity-edge linking based, and Hough transform based. The linking algorithms work by connecting edge pixels based on proximity and orientation, and dividing the contours into straight line segments. Our method uses a linking algorithm, but instead of connecting edge pixels, it uses computational geometry to connect oriented points.

Many algorithms related to image processing and specifically contour extraction are related to the field of discrete or digital geometry. This field has become more important in the last decade, but it has evolved independently of the field of computational geometry. However, researchers are now using results from computational geometry to solve problems in discrete geometry, and the integration of both fields seems promising. This method for extracting contours is a case wherein computational geometry is used to solve a problem in discrete geometry: extracting contours from points in a discrete grid. We extract contours from digital images by moving to a continuous domain and using well established results from computational geometry. The result is not discrete, but the extracted contours could be rasterized to discretize them if it is necessary.

1.4 Computational Geometry Approach

This method consists of two stages: a pre-processing stage that extracts a set of oriented points from the input image, and a second stage that finds the contours among the oriented points using geometric algorithms. The second stage is the most important and has three steps: (1) points are first filtered by a clustering technique; (2) then points are linked, based on proximity and orientation, into paths representing the contours; (3) and finally paths are simplified by reducing the number of points they have. See Fig. 1.4.

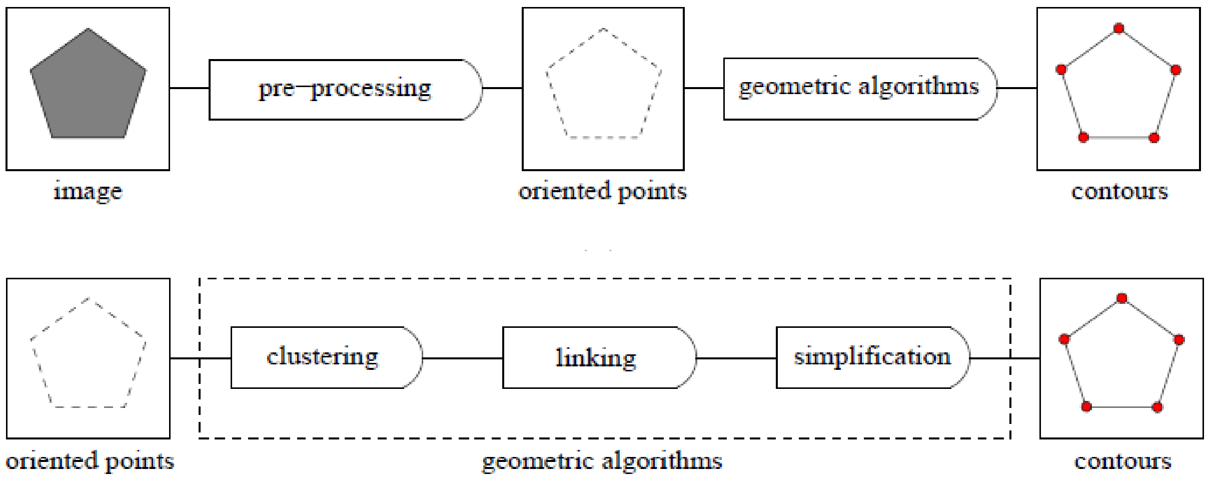


Fig 1.4 Overview of the method

Chapter 2

Oriented Edge Point Detection

We first transform the problem of finding contours into a geometric problem. The goal of the pre-processing stage is to find a set of points that are possibly part of the contours. To do this, we use an edge detector to extract edge pixels from the image, and then we convert them into oriented points. Finding the contours is then a matter of connecting those points into meaningful boundaries.

At the pre-processing stage, a Sobel edge detector [2] is used to determine possible contour pixels, which are then transformed into oriented points. The edge detector outputs a set of *edge pixels* wherein the intensity of the image changes abruptly. Each edge pixel has a *magnitude* indicating how good or strong the edge at the pixel location is, and a *direction* indicated by an angle. Next, each pixel is transformed into an *oriented* point p_i located at the centre (x_i, y_i) of the pixel, with its orientation α_i given by the edge direction, and a weight w_i initialized with the edge magnitude; see Fig. 2.1.

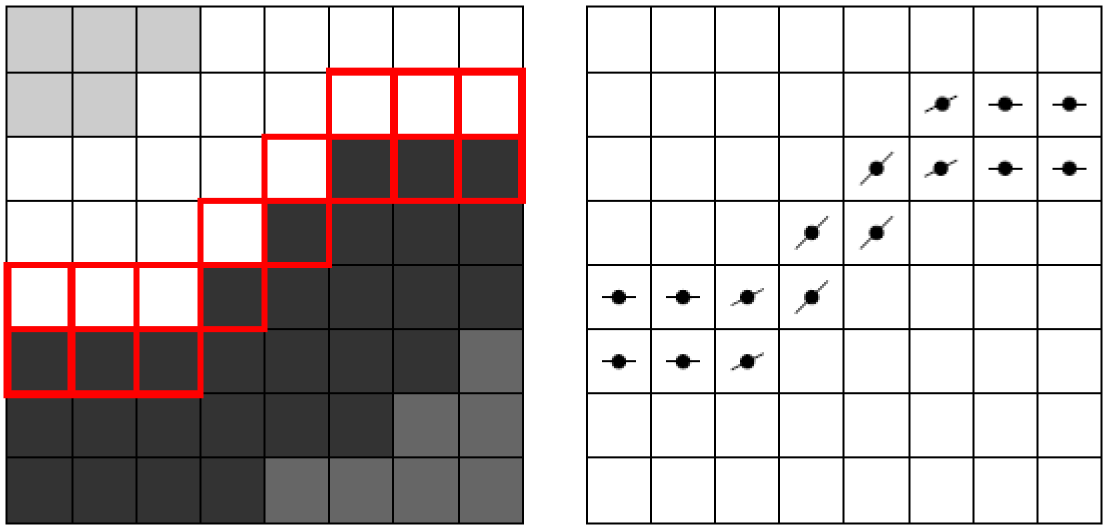


Fig 2.1 Digital image and corresponding oriented edge pixels

There are many edge detectors that can be used to extract edge pixels; some examples include Sobel, Laplacian and Canny. We use a Sobel edge detector, but others could be used as well. Since we want to focus on the geometric algorithms of the second stage, and they are independent of the method used by the pre-processing stage, this choice is not so relevant.

However, it is important that the method used return not only the gradient values, but also the orientation of gradient, which is used by later parts of the algorithm.

2.1 Implementation

As mentioned earlier, we have used Sobel edge detector as we need gradient magnitude and its orientation. The formulae used for the same are given below.

$$\text{Grad}(i,j) = |\text{Grad\_X}(i,j)|*0.5 + |\text{Grad\_Y}(i,j)|*0.5 \quad \text{--} \quad \textbf{(2.1)}$$

$$\text{Orientation}(i,j) = \text{atan}(\text{Grad\_X}(i,j)/\ \text{Grad\_Y}(i,j)) \ – \ \textbf{(2.2)}$$

$$\text{Threshold}(I) = \text{mean}(\text{Grad}(i,j)) + (255 - \text{mean}(\text{Grad}(i,j)))/\text{t\_par} \ – \ \textbf{(2.3)}$$

Note1:
The parameter t_par in equation **(2.3)** is used to change threshold according to image content and quality.

Note2:
As image coordinate axes are different from Euclidean coordinate axes (y axis is inverted). To fix this the orientation in equation **(2.2)** following method can be used.

Method:
1) Get the orientations in range [-pi,pi]
2) If orientation is negative make it positive.
3) If orientation is positive subtract it from PI.

2.2 Output

The output of this stage is ‘oriented edge points’ which has the following structure.

1) X-position
2) Y-position
3) Gradient Magnitude
4) Orientation
5) Unique Number

Thus each edge point, which is above threshold, would be passed on to the next stage with these four characteristics. So the file format looks like

We stored all the edge points in a file having following format.

```
/*
Image width, Image Height
Number of points
Position-X, Position-Y, Gradient, Orientation, Unique Number
Position-X, Position-Y, Gradient, Orientation, Unique Number
.
.
```

```
.
.
Position-X, Position-Y, Gradient, Orientation, Unique Number
*/
```

Chapter 3

Point Clustering

Clustering techniques are very useful for image processing and pattern recognition. The objective of clustering analysis is to partition a set of points into groups or clusters that are natural according to some similarity measure. Clustering algorithms frequently use computational geometry, as the similarity between clusters is usually expressed in terms of Euclidean distances between points in a feature space.

Clustering-based algorithm is used to reduce the number of points for two key reasons. First, it reduces the processing time of the following steps, and second, it improves the results of the linking step, which when the points are close together might find multiple lines where there should be a single contour. Geometric algorithms can be more time consuming than local processing pixel-based algorithms. Therefore, it is convenient to reduce the input size. When doing so, however, it is important for the new point set to resemble the initial point set. This is similar to the geometric problem of *dot map simplification* which, given a point set, tries to find a smaller set whose distribution approximates that of the original set [6]. The difference is that in our case we also require the orientations to approximate those of the original set.

3.1 Algorithm

Number of points are reduced using a simple iterative greedy algorithm that repeatedly merges the closest pair of points into a new point until the distance between the closest pair reaches a threshold d_{MAX} ; see Fig. 3.1.

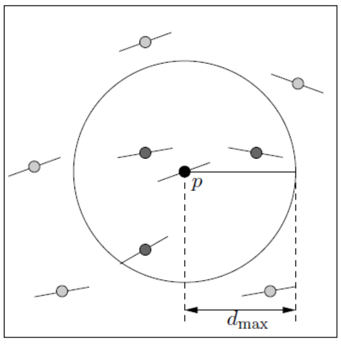


Fig 3.1 Pictorial representation of a particular situation while clustering

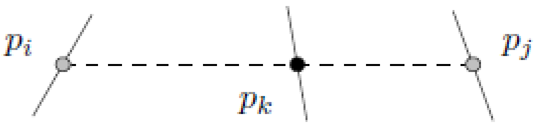
When a pair of points is merged into one, the values for the new point are weighted averages of the values of the original points. This ensures that the distribution and orientations of the new point set approximate those of the original set; see Fig. 3.2. However, we must take special care to determine the new orientation. Stepwise algorithm is as given below

1) Find d_min:
   d_min is the minimum distance among all the edge points.
2) Set d_max:
   d_max = c*d_min , c is inversely proportional to the number of points retained after clustering.

3) Find featurepairs(FP):
   Feature pairs are set of pairs of points whose distance is less than d_max.

4) Find Eligible pair for clustering:
   To find the eligible pair to be clustered from FP whose distance is less than all other points in FP. Let that pair be P={p1,p2}.

5) Merge P
   Two points in P are merged by using following formulas and the new point is given a unique number .



$$x_k = \frac{x_i w_i + x_j w_j}{w_i + w_j}, \quad y_k = \frac{y_i w_i + y_j w_j}{w_i + w_j}, \quad \alpha_k = \frac{\alpha_i' w_i + \alpha_j' w_j}{w_i + w_j}, \quad w_k = w_i + w_j.$$

Where $\alpha'_i \in \{\alpha_i , \alpha_i + \pi\}$ and $\alpha'_j \in \{\alpha_j , \alpha_j + \pi\}$ are chosen so that the orientation of p_k is

close to the orientations of both p_i and p_j :

if $|\alpha_i - \alpha_j| \leq \frac{\pi}{2}$ then $\alpha'_i = \alpha_i$ and $\alpha'_j = \alpha_j$;

if $\alpha_i < \alpha_j$ then $\alpha'_i = \alpha_i + \pi$ and $\alpha'_j = \alpha_j$, and

if $\alpha_j < \alpha_i$, then $\alpha'_i = \alpha_i$ and $\alpha'_j = \alpha_j + \pi$.

6) Update step
   Reomve points P from the original set.
   Add new merged point to original set.

```
        Remove all pairs of points in FP contains P.
        Add pairs formed by merged point and other points whose distance is less than dmax.
```

Repeat 4,5,6 until for every edge point there is no other point with in dmax.

3.2 Output

The output format is just the same as that of the previous stage. The only difference is number of points in the output of clustering are less than that of the previous stage.

Chapter 4
Point Linking

After an image has been segmented into a set of regions or edge pixels, the next step is to find the contours determined by those regions or pixels. This is usually done by a simple *contour tracing* algorithm, that traces the boundaries by starting from a known contour pixel and repeatedly moving to adjacent contour pixels until a stopping condition is met (usually returning to the original pixel). Contours obtained in this way are then stored as a sequence of pixels. However, these algorithms are very simple; the algorithms only work with very clean boundaries. Another possibility is to use edge linking algorithms, which link edge pixels if they are within a small neighbourhood and have a similar magnitude or direction [28–30, 34].

The linking step is the one that actually finds the contours. Prior to this step, we have a set of oriented points that have to be connected in order to find the contours. The objective is to find a set of paths representing the contours. These *paths* are sequences of line segments between the given points, or polylines. Paths can be represented by other types of functions, such as spline curves, but we use polylines because they are efficient for computations and because any curve can still be approximated by straight line segments.

Similar to edge linking algorithms, our algorithm links points based on proximity and orientation. However, our linking algorithm is conceptually simpler. A typical pixel-based contour extraction algorithm might: (1) extract edge pixels with an edge detector, (2) fill gaps between edges, (3) connect pixels (contour tracing), and optionally (4) approximate the contours with line segments. In our method, the edge detection is the same, but steps 2 to 4 are all done by the linking algorithm. Gaps are filled by linking points, and there is no need for a line approximation step, as points are linked by line segments. Also, pixel based algorithms usually have to consider multiple special cases. In contrast, our linking algorithm works by following very simple rules. We now describe the algorithm.

4.1 Algorithm

The algorithm extends the contour line in both the sides until it cannot extend it further given some constraints which are explained below.

$$w(p_i, p_j) = \min\{w_d(p_i, p_j), w_{\alpha_i}(p_i, p_j), w_{\alpha_j}(p_i, p_j)\},$$

$$w_d(p_i, p_j) = \begin{cases} 1 - \frac{|p_i p_j|}{d_{\max}} & \text{if } |p_i p_j| < d_{\max} \\ 0 & \text{otherwise,} \end{cases}$$
$$w_{\alpha_x}(p_i, p_j) = \begin{cases} 1 - \frac{\alpha_{ij}^x}{\alpha_{\max}} & \text{if } \alpha_{ij}^x < \alpha_{\max} \\ 0 & \text{otherwise.} \end{cases}$$

The initial segment of the new path is determined by a pair of isolated points (p_i, p_j) within the threshold distance d_{MAX} , such that the weight w(p_i, p_j) is the maximum. Next, the path is extended at both ends by repeatedly adding points until there are no more candidates or the added point is already part of some path before being added; When extending a path P = (. . . , p_{i-2}, p_{i-1}, p_i) from the end point p_i, the algorithm takes the point p_x with the best weight and that satisfies the following conditions: (1) the distance from p_x to p_i is at most d_{MAX} , and (2) the turn angle from $p_{i-1}p_i$ to p_ip_x is at most $\pi/2$. Fig. 4.1 provides an example of the point with the best weight being selected from among p_a, p_b, and p_c, since p_d and p_e do not satisfy the given conditions.

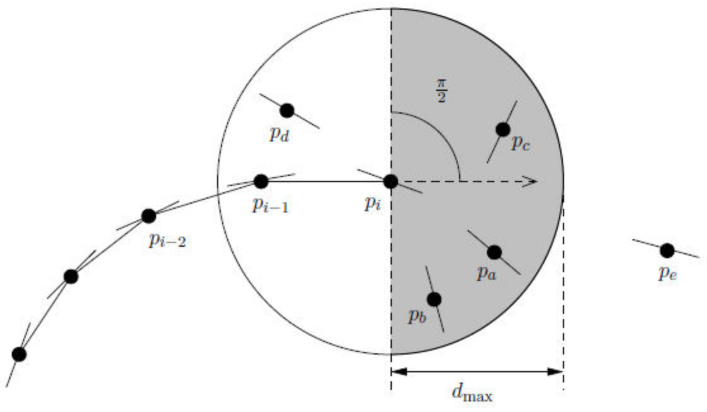


Fig 4.1 While extending the path, only points in the grey area are considered

The weight $w(p_i, p_j)$ of a pair of points (p_i, p_j) is determined by the distance between them $|p_ip_j|$ and the difference between their orientations and the orientation of the segment p_ip_j. Therefore, the weight depends on two parameters d_{MAX} and α_{MAX}. We use a function that decreases when the distance or the differences between the orientations increase, and that has a minimum value of 0 when the distance or the differences between the orientations are greater than or equal to d_{MAX} or α_{MAX}. Using this function, we link pairs of points if their weight is greater than 0.

We next describe the weight function. Let $\mathbf{a}^{\mathbf{i}}_{\mathbf{ij}}$ be the difference between the orientation of point p_i and segment p_ip_j; and let $\mathbf{a}^{\mathbf{j}}_{\mathbf{ij}}$ be the difference between the orientation of point p_j and segment p_ip_j. We determine a weight for the distance between the points $|p_ip_j|$ and for each of the orientation differences $\mathbf{a}^{\mathbf{i}}_{\mathbf{ij}}$ and $\mathbf{a}^{\mathbf{j}}_{\mathbf{ij}}$. These weights are higher when the values are smaller, and therefore, we take the minimum of the three values as the weight for the pair of points. Intuitively, this is the value for the worst of the three: the distance and each of the orientation differences.

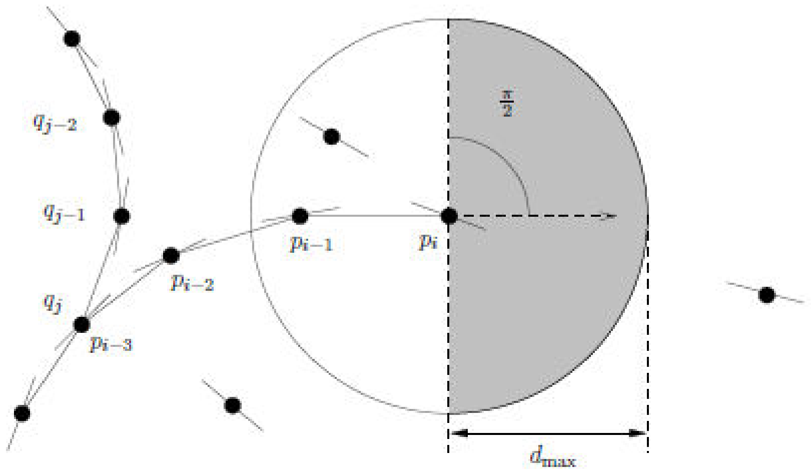


Fig 4.2 As there is no point in the grey region, the path can not be extended further

4.2 Output

As this stage links the oriented points, we get paths (contours) as output of this stage.

Chapter 5

Experiments and Results

There are four main tuning parameters upon which the final contours depend. In this section we include different experimentations that we carried out by altering these parameters and present the corresponding observations. The four tuning parameters are given below

The data set used in experiments is BSDS300 (Berkeley Segmentation Data Set).

T_par(Edge detection stage)

For removing weak edges after edge detection process we use threshold as described in equation **(2.3) .** This equation is adaptive. By changing this parameter different classes of images gets better results.

Threshold(I) = mean(Grad(i,j)) + (255 - mean(Grad(i,j)))/t_par

C_c (Clustering stage)

In order to assure minimum distance between two points after clustering, we provide $\mathbf{C_c}$ accordingly**.** By changing this parameter we can control the number of points for the next stage i.e, linking.

d_max = $\mathbf{C_c}$ *d_min

C_L (Linking stage)

In order to define the maximum proximity between points to get linked in the linking stage, we provide $\mathbf{C_l}$ accordingly.

d_max = $\mathbf{C_l}$ *d_min

Alpha$_L$(Linking stage)

This parameter deals with the allowable difference of orientation of the points to be linked. If orientation difference is more than this parameter the points are not linked.

5.1 How T_par affects ?

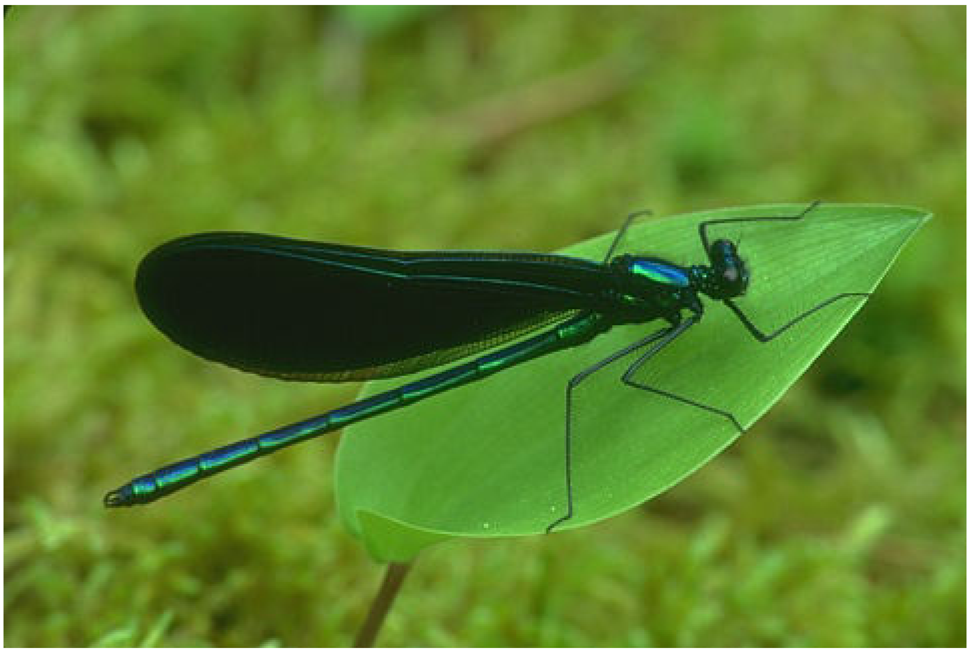


Fig 5.1.1 Original Image

The threshold varies inversely with respect to t_par as per the given equation. If the edges of interest are weak then it is quite possible that we might loose them in the first stage itself. This is more likely to happen when the variance of the gradients is less and we loose the edges of interest. In another case, if the edges of interest are strong but the mean is low, then the output might come out to be noisy. In order to prevent this the threshold should be increased with help of T_par.

Below here it is explained with help of an example how T_par affects the final output and can be used to improve the same. As it can be seen in the images that for higher threshold (lower T-par) the top of the leaf is thresholded so it is not linked in the later stage (Fig 5.1.2 and Fig 5.1.4). When we decrease the threshold (higher T_par), result gets improved, although this might lead to increment in noise in some cases.

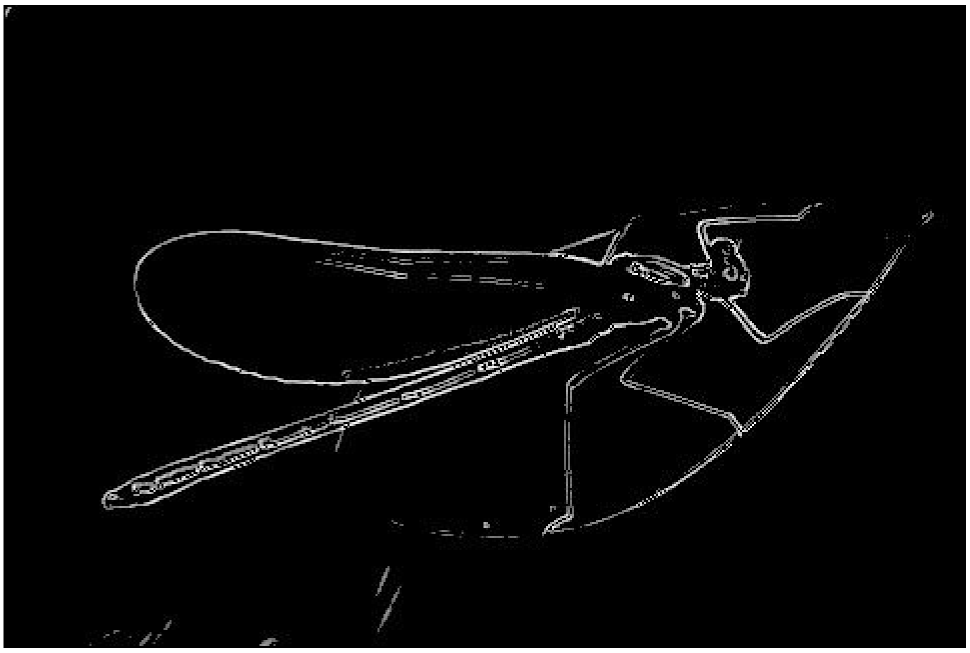


Fig 5.1.2 Gardient image for t_par = 3

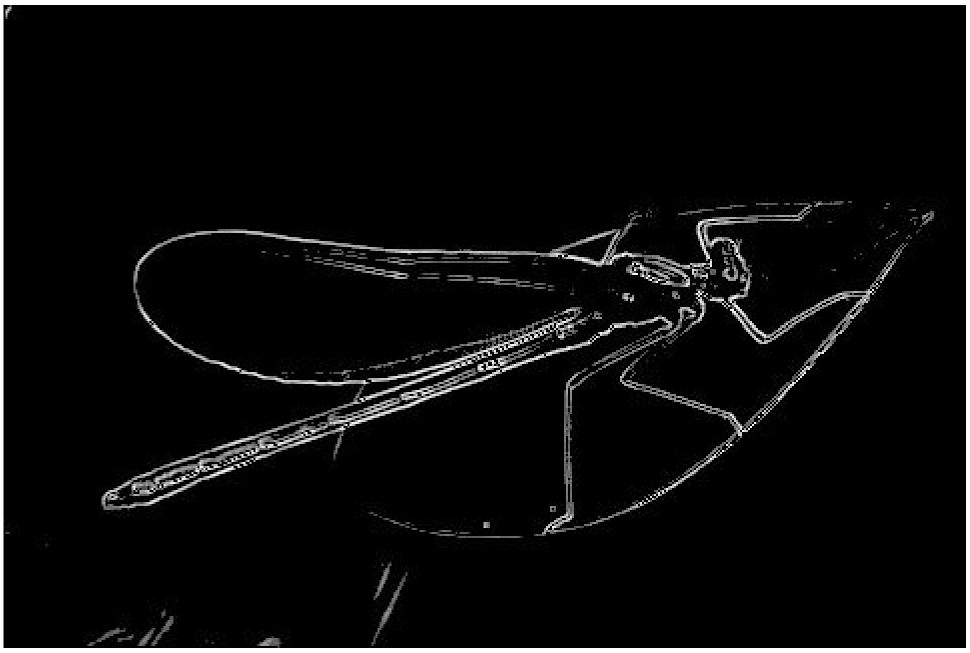


Fig 5.1.3 Gardient image for t_par = 4

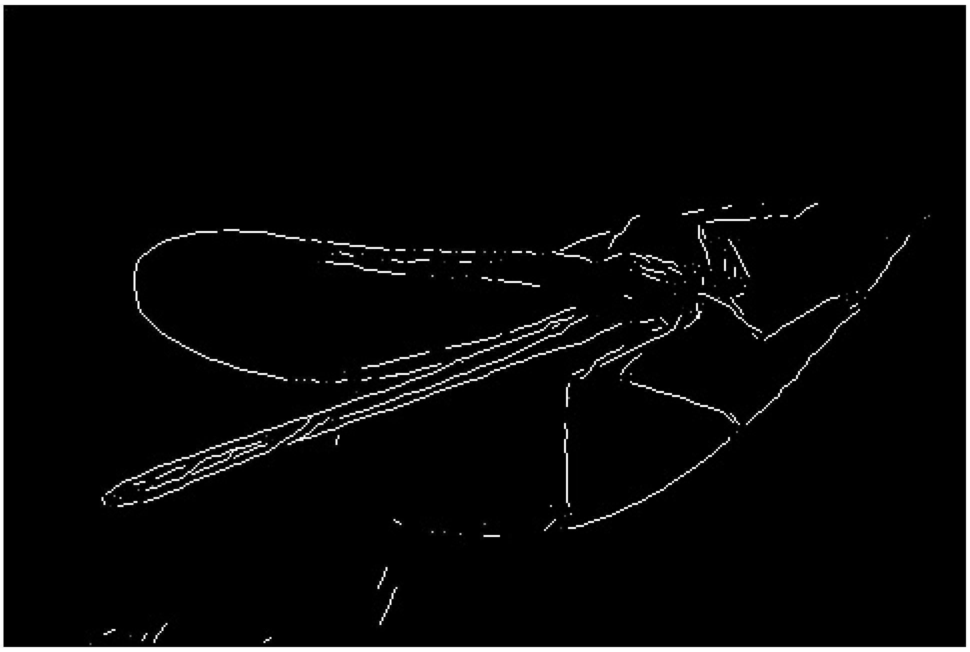


Fig 5.1.4 Linked image for T_par = 3

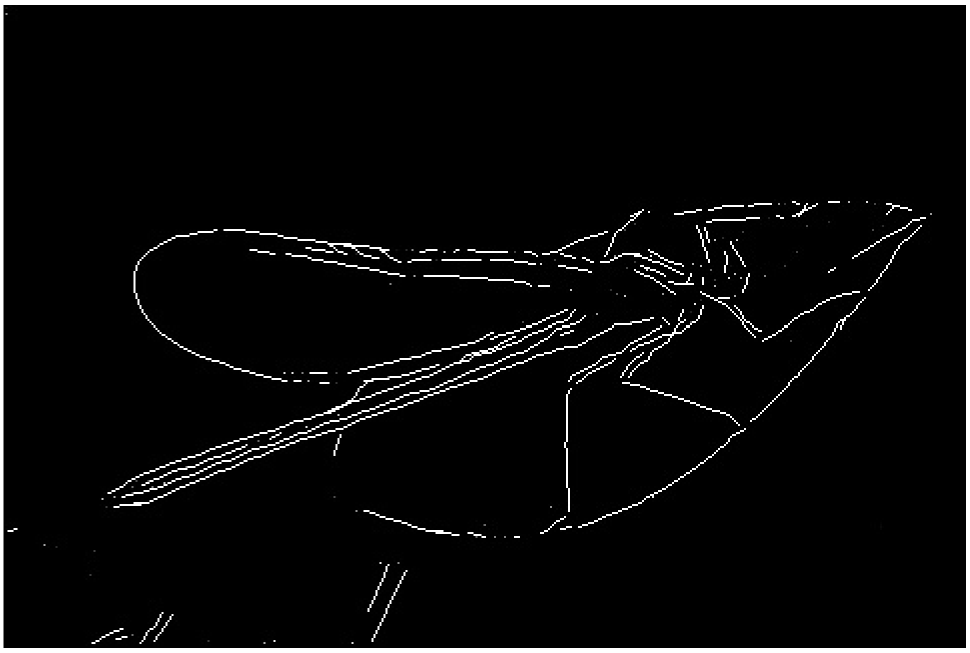


Fig 5.1.5 Linked image for T_par = 4

5.2 How C_c affects ?

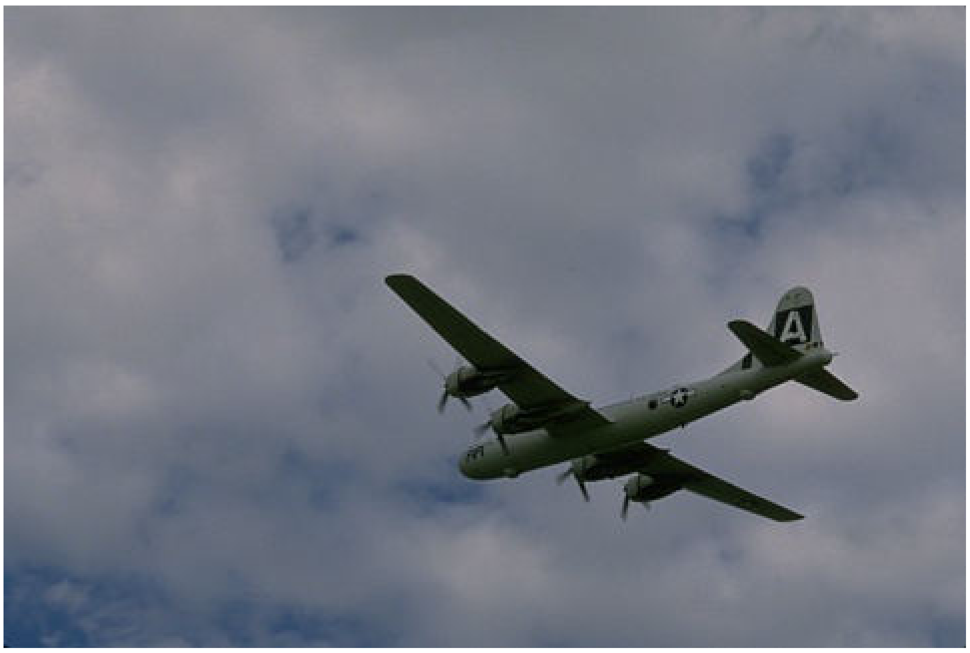


Fig 5.2.1 Original image

The purpose of the clustering is to reduce the number of points obtained by the output of edge detector, at the same time removing the redundant point representations of the same contour. This can be achieved by maintaining dmax, which depends on C_c.

We experimented with different values of C_c on an image. As it can be seen from the results that for lower C_c the redundancy is more, at the same time number of points are more than required. While in case of higher values of C_c, the basic structure is compromised.

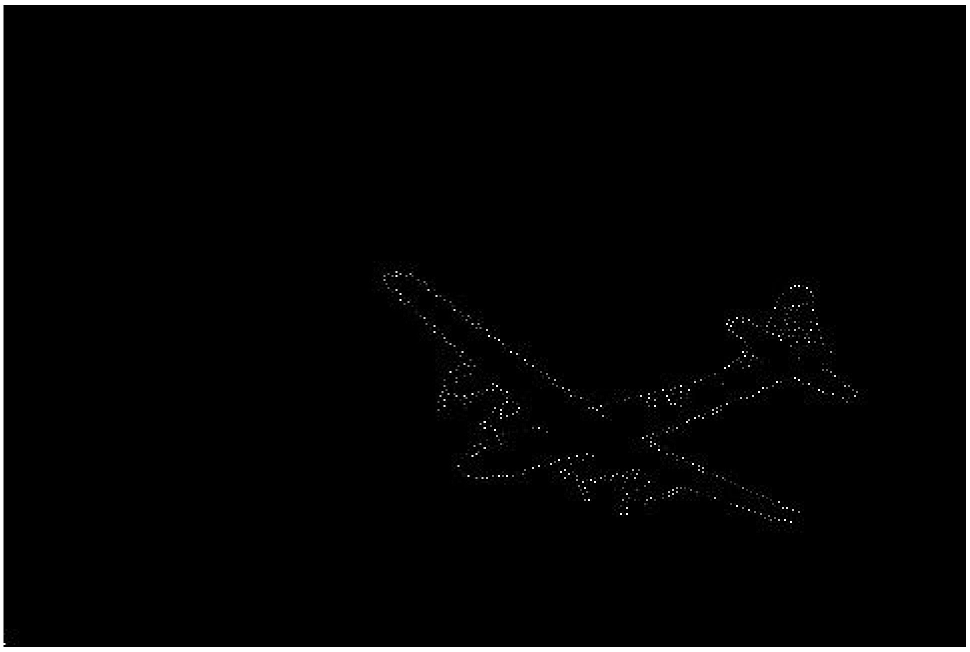


Fig 5.2.2 Clustered image with $C_C = 2$ (554 points)

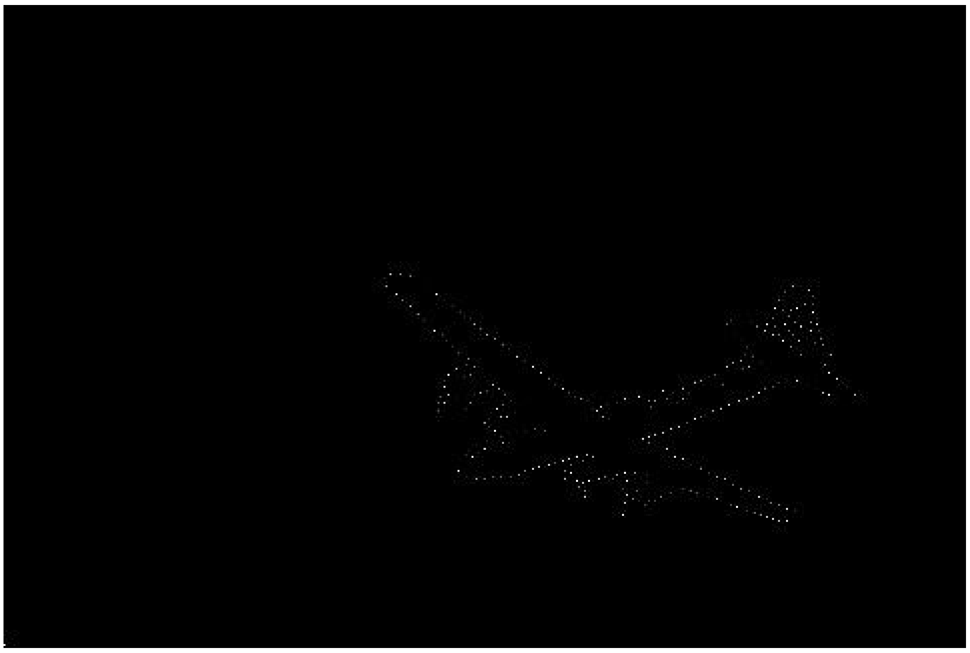


Fig 5.2.2 Clustered image with $C_C = 3$ (298)

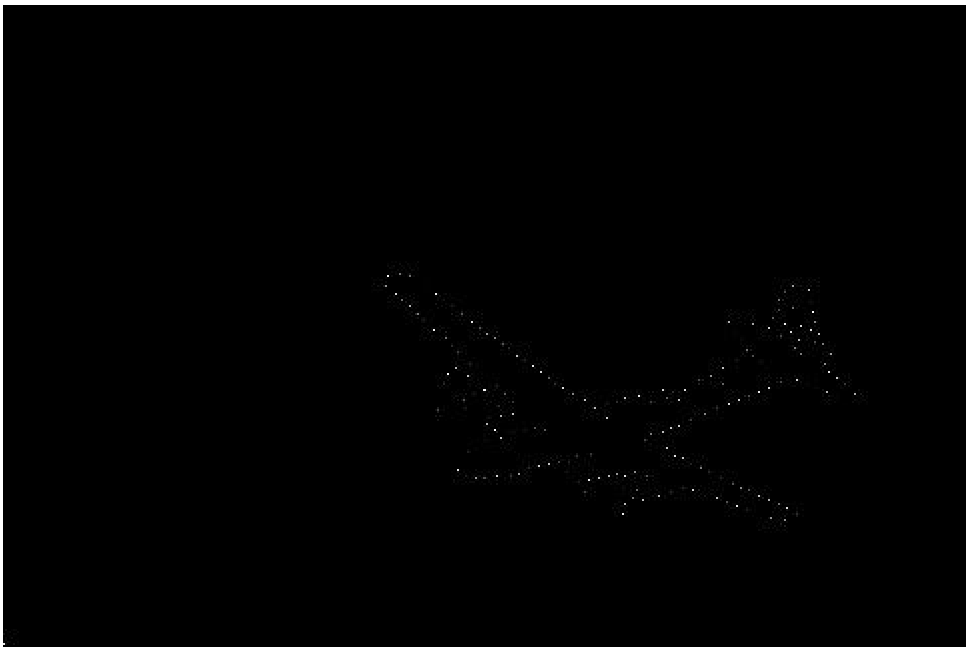


Fig 5.2.2 Clustered image with $C_C = 4$ (202)

5.3 How C_L affects?

It is the measure of proximity for two points to be linked. For lower values of C_L there might exist discontinuities in a contour. This is possible when the edge is weak on some specific portion of the contour because of which less number of edge points represent that portion distance between two points to be linked is more than dmax. For higher values of C_L wrong points are linked at times.

In Fig 5.3.1 ($C_L = 3$), discontinuities are found on the wings of aeroplane. While in case of Fig 5.3.2 ($C_L = 4$) those discontinuities are filled. But going further, for $C_L = 5$ (Fig 5.3.3) we can find some noisy contours, which is undesirable.

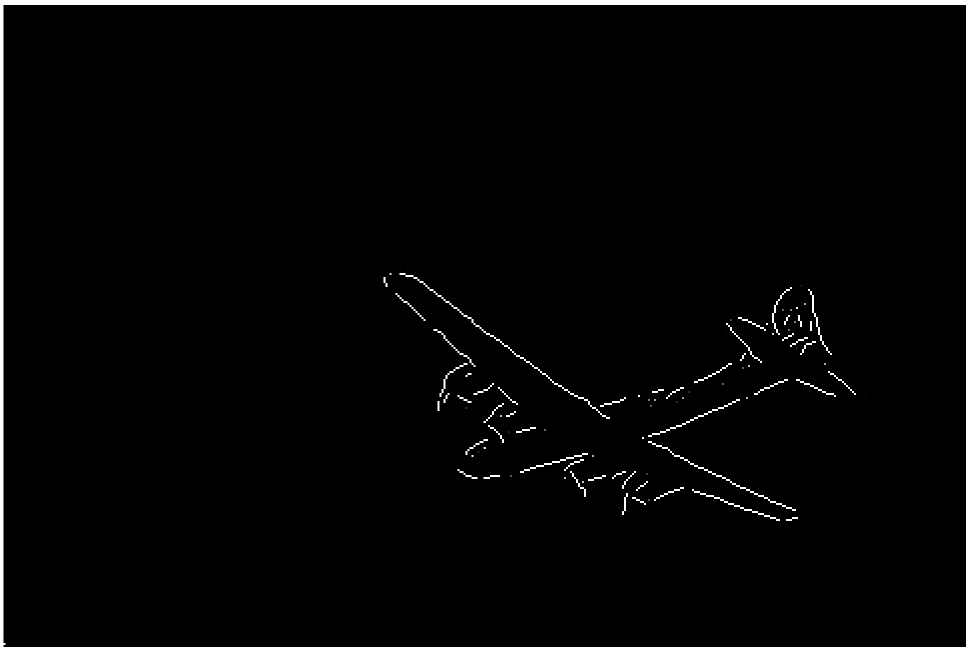


Fig 5.3.1 Linked image with $C_L = 3$

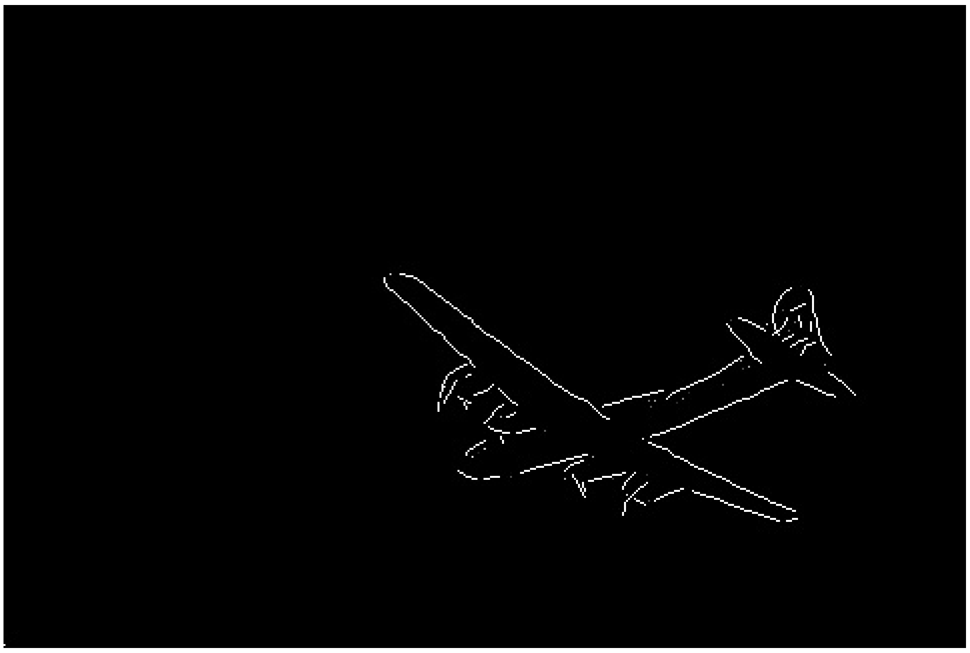


Fig 5.3.2 Linked image with $C_L = 4$

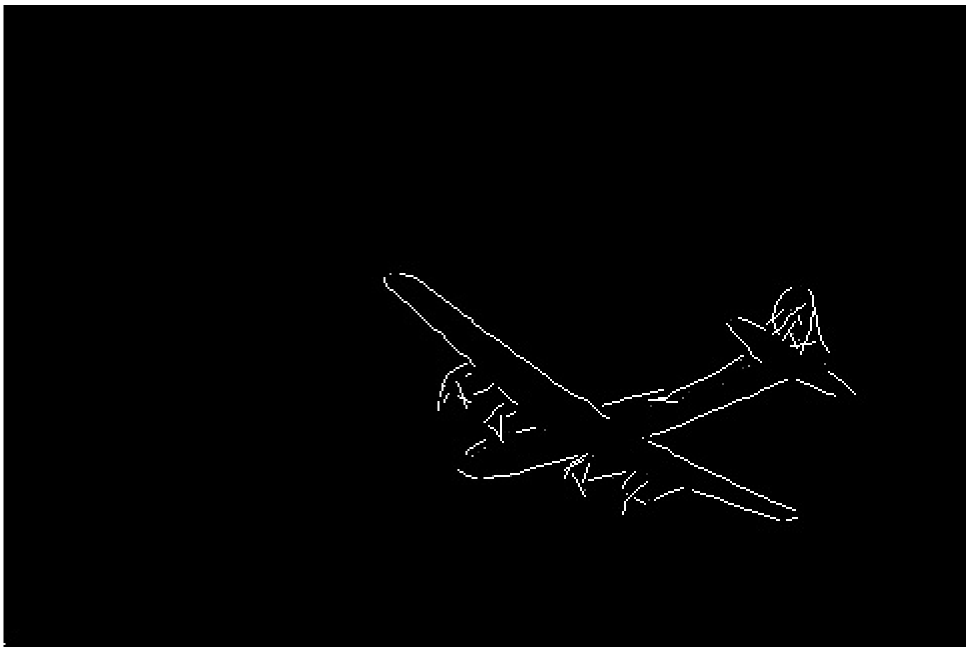


Fig 5.3.3 Linked image with $C_L = 5$

5.4 How Alpha$_L$ affects?

Alpha$_L$ is used as a higher bound for the different points to get linked on the basis of their orientations. Lower values of Alpha$_L$ will lead to discontinuities in the contour with curvature parts in it. While in case of higher values of Alpha$_L$ false contours are often formed .

Values of Alpha$_L$ are varied from 35 to 45 and finally 60 in the figures given below. It can be observed in the cockpit region of the aeroplane that there are discontinuity in the first image (Alpha$_L$ = 35), which is partially filled in the second image (Alpha$_L$ = 45). But for the third image (Alpha$_L$ = 60), false contours can be observed.

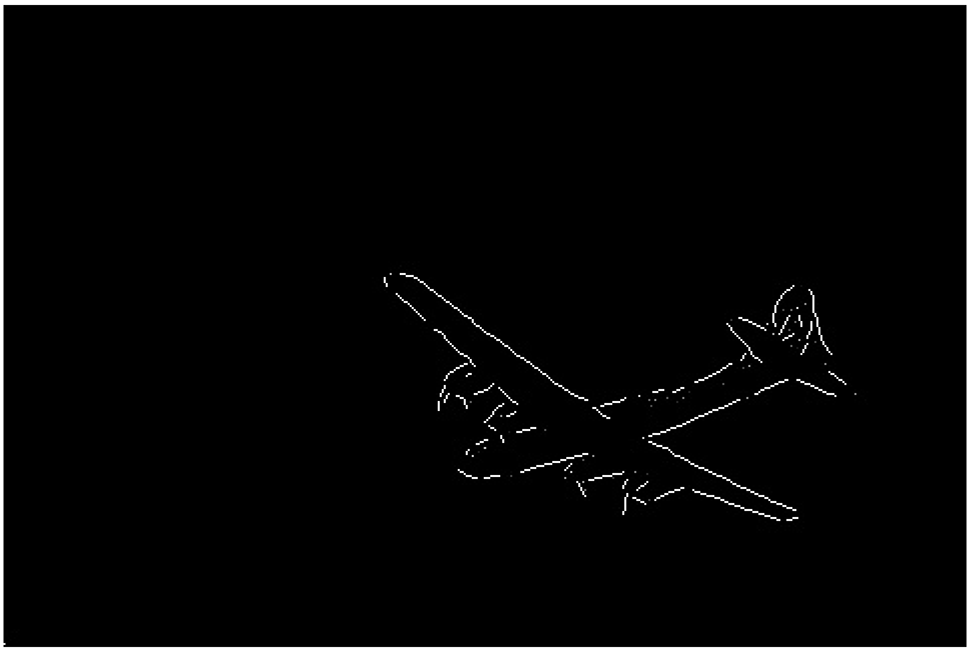


Fig 5.4.1 Linked image for Alpha$_L$ = 35

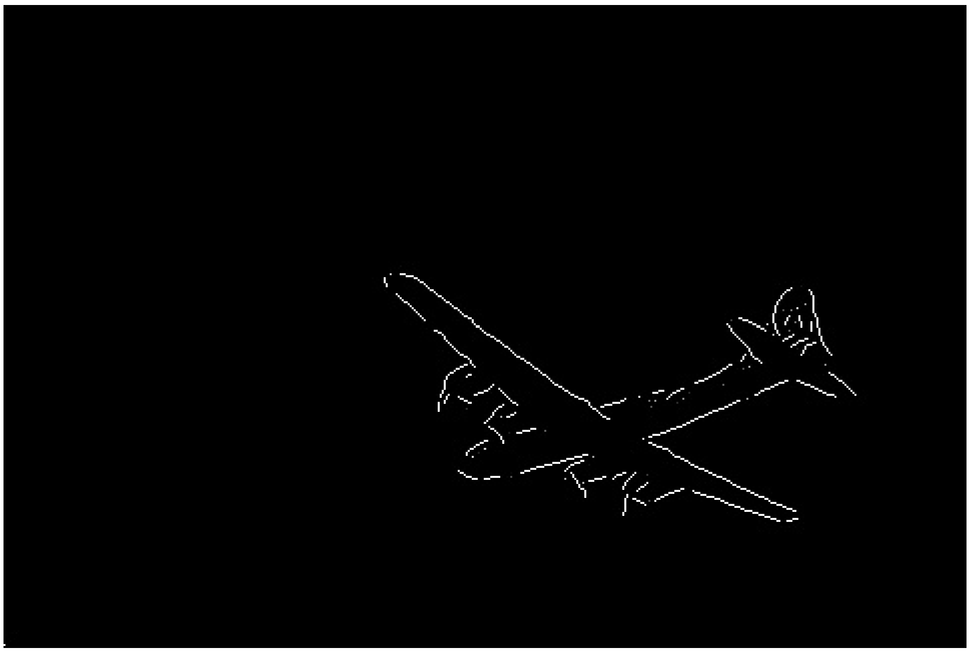


Fig 5.4.2 Linked image for Alpha$_L$ = 45

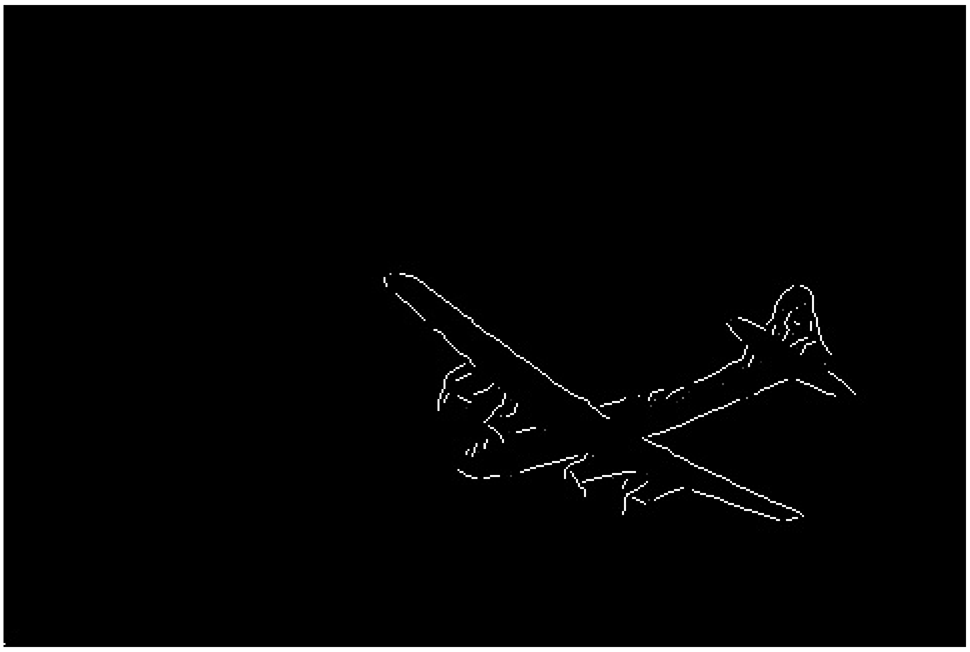


Fig 5.4.3 Linked image for $\text{Alpha}_L = 60$

5.5 Results

Image A

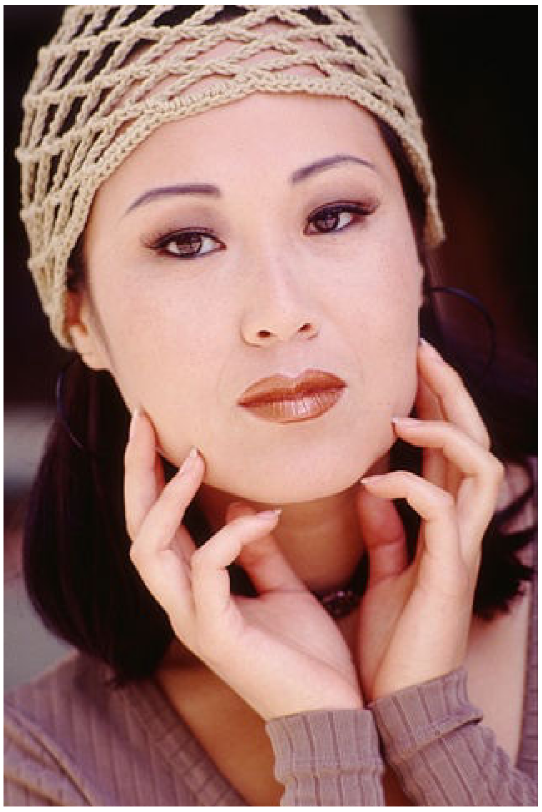


Fig 5.5.1.1 Original Image

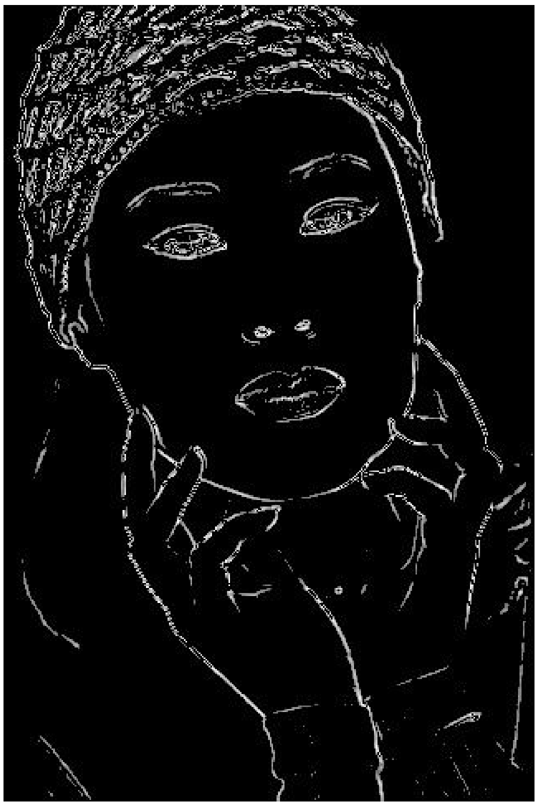


Fig 5.5.1.2 Gradient Image with 12,312 points

With T_par = 3

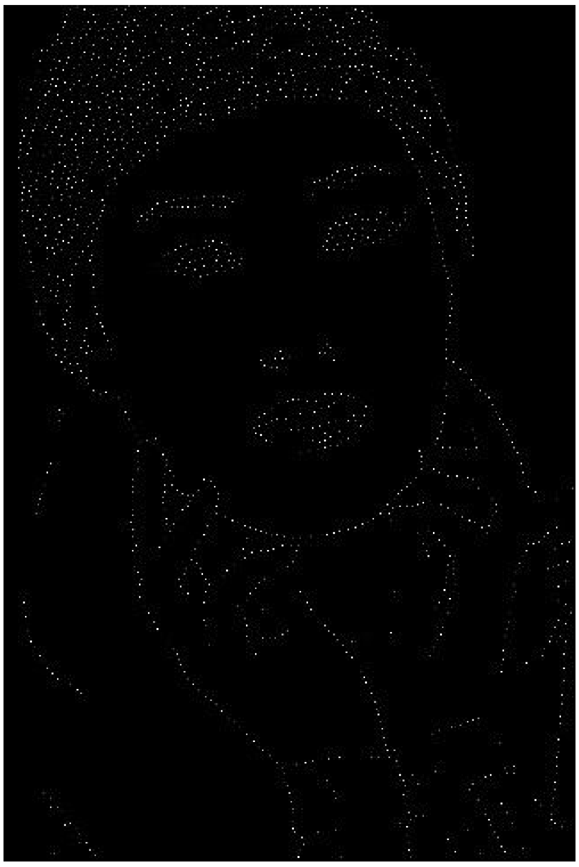


Fig 5.5.1.3 Clustered Image with 1597 points (C_C = 3)

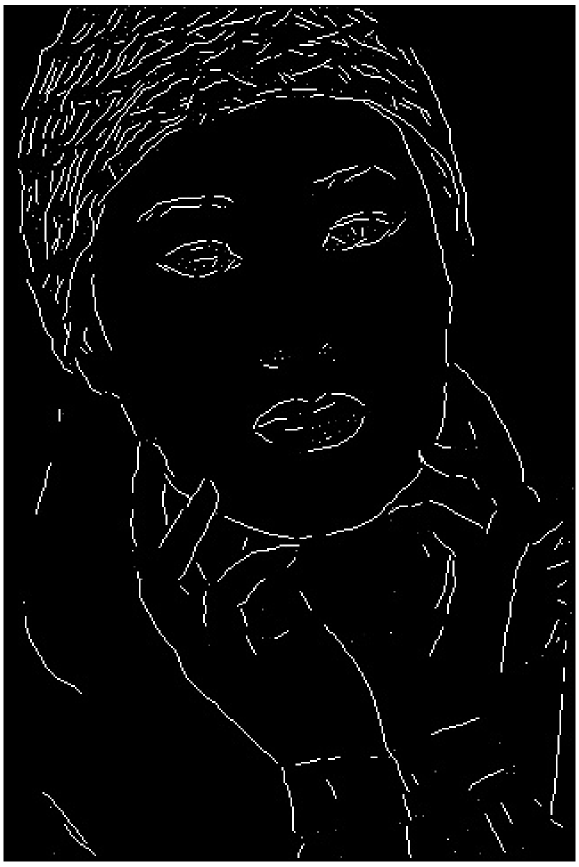


Fig 5.5.1.4 contour image with 527 paths (C_L = 4 , $Alpha_L$ = 35)

Image B

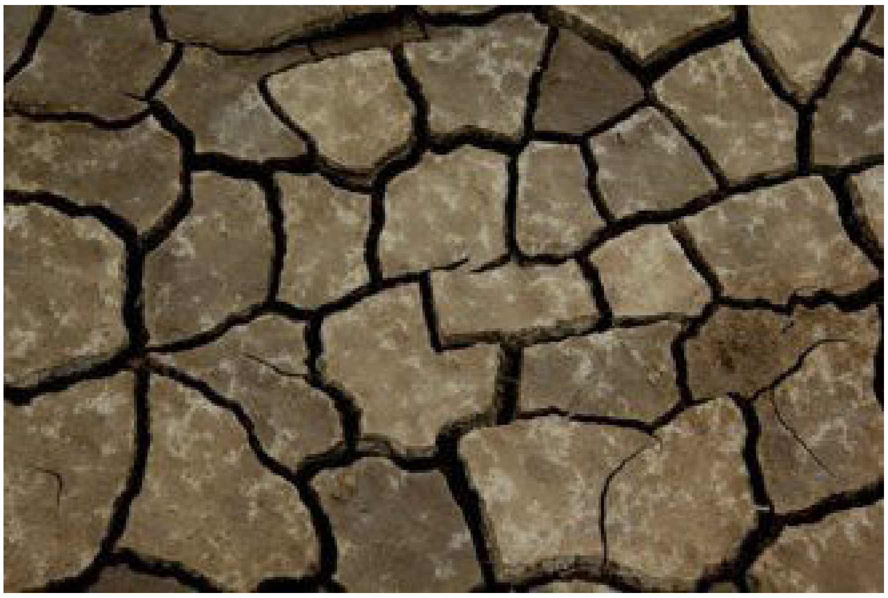


Fig 5.5.2.1 Original image

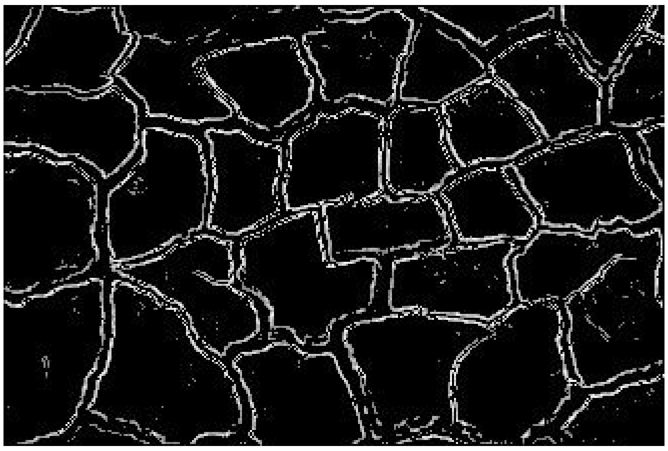


Fig 5.5.2.2 Gradient image with 8,482 points

With T_par = 3

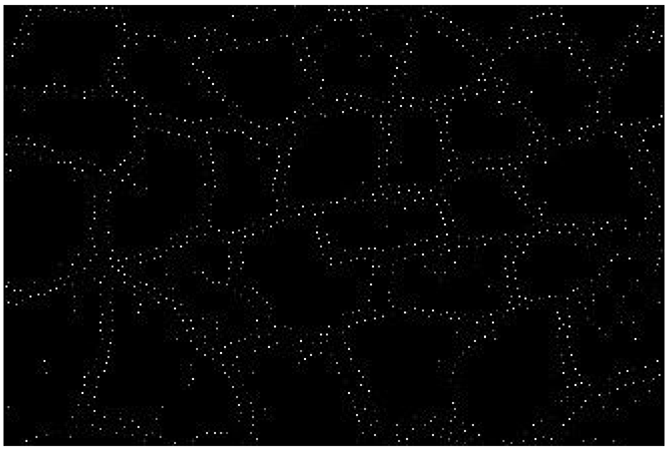


Fig 5.5.2.2 Clustered image with 1,291 points

(C_C = 3)

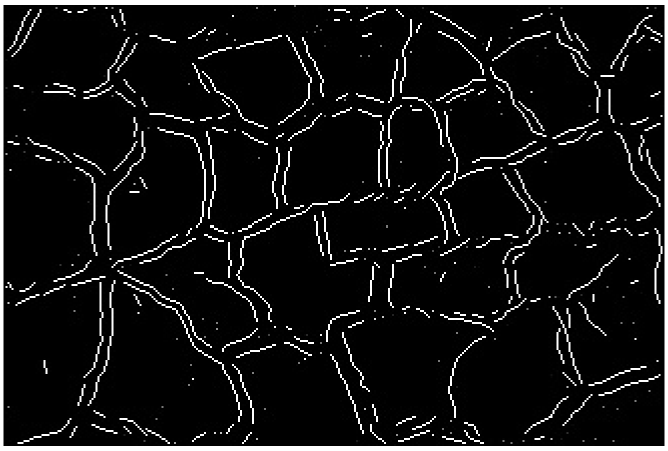


Fig 5.5.2.3 Linked image with 390 paths(C_L = 3 , Alpha$_L$ = 45)

Image C

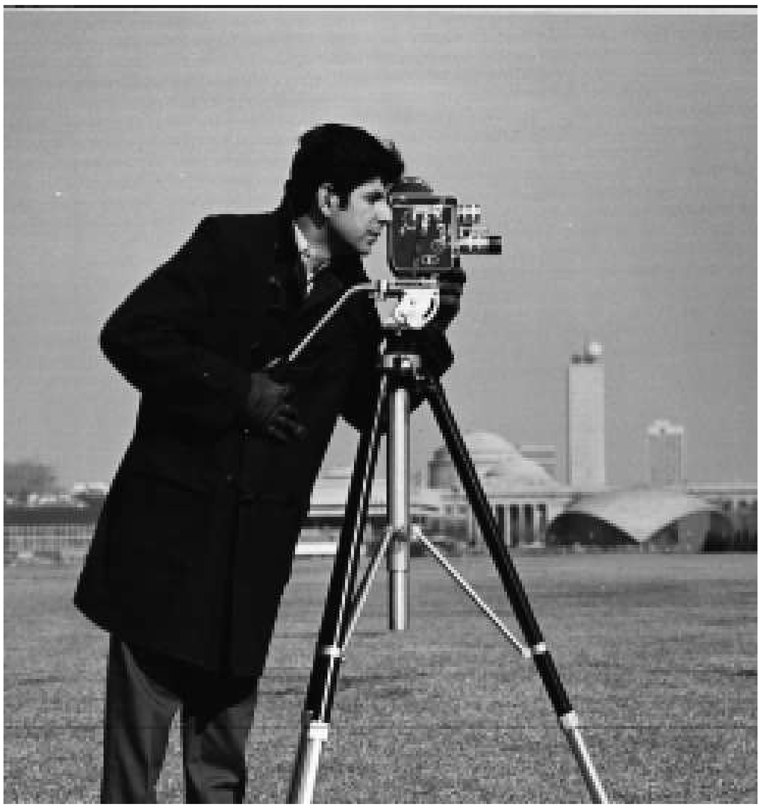


Fig 5.5.3.1 Original image

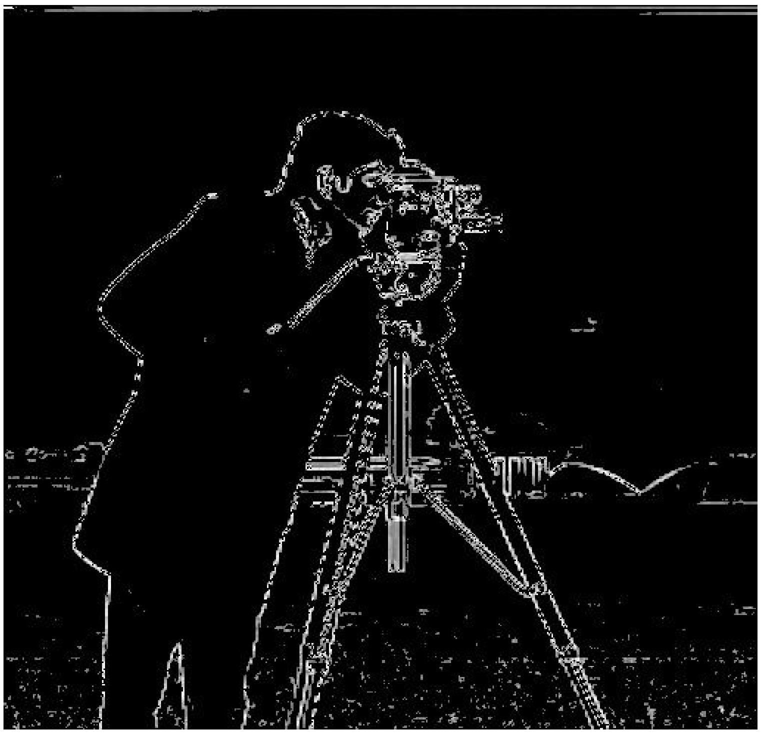


Fig 5.5.3.2 Gradient image with 16,879 points (T_par = 3)

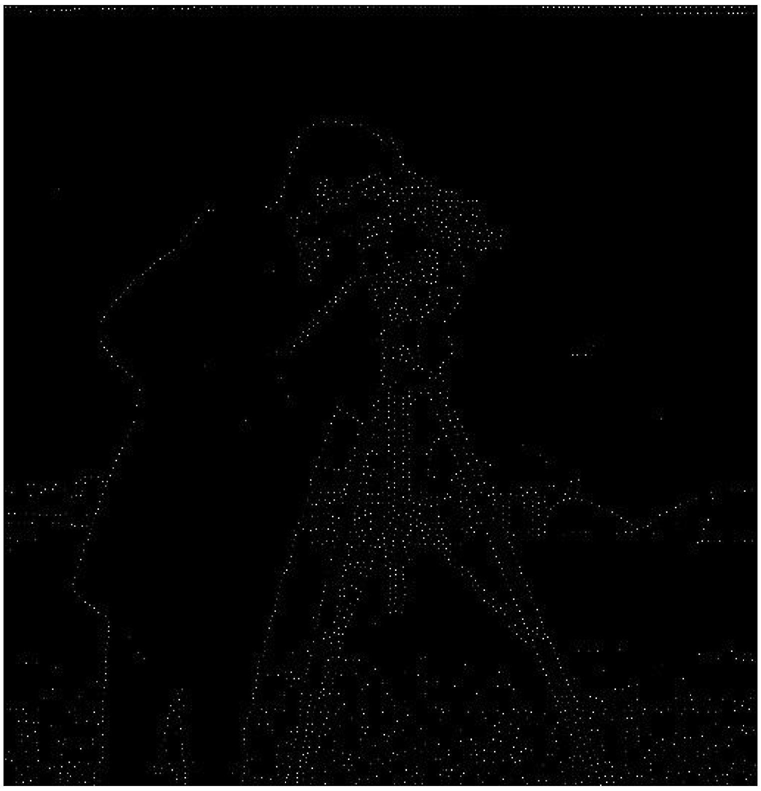


Fig 5.5.3.3 Clustered image with 2042 points ($C_C = 3$)

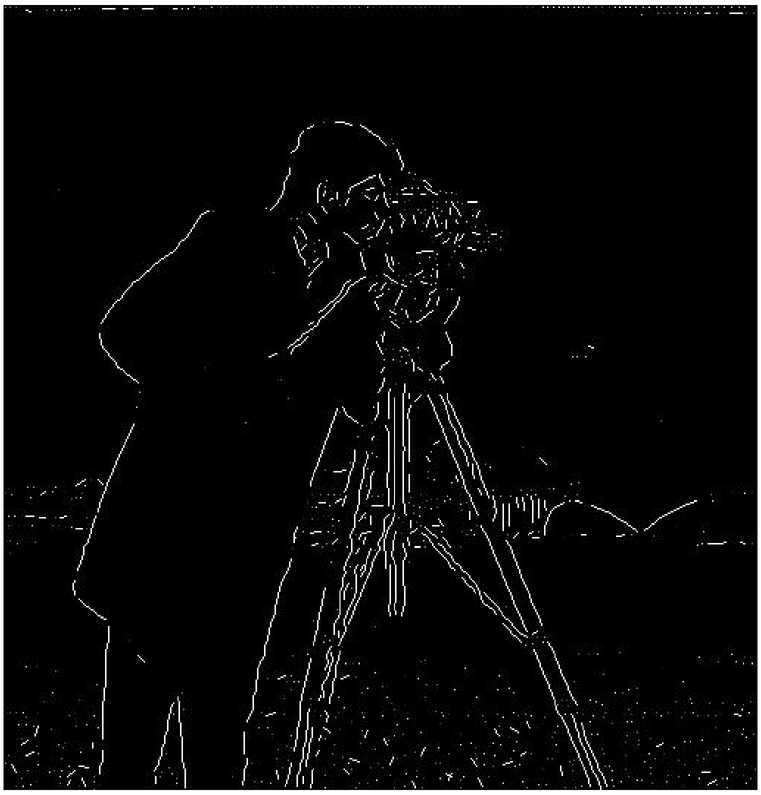


Fig 5.5.3.3 Clustered image with 1097 paths ($C_L = 3$, $Alpha_L = 35$)

5.6 Work Done and Work Division

We have implemented the mentioned thesis except for the last part (which is not mandatory to find contours) and obtained satisfactory results. Apart from that we experimented on different images of BSDS300 and some of our own images by altering all the possible tuning parameters. In terms of Modules

Module 1 – Finding oriented edge points	(Bipin)
Module 2 – Clustering & Testing	(Falak)
Module 3 – Linking & Integration of Modules	(Teja)

Chapter 5

Conclusion

- Optimum value of T_par is found to be between 3 and 4. For images in which the variance of the gradient is low, higher T_par is desirable. For images with higher variance, lower value of T_par is desired.
- From the above point, we propose what could be an improved way of determining the threshold of the gradient. In the thesis [1], the formula for an image is given as in equation 2.3
  - **Threshold(I) = mean(Grad(i,j)) + (255 - mean(Grad(i,j)))/3**
- Instead of the above equation, we first introduced T_par, which worked better. But this doesn't select the threshold automatically.
  - **Threshold(I) = mean(Grad(i,j)) + (255 - mean(Grad(i,j)))/T_par**
- After that, a new method is proposed for determining value of threshold which is adaptive, which could provide better results. This method utilizes the standard deviation of gradients of edge points. The formula for threshold is given as
  - **Threshold(I) = mean(Grad(i,j)) + (1.5) * (std deviation of grad)**
- Optimum value for C_C is found to be 3 to 4. As C_C is increased from optimum value, basic structure is often lost in case of contours with curvature, although it could provide better results for contours with long straight regions.
- While on the other side, if C_C is less than the optimum value, number of points may be more than required, which makes it less efficient for linking stage. Also we may get double contour lines in some cases.
- Optimum value for C_L is found to be 3 to 4. Lower values of C_L can lead to broken contours. As it gets higher, contours become closed and higher values of C_L can get false contours (noise).
- Optimum values of $Alpha_L$ are found to be 30 to 50 degree. For lower values of $Alpha_L$ contours with curvature may be found to be broken. For values of $Alpha_L$ that are much higher, false contours are formed.
- General practice for choosing C_L and $Alpha_L$ is, one of them is kept on lower side and the other one is kept on the higher side. This is done in order to improve result without introducing much of noise. Although it would depend on the content of the image.

References

[1] Tejada, Pedro J., "A Computational Geometry Approach to Digital Image Contour Extraction" (2009). *All Graduate Theses and Dissertations.* Paper 422.

[2] R.C. Gonzalez and R.E. Woods, *Digital Image Processing,* 3rd ed. Upper Saddle River, NJ: Pearson Prentice Hall 2008.