

---

## PROPOSAL AND DESIGN

In this chapter, the objectives of the project and the techniques for achieving them are identified. In particular, the reconstruction problem will be aimed at the removal of image defects such as cracks, scratches or blotches. Sections 3.1, 3.2 and 3.3 outline the main proposal while the remaining sections deal with the coarse-grained design of the system.

### 3.1 Problem Statement

Images are often afflicted by different types of defects such as cracks, scratches and blotches (including semi-transparent blotches). Moreover, the presence of objects like subtitles, logos, wires, and microphones are sometimes unwanted in images. Therefore, these images need to be restored without leaving any artifacts such that a user, who has not seen the original image, is not aware that any modification has occurred.

### 3.2 Objectives of the Project

The objective of this project will be thus to restore the different types of image defects that occur in images. For this purpose, we have divided these defects into three main categories according to their characteristics:

1. Semi-transparent blotches
2. Line defects such as scratches or straight cracks
3. Blotches, spots and other small unwanted (non-transparent) objects

Consequently, each technique that will be proposed will have as aim the restoration of one of these categories of defects. It should be noted although the term ‘restoration’ is being used throughout this section, the intending aim is mainly an inpainting effect i.e. after the defect has been removed, the user should not be aware that a defect was present originally in the image, even though we do not exactly get the original image. However, in cases where original information is available to some extent (like in semi-transparent blotches), we can make use of this information to recover the original image. Figure 3.1 shows a diagram that illustrates the system proposed.

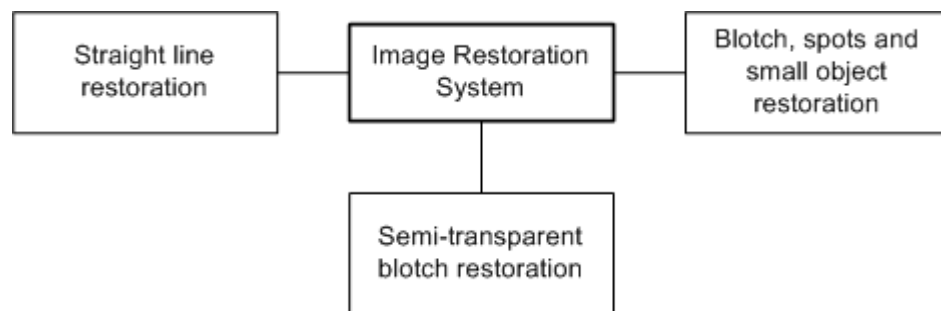


Figure 3.1: Diagram showing the system proposed

### 3.3 Proposal

For any of the three categories of defects mentioned in the previous section, the general restoration techniques like the Pixon method or the Richardson-Lucy technique are not suited for removing the type of defects being targeted. These methods aim to restore blurry or noisy images where the degradation needs to be mathematically modeled in advance.

**Semi-transparent blotches:** Restoration of semi-transparent blotches will be performed by the technique developed by Stanco *et al.* [STRT03]. This technique attempts to preserve available data under the semi-transparent blotch in the original image. Here, the use of inpainting techniques is not appropriate since these techniques do not take into account any of the information present in the region to be restored.

**Line defects:** The Line Scratch Correction Algorithm (LSCA) can remove unwanted lines from images. However, as mentioned in the analysis, LSCA restores only vertical lines, therefore, a more flexible method, based on LSCA, to restore lines running in any direction shall be used.

**Blotches, spots and other small unwanted (non-transparent) objects:** For this type of defect, an inpainting technique would be appropriate since the defects can be considered as missing data in the image. Inpainting techniques like [BERT00] or [OLIV01] use a diffusion process to propagate information from the surrounding area into the region to be inpainted. But if the surrounding information is completely different from the missing area, visible artifacts will be detected by the user. The image noise removal algorithm [HIRA96a] will thus be used as it seems more suitable since the user can select where the information that will be used for inpainting comes from.

Moreover, other criteria have been taken into account for choosing the above techniques. The selected techniques use algorithms that can be implemented within the time constraints imposed on the project, and the steps involved are easier to understand and implement since the underlying mathematical concepts are less complex. Also, the speed of execution of these methods is acceptable as compared to other methods which take several minutes to execute and/or require specialized hardware, extensive memory and processing capabilities.

### **3.4 Image Format**

The above mentioned restoration techniques will be applied not only to grayscale images but also to color images. LSCA and the semi-transparent blotch removal algorithm have been specifically developed for vintage films and photographic prints, and were tested only on grayscale images. However, over time, old photographs may turn into a brownish color; therefore, it would be useful to use both grayscale and color images for testing the restoration methods. Two different image formats have been identified to be used for this project:

- (i) The Portable GrayMap (**PGM**) format for grayscale images and
- (ii) The Portable Pixmap (**PPM**) format for color images.

**(i) PGM Images**

A PGM file consists of two parts: a header and the image data. The header consists of at least three parts normally delineated by carriage returns and/or linefeeds. The first 'line' is an identifier, it can be either 'P2' or 'P5'. 'P2' refers to the ASCII form and 'P5' refers to the binary form of the data. The next line consists of the width and height of the image as ASCII numbers. The last part of the header gives the maximum graylevel that a pixel can have. In addition to the above required lines, a comment can be placed anywhere with a '#' character, the comment extends to the end of the line. The image data stores grayscale information - one value per pixel. Figure 3.2 shows an example of a PGM header.

```
P5
#Author - Veronique Leung
256 256
255
```

**Figure 3.2:** Example of a PGM Header

**(ii) PPM Images**

The PPM format is similar to the PGM format except that it can store more than one value per pixel. PPM images use the RGB format i.e. the image data values are stored in the order red, green and blue. Also the identifiers for this format are either 'P3' (ASCII) or 'P6' (binary).

**3.5 Defect Detection**

As pointed out in [STAN03], the general restoration problem needs to be tackled in two steps: detection of the defect and actual restoration. To remove whatever defect present in an image, we need to localize the defect first and then based on this information, the restoration process can take place. However, these defects are very difficult to automatically detect since there are no simple rules to distinguish them from real image features [STRT03]. Therefore, for all the three methods that have been proposed, the user will manually identify the damaged area i.e. the user himself needs to define the noise mask.

For most restoration methods to work efficiently, the user should be able to identify the noisy regions to be restored as precisely as possible. In the case of LSCA, defect identification by the user is straightforward, since the noise is simply a straight line of a certain thickness. On the other hand, manually building a noise mask for other types of defects is a tedious and time-consuming process [OLIV01], since most defects present themselves as irregularly shaped objects in an image with no particular or regular attributes. Thus, the proper image processing and editing tools need to be provided to make the mask construction simpler, especially where we need to identify blotches, cracks, or other irregularly shaped objects. Consequently, both an automatic and a manual technique for noise selection will be provided to the user to build the noise mask.

### **3.5.1 Automatic Noise Selection**

The user should be able to select any pixel within an object and consequently, the whole object should be labeled as noise. However, this method will work well only for objects which have regions of uniform (or nearly uniform) intensity, but with image defects which present varying intensities, only part of the defect will be identified at one go. The whole process needs to be repeated several times for all the relevant pixels to be identified as noisy pixels. It can also happen that non-noisy pixels are automatically and wrongly identified as noise in the selection process.

### **3.5.2 Manual Noise Selection**

As mentioned in the previous section, automatic labeling may sometimes need a high number of iterations before the whole defect is identified. Hence, to simplify the noise detection procedure, a 'paintbrush' tool can be used by the user to more precisely select the noisy pixels. The brush should also be available in different sizes so that smaller regions like edges can be more accurately identified with small brushes, while larger brushes provide a quick and rough identification for large areas.

Also, to deselect any pixel that has been wrongly identified as a noise pixel, an 'eraser' tool should be provided. This eraser tool should work in the same way as the paintbrush tool, providing different size options, except that it changes the labeling

of noisy pixels to non-noisy. The paintbrush and eraser tools are meant to work in conjunction with the automatic labeling method to make the identification of the noisy areas as simple and quick as possible for the user.

### 3.6 Semi-Transparent Blotch Restoration

Semi-transparent (or water) blotches are special cases of image defects since they preserve part of the original information. Therefore any restoration process should make use of this information so that as much of the original image is recovered as possible. The algorithm developed by Stanco *et al.* [STRT03] is a technique for removing water blotches. This method makes use of information surrounding the blotch but also considers the information inside the blotch for restoration. The user needs to manually identify the region inside the blotch and this region should be precisely selected for the restoration process to provide accurate results.

The restoration process is done in two steps. The first step (step 1) consists of restoring pixels inside the blotch using an additive-multiplicative model, while the second step, which is itself split into two steps (step 2a and step 2b), consists of refining the contour area. The second step is needed because most water blotches have a darker border region as compared to the region inside the blotch. This is due to the fact that dust and dirt are dragged towards the border as the water diffuses in the paper.

#### 3.6.1 The Additive-Multiplicative Model

This model can be represented as:

$$J(\Omega_{x,y}) = \alpha * I(\Omega_{x,y}) + \beta \quad - (3-1)$$

where  $J(x,y)$  is the actual image data

$I(x,y)$  is the uncorrupted image data and

$\Omega_{x,y}$  is the region to restore.

$\alpha$  and  $\beta$  are parameters to be determined. Therefore, if we can compute values for  $\alpha$  and  $\beta$ , then equation (3-1) can be inverted to get an estimate of the original uncorrupted image.

Let  $Var [.]$  denote the variance and  $E [.]$  the mean in  $[.]$ . From equation (3-1), we can deduce that

$$\begin{aligned} Var[J(\Omega_{x,y})] &= \alpha^2 Var[I(\Omega_{x,y})] \\ E[J(\Omega_{x,y})] &= \alpha E[I(\Omega_{x,y})] + \beta \end{aligned}$$

However, at this stage, values for the mean and variance of the uncorrupted image  $I(x,y)$  are not available. Therefore, we need to find an estimate for  $I(x,y)$  i.e.  $I(\Omega_{x,y})$  is approximated with  $I(\Omega_{x,y}^1)$ . Using  $I(\Omega_{x,y}^1)$ , values for  $\alpha$  and  $\beta$  can be estimated from equations (3-2) and (3-3).

$$\tilde{\alpha} = \sqrt{\frac{Var[J(\Omega_{x,y})]}{Var[I(\Omega_{x,y}^1)]}} \quad - (3-2)$$

$$\tilde{\beta} = E[J(\Omega_{x,y})] - \alpha * E[I(\Omega_{x,y}^1)] \quad - (3-3)$$

From equations (3-2) and (3-3), we can substitute the estimated values of  $\alpha$  and  $\beta$  into equation (3-1) to get the restored value for each pixel such that:

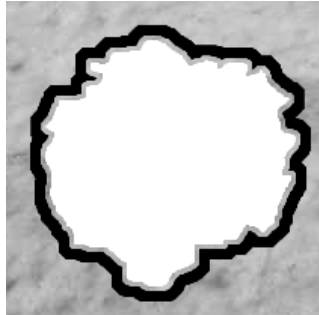
$$\tilde{I}(\Omega_{x,y}) = \frac{J(\Omega_{x,y}) - \tilde{\beta}}{\tilde{\alpha}}, \quad \tilde{I} \text{ is the restored image} \quad - (3-4)$$

### 3.6.2 Restoration Process

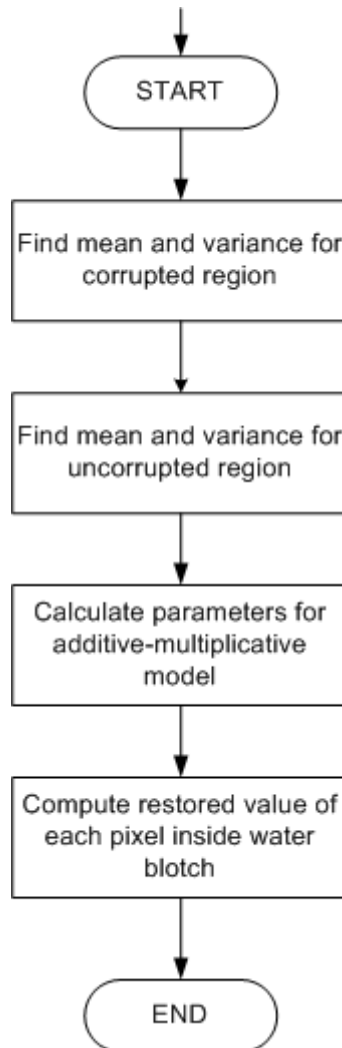
#### (i) Step 1

The first step in the semi-transparent blotch restoration method makes use of the additive-multiplicative model to restore pixels inside the blotch. The flowchart for step 1 is given in Figure 3.4. Suitable values for the parameters  $\alpha$  and  $\beta$  need to be computed from the mean and variance values obtained from the corrupted region and from the uncorrupted region. The uncorrupted region is taken as a region of width  $W$  surrounding the blotch. Since pixels that are next to the border may also be

corrupted, the region is shifted a distance  $S$  (the shift distance) from the border of the blotch, as shown in Figure 3.3.



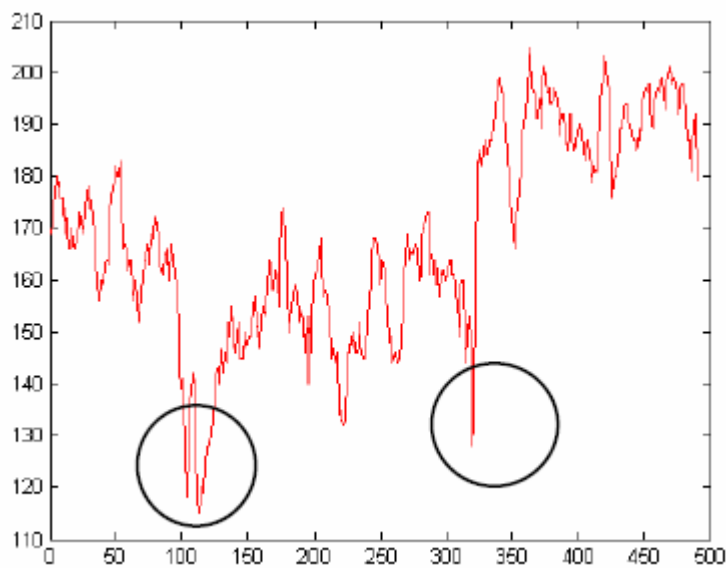
**Figure 3.3:** Corrupted region and region used for estimation of uncorrupted image. The white region indicates all the corrupted pixels inside the semi-transparent blotch, while the black region indicates the uncorrupted region



**Figure 3.4:** Semi-transparent blotch restoration (Step 1)

**(ii) Step 2a**

Step 1 restores pixels inside the blotch, but does not restore well the darker contour that makes the blotch. If we take any line that cuts across the blotch as shown in Figure 3.5, we can notice that there are two sharp negative peaks where the blotch starts and ends. To eliminate these peaks, a linear interpolation is performed across the contour in a direction perpendicular to the gradient of the contour. All pixels within a distance of  $L$  from the border need to be processed in the interpolation operation.



**Figure 3.5:** *The original graylevel in one row of an image with a water blotch [STR03]*

**(iii) Step 2b**

When the normal direction is computed in the previous step, it may happen that for adjacent pixels in the contour, their corresponding normal directions are very different. In this case, not all pixels that are within a distance  $L$  from the border are processed. To overcome this problem, all pixels within a distance  $L$  from the border *and* which have not been processed in the previous step are assigned the average value of its neighbors. Figure 3.6 shows the flowchart for Step 2b.

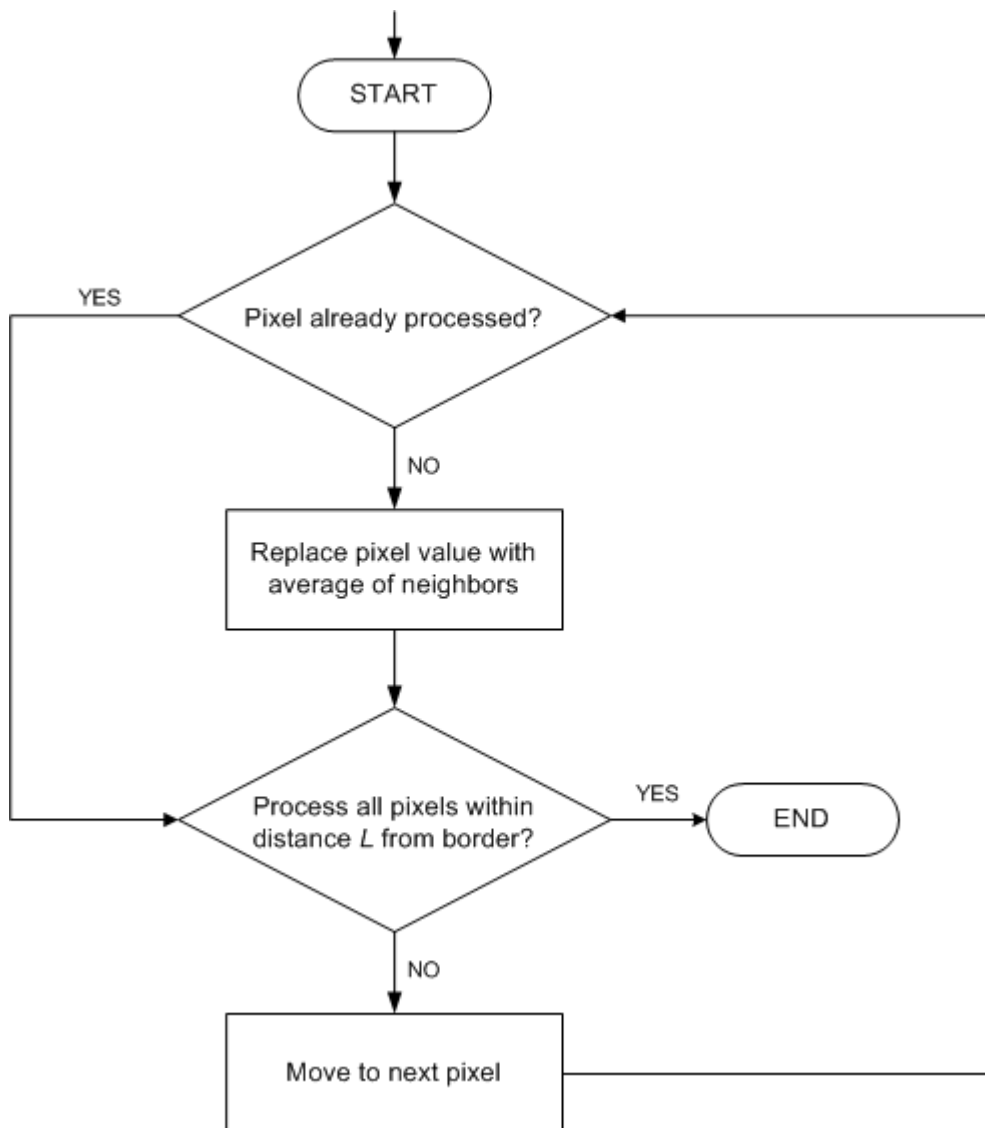


Figure 3.6: Semi-Transparent Blotch removal algorithm – Step 2b

### 3.7 Straight Line Restoration

The Line Scratch Correction Algorithm (LSCA) [TENZ03] has been originally developed to remove vertical straight lines from movie frames. However, in the case of damaged digital photographs, lines like scratches or cracks may run in any direction, not only in the vertical direction. Therefore, LSCA should be extended to work on lines that may have any orientation.

### 3.7.1 LSCA

LSCA uses the same additive-multiplicative model described in section 3.6.1 for the restoration process. Since the corrupted region spans along the vertical direction of the whole frame, then equation (3-1) from the additive-multiplicative model can be simplified to

$$J(\Omega_x, y) = \alpha(\Omega_x, y) * I(\Omega_x, y) + \beta(\Omega_x, y)$$

where  $\Omega_x$  indicates the set of horizontal coordinates of the corrupted region and  $X_{start} < x < X_{stop}$ , where  $X_{start}$  and  $X_{stop}$  are the outer boundaries of the corrupted region.

We can consider a vertical line of thickness  $T$  as a series of  $T$  adjacent columns, each of one pixel thick. LSCA works separately on each column of pixels making the line i.e. it computes the mean and variance for both the corrupted pixels in one column and an estimation of the uncorrupted pixels for the same column. Thus, the mean  $E[J(\Omega_{x,y})]$  can be reduced to  $E[J|\Omega_y]$  to indicate the mean of all pixels located in a set of vertically adjacent coordinates  $\Omega_y$ , at any abscissa<sup>1</sup>  $x \in \Omega_x$ . Similarly, the variance  $Var[J(\Omega_{x,y})]$  is reduced to  $Var[J|\Omega_y]$ . Therefore, we can write

$$Var[J|\Omega_y] = \alpha^2 Var[I|\Omega_y]$$

$$E[J|\Omega_y] = \alpha E[I|\Omega_y] + \beta$$

The uncorrupted region is estimated by a horizontal interpolation. Let  $s(x)$  denote the normalized distance of a pixel within a particular column from  $X_{start}$ , then

$$s(x) = \frac{x - X_{stop}}{X_{start} - X_{stop}}$$

Interpolation is done by substituting  $s(x)$  into the following equation:

---

<sup>1</sup> An abscissa is the  $x$ -coordinate of a point [HUTC93]

$$\hat{I}(x, y) = s(x) * J(X_{start}, y) + (1 - s(x)) * J(X_{stop}, y) \quad \forall x \in \Omega_x$$

where  $\hat{I}(x, y)$  is the estimated value for the uncorrupted pixel value. Substituting the value of  $\hat{I}(x, y)$  in equations (3-2) and (3-3) will yield estimated values for  $\alpha$  and  $\beta$ , and subsequently these values can be substituted in equation (3-4) to get the final restored pixel value.

LSCA has the advantage that it can make use to some extent preserved information under the defect, for example, where a scratch only removes the top layer of a photographic print. The flowchart in Figure 3.7 on the next page outlines the basic steps involved in LSCA.

### 3.7.2 Extending LSCA

The algorithm to restore straight lines running in any direction will rely on the same principle as that behind LSCA, but in this case, another approach needs to be used when estimating the mean and variance values for the uncorrupted region. This new approach is based on the fact that LSCA restores pixels in columns which are parallel to the boundaries. Hence, the additive-multiplicative model needs to be applied on the lines parallel to the boundaries which may be vertical or in any other orientation.

### 3.7.3 Defining The Corrupted Line

For LSCA, only the  $x$ -coordinates of the boundaries of the corrupted (line) region are required as input to the restoration process, since the boundaries are straight vertical lines too and the corrupted region runs across the whole frame. Hence, from these two values, the region to be restored can exactly be defined. However, if the algorithm is extended to restore lines running in any direction, the coordinates of the endpoints of the lines making up the boundaries will be also required. And just like in LSCA, we will assume that the line has a constant width along its length, therefore, the user will need to indicate only one boundary (the two endpoints of the line should be sufficient) and also provide the line width. It should be highlighted that the boundary is the *outer* boundary of the noisy line i.e. it is not part of the corrupted region.

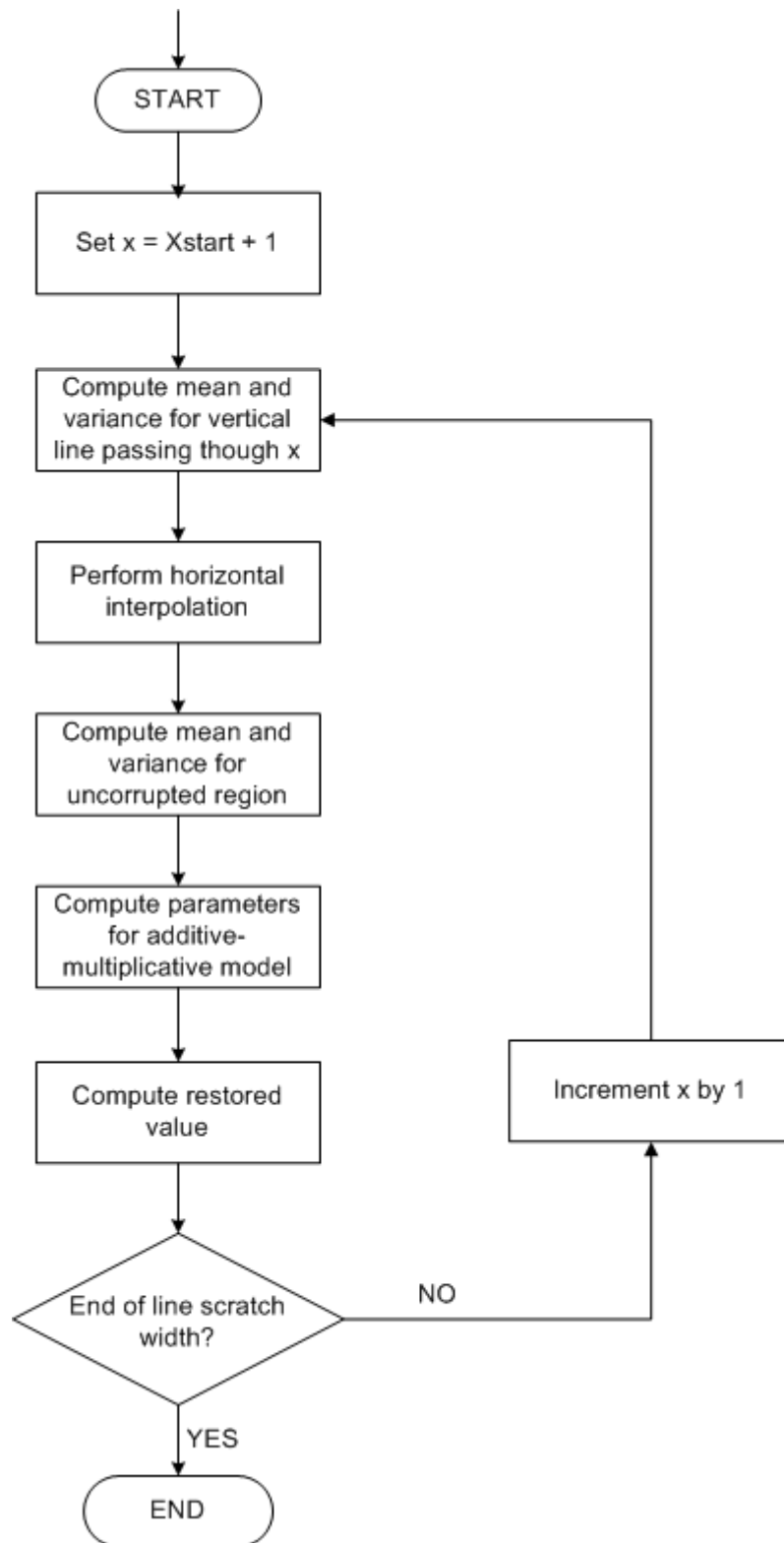


Figure 3.7: Flowchart for LSCA

### 3.7.4 The Digital Difference Analyzer Algorithm

Since the exact location of each pixel making up the boundary of the corrupted region needs to be known, a line drawing algorithm is needed to compute every pixel position on a line given the two endpoints of the line.

The digital difference analyzer (**DDA**) algorithm is an algorithm used to calculate pixels positions on a line [HEAR94]. It is faster and more efficient than using the simple well-known equation  $y = mx + c$ . Its main advantage comes from the fact that the algorithm uses incremental calculations. Incremental or iterative function calculations use previous function values to compute future values. Often expressions within loops depend on a value that is being incremented or decremented on each loop iteration. In these cases the actual function values might be more efficiently calculated by first computing an initial value of the function during the loop set up, and updating the subsequent function values using a discrete version of a differential equation called a difference equation.

#### Derivation of DDA

Consider the equation of a line, where  $m$  is the line gradient and  $c$  is the  $y$ -intercept:

$$y = mx + c$$

Therefore, we can define

$$m = \frac{y_1 - y_2}{x_1 - x_2} = \frac{\Delta y}{\Delta x}$$

where  $(x_1, y_1)$  and  $(x_2, y_2)$  are two points on the line. Consequently, we can also define:

$$\Delta y = m \Delta x$$

and 
$$\Delta x = \frac{\Delta y}{m}.$$

We need to sample the line at discrete positions and determine the nearest pixel to the line at each sampling position. If the line gradient is less or equal to 1, then we

need to sample along the  $x$ -axis to compute each successive  $y$  value, otherwise we sample along the  $y$ -axis to compute each successive  $x$  value.

(i)  $m \leq 1$

$$\Delta y = m (\Delta x)$$

$$y + \Delta y = y + m (\Delta x)$$

$$\Rightarrow y_{k+1} = y_k + m (\Delta x)$$

Since we are sampling along the  $x$ -axis,  $\Delta x = 1$ . Therefore

$$y_{k+1} = y_k + m \quad - (3-5)$$

(ii)  $m > 1$

$$\Delta x = \frac{\Delta y}{m}$$

$$x + \Delta x = x + \frac{\Delta y}{m}$$

$$\Rightarrow x_{k+1} = x_k + \frac{\Delta y}{m}$$

Since we are sampling along the  $x$ -axis,  $\Delta y = 1$ . Therefore

$$x_{k+1} = x_k + \frac{1}{m} \quad - (3-6)$$

Equations (3-5) and (3-6) are based on the assumption that the line is processed from the left hand point to the right hand point. If the processing is reversed, we get:

(iii)  $|m| \leq 1$

$$\Delta x = -1$$

$$y_{k+1} = y_k - m \quad - (3-7)$$

(iv)  $|m| > 1$

$$\Delta y = -1$$

$$x_{k+1} = x_k - \frac{1}{m} \quad - (3-8)$$

From equations (3-5), (3-6), (3-7) and (3-8), the points on one boundary line of a corrupted region can be successfully computed, given the two endpoints of the line. This boundary line is needed to get the pixels locations on the lines parallel to the boundary line so that parameters for the additive-multiplicative model can be computed. Also, the second outer boundary can be automatically calculated from the first one, since the user has already provided the line width. Calculation of the second boundary is dependent on the gradient of the first boundary.

### Applying the additive-multiplicative model

For each parallel line, the mean and variance values are calculated for all pixels located in the line. The values correspond to  $E[J(\Omega_{x,y})]$  and  $Var[J(\Omega_{x,y})]$  i.e. values for the corrupted region. To estimate the uncorrupted region, an interpolation needs to be performed. However, unlike LSCA which performs only horizontal interpolation, here, the direction of interpolation will depend on the gradient of the line. The normalized distance,  $s(x)$ , of a pixel  $p$  on the line is calculated as follows:

$$s(x) = \frac{k}{line\ width + 1} \quad \text{where } 1 < k < \text{line width}$$

Using the normalized distance obtained, the interpolation at a point on the line can now be performed using values from the two outer boundary lines. The restored value of each pixel on the line can hence be computed using equation (3-4).

## 3.8 Image Noise Removal Method

The image noise removal technique can be used to remove blotches, spots and other small non-transparent objects from an image. This method combines frequency and spatial domain information in order to fill a given region with a selected texture. Processing is done in the frequency domain to capture global features and large textures like overall image intensity or slow variation in intensity while processing in the spatial domain is performed to preserve continuity and sharpness for rapid variations in intensity like lines or edges.

The user needs to manually identify the noisy region (the noise mask). In this case, the mask must contain all the noisy pixels, but can be larger than the actual noise. The user also needs to provide the following information for the restoration procedure:

- (i) The neighborhood of the noisy pixels (the **repair subimage**) to provide spatial information. This subimage covers all or part of the noisy pixels.
- (ii) The corresponding prototype subimage (the **sample subimage**) - another subimage that is similar to the repair subimage, but which does not contain any noise, to provide frequency information.

The user can specify any number of pairs of repair and sample subimages and all repair subimages need not be of the same size, though each sample subimage must be of the same size as its corresponding repair subimage.

The image noise removal algorithm can be implemented via three methods: the basic method, the soft scratch method and the split frequency method. The last two methods are based on the basic method. The soft scratch method is an improvement to overcome certain defects arising from the basic method. The split frequency method is a further improved method that combines the basic method and the soft scratch method. All three methods use the same basic concepts namely frequency domain processing, projection onto convex sets and spatial domain processing. These concepts are explained in more details in the next sections.

### 3.8.1 Frequency domain processing

Processing in the **spatial domain** means that operations are applied directly on the pixels in an image while **frequency domain** processing involves modifying the Fourier transform of the image.

#### (i) The Fourier Transform

In general, a transform maps image data into a different mathematical space via a transformation equation. The Fourier transform is based upon the Fourier series which states that any function that periodically repeats itself can be expressed as the sum of sines/cosines of different frequencies, each multiplied by a different coefficient. With the Fourier transform, even functions that are not periodic, but

which have a finite area under the curve, can be expressed as the integral of sines/cosines multiplied by a weighing function. An interesting property of the Fourier transform is that it can be reconstructed back via an inverse process with no loss of information.

**(ii) The One-Dimensional Fourier Transform**

The Fourier transform, denoted by  $\mathfrak{F}\{f(x)\}$ , of a continuous function,  $f(x)$ , of a single variable, is defined by the equation

$$\mathfrak{F}\{f(x)\} = F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux} dx$$

Its inverse transform is given by

$$\mathfrak{F}^{-1}\{F(u)\} = f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux} du$$

where in both cases,  $j = \sqrt{-1}$  and  $e$  is the natural exponent.

**(iii) The Discrete Fourier Transform (DFT)**

To be able to apply the Fourier transform on images, which consist of discrete pixels, the discrete Fourier transform is used instead. The Fourier transform of a discrete function of a single variable,  $f(x)$ ,  $x = 0, 1, 2, \dots, M-1$ , is defined as:

$$F(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(x) e^{-j2\pi ux/M} \quad - (3-9)$$

for  $u = 0, 1, 2, \dots, M-1$

We can also replace  $e^{j\theta}$  by  $\cos \theta + j \sin \theta$  in equation (3-9) using Euler's formula to get an equivalent expression:

$$F(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(x) [\cos 2\pi ux / M - j \sin 2\pi ux / M]$$

The inverse DFT is defined as:

$$f(x) = \sum_{u=0}^{M-1} F(u) e^{j2\pi u x / M} \quad \text{for } u = 0, 1, 2, \dots, M-1$$

Since the components of the Fourier transform involve complex quantities, we can convert  $F(u)$  into polar form, thus leading to this expression:

$$F(u) = |F(u)| e^{j\phi(u)}$$

where  $|F(u)|$  is called the **magnitude** or **Fourier spectrum** of the transform

$$|F(u)| = [R^2(u) + I^2(u)]^{1/2}$$

and  $\phi(u)$  is the **phase angle** or **phase spectrum** of the transform

$$\phi(u) = \tan^{-1} \left[ \frac{I(u)}{R(u)} \right]$$

$R(u)$  is the real part and  $I(u)$  is the imaginary part of  $F(u)$ .

#### (iv) The Two-Dimensional Discrete Fourier Transform

To get the DFT for an image, we need to extend the one-dimensional DFT to two-dimensions. Hence, for an image  $f(x,y)$  of size  $M \times N$ , its DFT is

$$F(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M + vy/N)} \quad - (3-10)$$

for  $u = 0, 1, 2, \dots, M-1$  and  $v = 0, 1, 2, \dots, N-1$  while the inverse of this transform is

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M + vy/N)}$$

for  $x = 0, 1, 2, \dots, M-1$  and  $y = 0, 1, 2, \dots, N-1$

As the above equations show, implementing the two-dimensional DFT involves extensive computation, but we can use an interesting property of the two-

dimensional DFT to reduce computations such that a two-dimensional DFT is computed as two separate one-dimensional DFT. This property is known as the **separability** property.

The two-dimensional transformation can be realized by a succession of one-dimensional transformations along each of the spatial coordinates. Thus, equation (3-10) can be expressed as

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} e^{-j2\pi u x / N} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi v y / N} \quad - (3-11)$$

for  $M=N$  and  $u, v = 0, 1, 2, \dots, N-1$

Equation (3-11) can be expressed in the form

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} F(x, v) e^{-j2\pi u x / N}$$

where

$$F(x, v) = N \left[ \frac{1}{N} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi v y / N} \right]$$

Thus, in the case of a digital image which consists of rows and columns, the two-dimensional DFT  $F(u, v)$  can be obtained by

1. taking the one-dimensional DFT of every row of image  $f(x, y)$  to get  $F(x, v)$  and multiplying the result by  $N$ .
2. taking the one-dimensional DFT of every column of  $F(x, v)$

### (v) The Fast Fourier Transform

The DFT contains many additions and multiplications involving complex numbers. But proper decomposition of equation (3-9) can reduce the overhead associated with these calculations. This decomposition procedure is known as the Fast Fourier Transform (FFT) algorithm. While the one-dimensional Fourier transform of  $M$

points requires about  $M^2$  complex multiplication/addition operations, the FFT requires only about  $M \log_2 M$  operations (assume  $M=2^n$ ).

If  $M = 2^n$  and  $W_{2K} = e^{-j2^\pi/M}$ , then equation (3-9) can be expressed as

$$F(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(x) W_M^{ux}$$

Since  $M = 2^n$ , there exists  $K$  such that  $M = 2K$

$$F(u) = \frac{1}{2K} \sum_{x=0}^{2K-1} f(x) W_{2K}^{ux} = \frac{1}{2} \left[ \frac{1}{K} \sum_{x=0}^{K-1} f(2x) W_{2K}^{u(2x)} + \frac{1}{K} \sum_{x=0}^{K-1} f(2x+1) W_{2K}^{u(2x+1)} \right]$$

Now, since the square of a  $2K^{\text{th}}$  root of unity is an  $K^{\text{th}}$  root of unity, we have that

$$W_{2K}^{u(2x)} = W_K^{ux}$$

Hence we can rewrite  $F(u)$ :

$$F(u) = \frac{1}{2} \left[ \frac{1}{K} \sum_{x=0}^{K-1} f(2x) W_K^{ux} + \frac{1}{K} \sum_{x=0}^{K-1} f(2x+1) W_K^{ux} W_{2K}^u \right] \quad - (3-12)$$

Defining  $F_{\text{even}}(u) = \frac{1}{K} \sum_{x=0}^{K-1} f(2x) W_K^{ux}$  and  $F_{\text{odd}}(u) = \frac{1}{K} \sum_{x=0}^{K-1} f(2x+1) W_K^{ux}$

for  $u = 0, 1, 2, \dots, K-1$  reduces equation (3-12) to

$$F(u) = \frac{1}{2} \left[ F_{\text{even}}(u) + F_{\text{odd}}(u) W_{2K}^u \right] \quad u = 0, 1, \dots, K-1 \quad - (3-13)$$

and

$$F(u+K) = \frac{1}{2} \left[ F_{\text{even}}(u) - F_{\text{odd}}(u) W_{2K}^u \right] \quad u = K, K+1, \dots, 2K-1 \quad - (3-14)$$

Equation (3-14) is due to the fact that  $W_K^{K+u} = W_K^u$  and  $W_{2K}^{K+u} = -W_{2K}^u$ .

Thus, a discrete Fourier transform of length  $N$  can be rewritten as the sum of two discrete Fourier transforms, each of length  $N/2$ . One of the two transforms is formed from the even-numbered points of the original  $N$ , the other from the odd-numbered points. Similarly, each of the  $N/2$ -point transforms can be computed using two  $N/4$ -point transforms i.e. each one of these parts can be broken into its even-numbered and odd-numbered parts, and the process can be continued until the summation has become divided into 1-point Fourier transforms (which are identity transforms).

We can compute an  $N$ -point DFT by dividing it into two parts:

- (i) The first half of  $F(u)$  for  $u = 0, 1, 2, \dots, K-1$  can be found from equation (3-13)
- (ii) The second half for  $u = K, K+1, \dots, 2K-1$  can be found simply by reusing the same terms differently as shown by equation (3-14).

#### (vi) Properties of the Fourier domain

The Fourier transform of any image gives the frequency aspects of that image which give vital clues leading to the details of the image. The value of the transform at the origin i.e. at  $F(0,0)$ , is the average of  $f(x,y)$ . Thus, if we have an image  $f(x,y)$ , then this value represents the average gray level of the image. As we move away from the origin, the low frequencies correspond to the slowly varying components of an image i.e. smooth transitions in gray levels, while further away, the higher frequencies correspond to faster changing gray levels in the image like edges and noise.

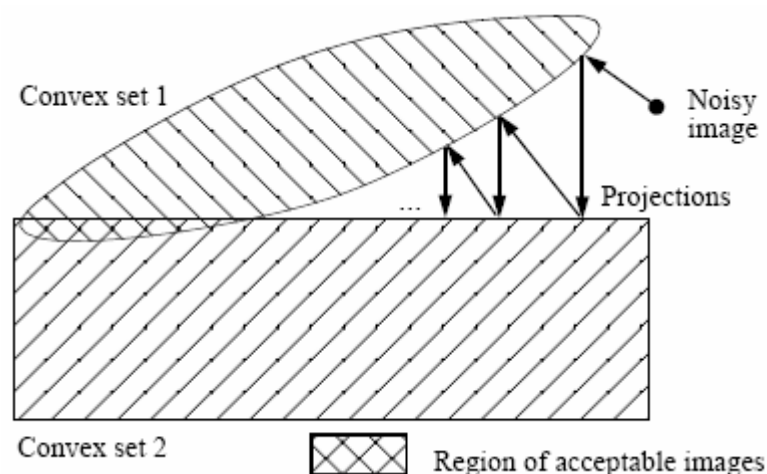
### 3.8.2 Projections Onto Convex Sets

The method of projections onto convex sets (POCS, [YOUL82]) can be used to obtain information from one domain using information available in other domains. This method allows the use of any information about the image (or any other signal) as long as the information can be represented as a closed convex set. For digital image restoration, it is convenient to restrict the method to finite dimensional spaces. This space might be, for example, the space of all  $M \times N$  complex matrices where the image has  $M$  rows and  $N$  columns.

Given a set  $C$  in such a space,  $x, y \in C$ , we say the  $C$  is convex if and only if for any  $0 \leq \mu \leq 1$ ,

$$\mu x + (1 - \mu) y \in C$$

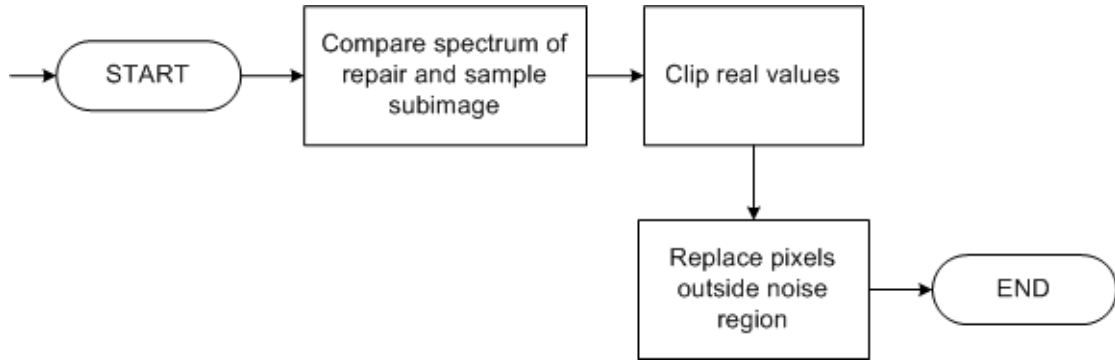
$C$  is closed if it contains all its limit points. Projection onto a convex set consists of finding an image satisfying the constraint (the closed convex set) and 'closest' to the image being projected. A projection can be thought of as making the least possible change to make its input satisfy the constraints. POCS allows an arbitrary amount of a priori information to be incorporated as constraints. Repeated projection onto all the convex sets is guaranteed to find an image that satisfies all the constraints if at least one such image exists. The convexity of the sets is required to prevent the occurrence of divergence. Figure 3.8 shows a pictorial representation of POCS.



**Figure3.8:** Pictorial representation of POCS

### 3.8.3 Basic Method

After the user has marked the noisy region, and chosen at least a pair of repair and sample subimage, the automatic restoration procedure can be initiated. Figure 3.9 shows the basic steps involved in this algorithm.



**Figure 3.9:** One iteration in the basic algorithm

### Frequency Domain Processing

The next step consists of modifying the Fourier spectrum of the repair subimage, using the spectrum of the sample subimage as reference.

Let  $R$  be the Fourier transform of the repair subimage and  $S$  be the Fourier transform of the sample subimage.

The magnitude of  $R$  is compared against that of  $S$ : the magnitude of the repair subimage is then the minimum magnitude between that of  $R$  and  $S$  for each frequency  $(u, v)$  in the spectrum, except for the point  $u=0$  and  $v=0$  (the DC), where the magnitude of  $R$  is not changed.

$$R_{\text{new}}(u, v) = \begin{cases} R_{\text{old}}(u, v), & \text{if } |R_{\text{old}}(u, v)| < |S(u, v)| \\ S(u, v), & \text{if } |R_{\text{old}}(u, v)| > |S(u, v)| \\ R_{\text{old}}(u, v) & \text{If } (u, v) = (0, 0) \end{cases}$$

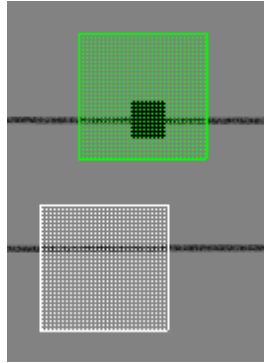
$R_{\text{new}}$  : New repair subimage

$R_{\text{old}}$  : Old repair subimage

$S$  : Sample subimage

The above operation in the frequency domain is performed since noise will add to the spectrum of the repair subimage, and therefore, by taking the minimum, we are reshaping the spectrum of the repair subimage using the sample spectrum as reference. The DC value is kept unchanged since this represents the overall subimage intensity. Furthermore, the spectrum of an image does not change when part of the

image is translated with wrap-around [HITO96b]. This means that features (like lines) in the sample subimage need not be in the same location as in the corresponding repair subimage (Figure 3.10) since these translated features are automatically realigned in the repair subimage.



**Figure 3.10:** *Choosing the sample subimage from another region.*

*The green mask represents the repair subimage, the white mask represents the sample subimage, and the black mask represents the noisy pixels*

The inverse Fourier transform is then taken using the new magnitude and the unchanged phase of  $R$ . The operation performed also satisfies the POCS principle since the underlying set  $C$

$$C = \{r: |R(u,v)| \leq |S(u,v)|, (u,v) \neq 0\}$$

is closed and convex and we can consider the operation performed as being a projection [HITO96a].

### Spatial domain processing

The intensity range of the new repair subimage obtained is clipped such that its real intensity values lie in the range of 0 to the maximum possible graylevel. The pixels surrounding the noisy pixels i.e. pixels found within the repair subimage, but outside the noise mask, are replaced with the pixels of the original repair subimage. This is necessary since non-noisy pixels found within the repair subimage might also have been transformed during the frequency domain operation.

Thus, for each pixel,  $r(x,y)$ , in the repair subimage  $r$ , the following equation is applied:

$$r(x,y) = r(x,y) * (1 - w) + r_0(x,y) * w \quad - (3-15)$$

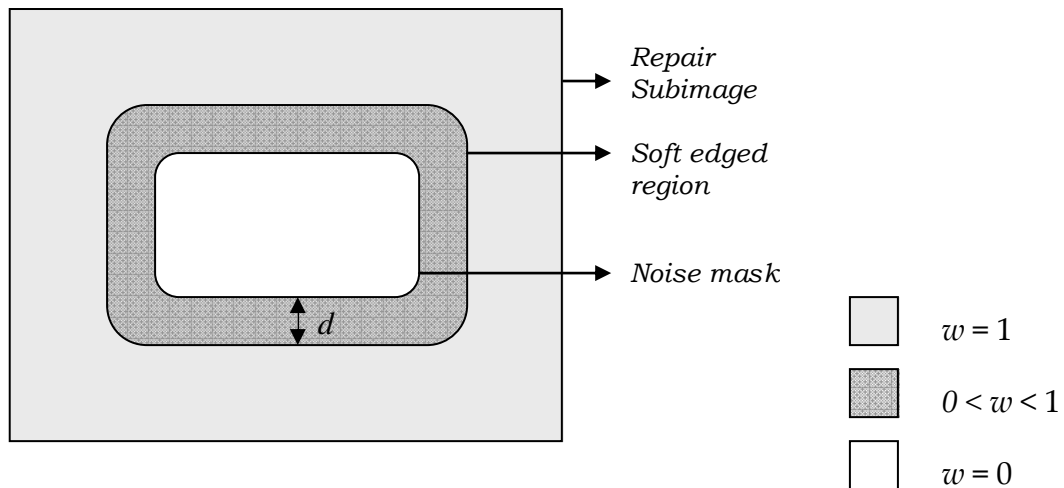
where  $w = 0$  if  $r(x,y)$  is also a noisy pixel, otherwise  $w = 1$ .  $r_0$  is the original repair subimage.

### Iterations

The above operations in the frequency and spatial domains are repeated using the new repair image obtained from the previous iteration as input for the next iteration. This is again done for each repair subimage that the user has specified. If the image being processed is a color image, then the same operations are applied to each of the three color bands (red, green and blue).

### 3.8.4 Soft Scratch Method

The soft scratch method builds upon the basic algorithm to enhance the final output. The aim of this modification is to reduce the sharp transition that might be visible on the final image around the noise mask when the basic algorithm is used. For this purpose, when working in the spatial domain, a soft edged noise mask (Figure 3.11) is used.



**Figure 3.11:** Soft edged noise mask

Thus, equation (3-15) needs to be slightly modified to take into account a soft edged region such that  $w$  increases towards 1 as the distance from the edge of the noise mask increases. The equation remains the same except that the value of  $w$  depends on the position of the pixel under consideration.  $w$  takes values as follows:

- (i) 0 if  $r(x,y)$  is a noisy pixel
- (ii)  $0 < w < 1$  if  $r(x,y)$  is inside the soft edged region
- (iii) Otherwise  $w = 1$ .

### 3.8.5 Frequency Split Method

The frequency split method is based on both the basic and the soft scratch methods. The goal of this method is to allow the user to choose a sample subimage that has a different shading as compared to the repair subimage to provide more flexibility in choosing a reference subimage. To ignore any difference in shading between the repair and the sample subimage, the low frequency components of the repair subimage are not processed when operations in the frequency domain are being performed, since these frequencies are associated with the global shading. To achieve this, the first step when working in the frequency domain is to perform a high pass filtering on the repair subimage to split the image into a high pass filtered image and a low pass filtered image.

The spectrum of the high pass filtered image is then compared with that of the sample subimage, using the same rules as in the basic algorithm. However, in this case, since the low frequencies are being ignored, the third rule can be disregarded. The spectrum of the high pass filtered repair subimage is then defined as

$$R_{\text{new}}(u, v) = \begin{cases} R_{\text{old}}(u, v) & \text{if } |R_{\text{old}}(u, v)| < |S(u, v)| \\ S(u, v) & \text{if } |R_{\text{old}}(u, v)| > |S(u, v)| \end{cases}$$

- $R_{\text{new}}$  : New repair subimage
- $R_{\text{old}}$  : Old repair subimage
- $S$  : Sample subimage

After comparing the magnitudes, an inverse FFT is applied on the high pass filtered component of the repair subimage. A replace operation is then performed such that pixels within the repair subimage but outside the noise mask are replaced with intensity values of the high pass filtered subimage obtained from the previous iteration. The processed high pass filtered image can now be merged with the

unprocessed low pass filtered part. The subsequent operations in the spatial domain are the same as those in the soft scratch method. Figure 3.12 shows the data flow within the split frequency method.

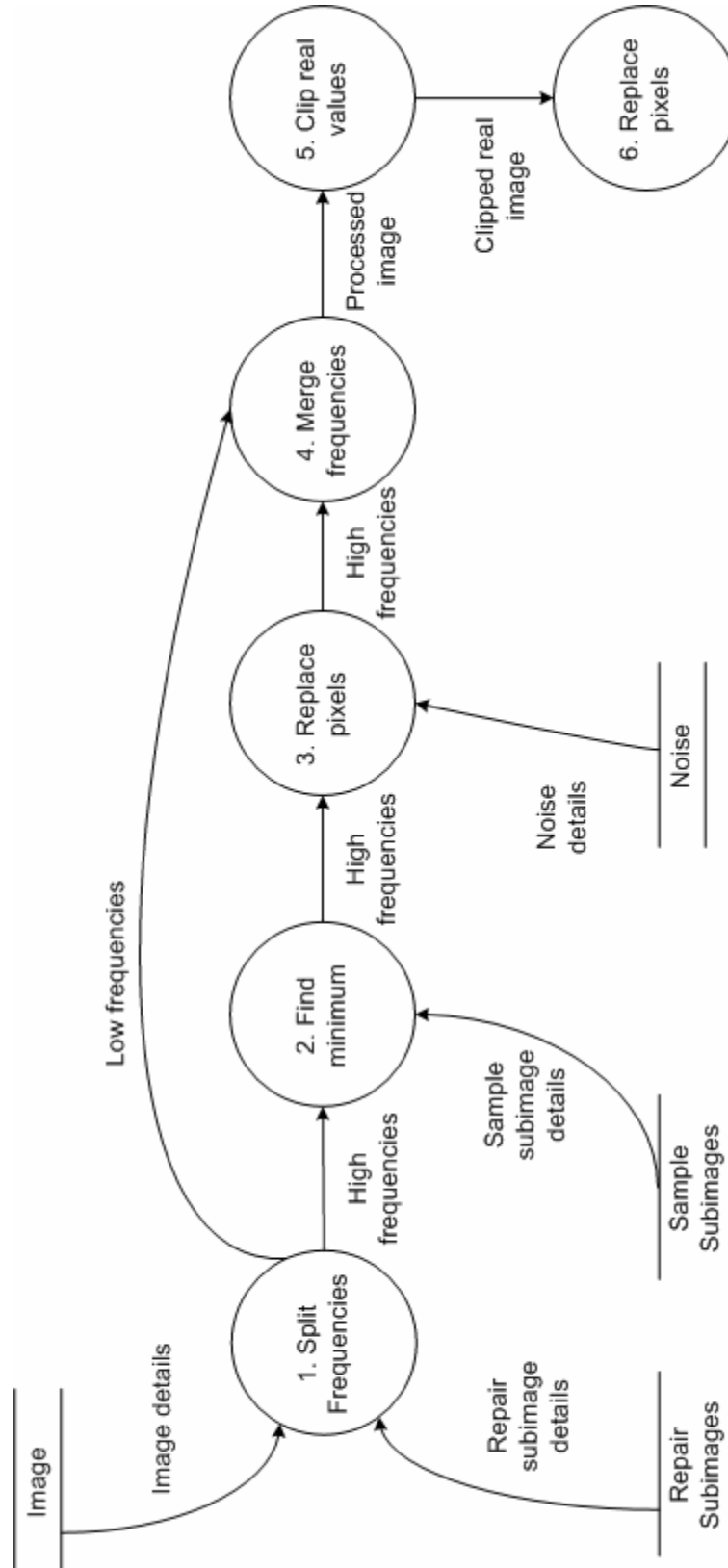


Figure 3.12: Data Flow Diagram for frequency split method

### 3.9 The User Interface

The critical inputs to the methods will be provided by the user via the system interface: the user needs to manually identify the defects on the screen itself and may also need to define repair and sample subimages if the image noise removal algorithm is being used. Hence, the interface that will be provided to the user is a very important issue while designing the system since the result of the restoration process depends largely on the inputs provided. If the user cannot properly define the region he wants to restore, the results will not be of acceptable quality. Also, the aim is to simplify the task of restoration for the user, therefore, he should not be spending too much time to define noise or subimage masks as this would be equivalent to using commercial software where restoration is a tedious and time-consuming task.

To provide a simple yet effective interface, there are certain principles of human-computer interaction (HCI) that need to be followed. An interface should be simple (not simplistic), easy to learn, and easy to use. The interface should be designed in such a way that a novice user can easily use the software. One way to support simplicity is to reduce the presentation of information to the minimum required to communicate adequately. For example, wordy descriptions for command names or messages should be avoided. Another way to design a simple but useful interface is to use natural mappings and semantics. The arrangement and presentation of elements affects their meaning and association and the use of graphical controls helps to support user recognition rather than user recall. This is particularly useful since users remember a meaning associated with a familiar object more easily than they remember the name of a particular command. A natural mapping also makes the relationship between the actual action of the device and the action of the user obvious. Shortcuts should be provided so that frequent functions can be quickly available.

Users, even expert ones, usually make errors; therefore, we should anticipate possible sources of errors and prevent the possible occurrence of such errors e.g. by using forcing functions like disabling menus and buttons not required for a particular method. Feedback to the user should also be provided either to inform the

user that he has performed an inappropriate or incomplete task or to indicate the state of the process e.g. by using progress bars.

A multiple document interface (MDI) will be used for opening and manipulating images. The advantage of using an MDI is that the user is able to open the same or another image in a new window; this is useful when the user wants to compare an original image and the results after working on the image. A menu will be provided from which the appropriate restoration method can be chosen. In addition to the noise identification tools mentioned in section 3.5, a tool to select repair and sample subimages need to be provided such that the user can choose subimages of appropriate sizes.

Details of the interface can be found in Appendix A.

### **3.10 Summary**

In this chapter, three techniques have been identified for the removal of defects from images namely the semi-transparent blotch removal algorithm, an extended version of the line scratch correction algorithm and the image noise removal algorithm. The techniques that will be used for defect detection are also identified and a coarse-grained design of the proposed system is provided.