2016 IEEE 6th International Conference on Advanced Computing

# Vectorization and optimization of fog removal algorithm

Krishna Swaroop Gautam
Uurmi Solutions Pvt. Ltd.,
Hyderabad, India
Email: krishnasg@uurmi.com

Abhishek Kumar Tripathi
Uurmi Systems Pvt. Ltd.,
Hyderabad, India
Email: abhishekt@uurmi.com

M.V. Srinivasa Rao
Uurmi Systems Pvt. Ltd.,
Hyderabad, India
Email: mvs@uurmi.com

***Abstract*—Some of the image processing algorithms are very costly in terms of operations and time. To use these algorithms in real-time environment, optimization and vectorization are necessary. In this paper, approaches are proposed to optimize, vectorize and how to fit the algorithm in low memory space. Here, optimized anisotropic diffusion based fog removal algorithm is proposed. Fog removal algorithm removes the fog from image and produces an image having better visibility. This algorithm has many phases like anisotropic diffusion, histogram stretching and smoothing. Anisotropic diffusion is an iterative process that takes nearly $70\%$ of time complexity of the whole algorithm. Here, optimization and vectorization of the anisotropic diffusion is proposed for better performance. However, optimization techniques cost some accuracy but that can be neglected for significant improvement in performance. For memory constraint environment, a method is proposed to process the entire block of image and maintains the integrity of operations. Results confirm that with our optimization and vectorization approaches, performance is increased up to $90$ fps (approximately) for VGA image on one of the image processing DSP simulator. Even if, system doesn't have vector operations, the proposed optimization techniques can be used to achieve better performance ($2\times$ faster).**

***Index Terms*—Anisotropic diffusion, image smoothing, DSP (Digital Signal Processing), Vectorization, SIMD, image restoration, DMA (Direct Memory Access).**

## I. Introduction

Poor weather conditions such as fog and rain degrades atmospheric visibility. Low visibility degrades perceptual image quality and efficacy of the computer vision algorithms such as surveillance, object segmentation, recognition and tracking. Thus, it is essential to make vision algorithms more robust to change in weather conditions. Low visibility in poor weather is due to the suspension of water particles in atmosphere. Light coming from the atmosphere and light reflected from an object are scattered by these water particles, resulting the low visibility of the scene. Fog removal algorithm is used to remove the fog from the foggy image/scene. However, fog removal algorithm is costly and complex in terms of operations for both low and high resolution images. Any computer vision based application looks more realistic with its high resolution image/video processing capability and consumer wants to use only high quality real-time computer vision based applications. So, it is very important for any vendor to provide the solution for such a heavy computer vision application. They are supposed to resort powerful hardware to support operations used in particular algorithm with adequate performance and also needs to optimize the algorithm according to chosen hardware capabilities, which utilize it at extreme. Unfortunately, fog removal algorithm is one that cannot be used directly in real-time high resolution image processing applications. Here, some sorts of generic optimization and vectorization techniques are proposed for SIMD and non-SIMD based processors to use the fog removal algorithm on real-time high quality application.

Optimization is a process to reduce the number of operations. Vectorization is a process to vectorizing the scalar operations into vector operations. In scalar operations, single instruction of operation is perform on one pair of operand, while in vectorization, single instruction of operation refers to a pair of vector operands. Vectorization is based on SIMD (Single Instruction Multiple Data) architecture, which achieves the parallelism. If, anisotropic diffusion based fog removal algorithm [1] is implemented without any optimization, vectorization and memory constraints then to process a VGA image on Intel $i3$ with processor $3.1$ GHz takes $0.2$ sec i.e. $5$ fps (frame per second). To use the fog removal algorithm in the real-time environment, proposed optimization and vectorization approaches can be used in environments like GPU, DSP processor and FPGA to achieve much higher performance. Fog removal algorithm works on $24$-bits precision, to optimize the anisotropic diffusion operations, a conversion from $24$-bit float to $16$-bit integer needs to be done.

This paper is organized as follows. In section II, related work is discussed. Section III describes the fog removal algorithm and its optimization and vectorization methodologies. Results are discussed in section IV. Section V concludes the paper.

## II. Related Work

In last few years, many vision algorithms [1]-[6] have been proposed for the removal of fog from images and videos. It is noted that amount of fog is directly proportional to the distance between the viewer and the object. At long distance, density of fog is high in comparison with short distance. Hence for the removal of fog, estimation of the scene depth is required. If, input is a single foggy image then estimation of the scene depth is under constrained. There are many algorithms [1]-[5] which remove fog from single image. For the estimation of

978-1-4673-8286-1/16 $31.00 © 2016 IEEE
DOI 10.1109/IACC.2016.73


Fig. 1: Block diagram of fog removal algorithm

scene depth, these vision algorithms rely on some assumptions like pre-estimated scene distance, relation between shading & transmission functions, and relevant scene properties. Here, we proposed the optimization of fog removal algorithm proposed by Tripathi et.al [1] due to its efficiency and performance. This algorithm is data driven and avoids user intervention and can be used for color as well as monochrome image. Even in case of high amount of fog, this algorithm performs well.

There is no previous work on optimization of fog removal algorithm. But as the part of its improvement, parallelization of fog removal algorithm is performed, where, multi-threading concept is implemented by using *pthread* and *OpenCV* libraries in C. *pthread* is multi-threading library available in C. *OpenCV* is a open source computer vision library under open-source BSD license. By using 8-threads on 8 cores, nearly two times better performance is achieved than its single thread implementation. For further improvement, optimization and vectorization of all the operations in the algorithm need to be done.

## III. Fog Removal Algorithm

According to Koschmieder's law, fog phenomenon can be represented as the combination of attenuation and airlight as mentioned in Eq (1)

$$I(x,y) = I_0(x,y)e^{-\beta d(x,y)} + I_\infty(1 - e^{-\beta d(x,y)}) \quad (1)$$

Where, $I_0(x,y)$ is image intensity in clear weather condition, $\beta$ is the extinction coefficient and $d(x,y)$ is the scene point distance from the camera. $I_\infty$ is global atmospheric constant and $I(x,y)$ is observed image intensity.

In Eq (1), right hand side is the combination of two terms i.e. attenuation and airlight. Attenuation is a monotonically decreasing exponential function of distance which reduces contrast of the object and thus its visibility in the scene. Airlight is an increasing function of the scene point distance $d(x,y)$ which adds whiteness in the scene. Block diagram of the fog removal algorithm is shown in Fig. 1. First module, fog removal is used for the removal of fog from the images. As amount of fog depends on the depth of the scene thus estimation of the depth for the image is required. Airlight map is estimated, which stores the depth information. Airlight map is the function of the distance of object from the camera. Value of airlight should be different for those objects which are at different distance. Considering this requirement, airlight map should be smooth except the edges of the objects. Hence, airlight map preferred intra-object smoothing over inter-object smoothing. This requirement is satisfied by anisotropic diffusion. Thus coarse airlight map is estimated by dark channel prior. Anisotropic diffusion is used for the refinement of the airlight map. Once airlight map is estimated, image is restored using inverse Koschmeider's law. De-foggy image may have low contrast thus histogram stretching is used to enhance the contrast. Finally, de-noising is performed to remove the noise present in the image. Box filter of kernel size $3 \times 3$ is used for image de-noising.

### A. Optimization

Optimization is a process of improving the algorithm or reduces the time complexity. In optimization, anisotropic diffusion and histogram stretching are focussed here, as these are the most time consuming process in the algorithm.

*1) Anisotropic Diffusion:* Anisotropic diffusion performs smoothing by preserving edge information. Thus, where edge information is high, less smoothing or diffusion required. According to Perona and Malik [7], there are two diffusion functions for anisotropic diffusion as given in Eq (2) and Eq (3), where $D$ is the edge information and $c$ is the conduction coefficient of the diffusion process. Both are asymptotically decreasing, but operation wise Eq (2) is more costly, because of the use of exponential function.

$$c = e^{-D^2} \quad (2)$$

$$c = \frac{1}{1 + D^2} \quad (3)$$

On real time environment computation of Eq (3) is faster than Eq (2). For further optimization, diffusion functions is replaced with one lookup table (LUT). That mean, no need to calculate the diffusion function every time for each pixel. However, lookup table can be used only in case of conversion of algorithm from 24-bits float precision to 16-bits integer that defines the range/size of lookup table. Lookup table implementation for both diffusion methods (Eq (2) and Eq (3)) gives the same performance, because of its one time calculation of diffusion function.

*2) Histogram Stretching:* Histogram stretching is used to improve the contrast of the image. For that, transfer function of the image is calculated. We replaced the histogram stretching function with one lookup table that reduces the computation time for each pixel. Transfer function of histogram stretching is shown in Fig. 2. Where, $r$ and $s$ axis represent input and output intensity values respectively and parameters $r1$, $s1$, $r2$, and $s2$ determine the shape of the transfer function. Values

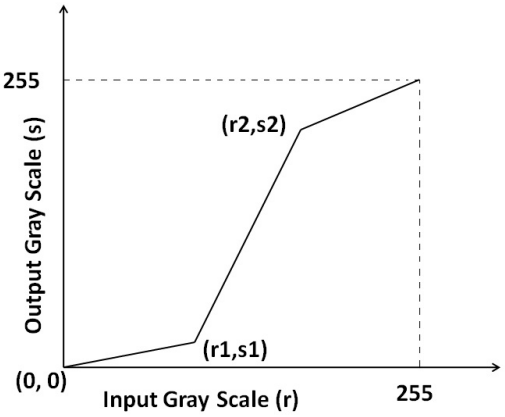


Fig. 2: Transfer function of histogram stretching

of the parameters are chosen based on the histogram of the image. The values of $r1$ and $r2$ are intensity values of $10\%$ and $90\%$ of the cumulative histogram respectively. The values of $s1$ and $s2$ are $5\%$ and $95\%$ of the output intensity range. Estimation of these parameters is data driven which avoids the user intervention.

### B. Vectorization

Vectorization is a process to speed up the algorithm by processing multiple elements in a vector. Vectorization achieves the parallelism. Vectorization is based on SIMD (Single Instruction Multiple Data) architecture. For vectorization, we have to select the appropriate vector data structure to maintain the integrity and accuracy of algorithm. Some DSP processor doesn't provide floating point operations on vector. To port the algorithm on such a platform a conversion is required as per processor configuration. To vectorize any algorithm, first identify the parts that can be vectorized. For example, below mentioned sample pseudo code can be vectorized.

$$for(i = 0; i < n; i++)$$
$$A[i] = B[i] + C[i];$$

Elements of $A$ can be calculated from $B$ and $C$ parallelly, because $A$ is not dependent on the previous or old value of itself. $A[i]$, $A[i+1]$ and so on can be calculated parallelly. Following sample code cannot be vectorized as element at $i$ of $A$ is dependent on element at $(i-1)$ of itself. So, $A[i+1]$, $A[i]$, $A[i-1]$ and so on, cannot be calculated parallelly and cannot be vectorized.

$$for(i = 0; i < n; i++)$$
$$A[i] = B[i] + A[i-1];$$

Vectorization of the fog removal algorithm is a big challenge, because the algorithm has many phases and every phase needs a kind of shift/selection operations to achieve vectorization. In the below sections, all phases are described with their corresponding vectorization method.

*1) Anisotropic Diffusion:* Anisotropic diffusion performs smoothing on intra region and preserves the edges. This function uses the neighbor's information to estimate the amount of smoothing required. Pictorial diagram to get the neighbor's information is given in Fig. 3. Detailed explanations of anisotropic diffusion are given in [7] and [8]. Below is scalar pseudo code to get edge information for anisotropic diffusion:

```
for (i=1; i<height-1; i++)
    for (j=1; j<width-1; j++){
        //calculate nabla-N,S,W,E (nabla is
        edge information in particular direction)
        nablaN = image[i-1][j] - image[i][j];//North
        nablaS = image[i+1][j] - image[i][j];//South
        nablaW = image[i][j-1] - image[i][j];//West
        nablaE = image[i][j+1] - image[i][j];//East
        //Code for diffusion using nablaN, nablaS,
        nablaW and nablaE
        ...
    }
```

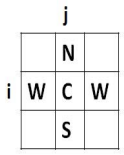


Fig. 3: Neighbor Pixels in Image

For vectorization of above code, vectors for each direction north, south, west and east as vecNablaN, vecNablaS, vecNablaW and vecNablaE respectively are created. Suppose size of each vector is $n$, vectorization of anisotropic diffusion will be as given in below pseudo code:

```
for(i=1; i<height-1; i++)
 for(j=1; j<width-1; j=j+n){
  /* Vector for each direction vector */
  vecCentre[1:n] =  image[i][j:j+n-1];
  vecNorth[1:n]  =  image[i-1][j:j+n-1];
  vecSouth[1:n]  =  image[i+1][j:j+n-1];
  vecEast[1:n]   =  image[i][j+1:j+n];
  vecWest[1:n]   =  image[i][j-1:j+n-2];

  /* Calculate nabla-N,S,W,E using above vectors */
  vecNablaN[1:n] = vecNorth[1:n] - vecCentre[1:n];//North
  vecNablaS[1:n] = vecSouth[1:n] - vecCentre[1:n];//South
  vecNablaW[1:n] = vecWest[1:n] - vecCentre[1:n];//West
  vecNablaE[1:n] = vecEast[1:n] - vecCentre[1:n];//East

  /* Code for edge information for diffusion using
  vecNablaN, vecNablaS, vecNablaW and vecNablaE */
    ...
}
```

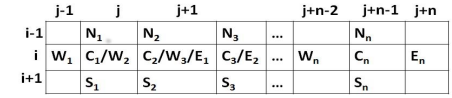


Fig. 4: Pictorial diagram to get vector for all direction

Pictorial diagram shown in Fig. 4 shows the selection of vectors of directions (North, South, East and West) for vector of center pixels. Where, $C_1$, $C_2$, $C_3$, ... $C_n$ refers to vector

of $n$ center elements indexed from $(i, j)$ to $(i, j + n - 1)$. $N_1$, $N_2$, $N_3$, ... $N_n$ refers to vector of $n$ north direction neighbor's elements for corresponding centers $C_1$, $C_2$, $C_3$, ... $C_n$ and indexed from $(i - 1, j)$ to $(i - 1, j + n - 1)$. $S_1$, $S_2$, $S_3$, ... $S_n$ refers to vector of $n$ south direction neighbor's elements for corresponding centers $C_1$, $C_2$, $C_3$, ... $C_n$ and indexed from $(i + 1, j)$ to $(i + 1, j + n - 1)$. $W_1$, $W_2$, $W_3$, ... $W_n$ refers to vector of $n$ west direction neighbor's elements for corresponding centers $C_1$, $C_2$, $C_3$,... $C_n$ and indexed from $(i, j - 1)$ to $(i, j + n - 2)$. $E_1$, $E_2$, $E_3$,... $E_n$ refers to vector of $n$ east direction neighbor's elements for corresponding centers $C_1$, $C_2$, $C_3$, ... $C_n$ and indexed from $(i, j + 1)$ to $(i, j + n)$.

*2) Restoration:* Output $A$ (airlight map) of anisotropic diffusion is used to restore the image $I_0(x, y)$. For restoration, inverse Koschmeider's law is used and given in Eq (4).

$$I_0(x, y, c) = \frac{I(x, y, c) - A(x, y)}{1 - (A((s, y))/I_\infty(c))} \tag{4}$$

Where, $c$ belongs to channel red, green and blue i.e. $c \ \varepsilon \ \{r, g, b\}$. Here, vectorization is possible. First, store the data into intermediate vectors and then process the vectors.

Sample scalar pseudo code for restoration of red channel as follows

```
for(i=1; i<height; i++)
    for(j=1; j<width; j++){
        factor = 1.0/(1-A[i][j]);
        Red = image[i][j] - A[i][j];
        out[i][j] = Red*factor;
    }
```

Vectorization for the above pseudo code is:

```
for(i=1; i<height; i++)
    for(j=1; j<width; j=j+n){
        vecI[1:n] = image[i][j:j+n-1];
        vecA[1:n] = A[i][j:j+n-1];
        vecF[1:n] = 1/vecI[1:n];
        vecR[1:n] = vecI[1:n] - vecA[1:n];
        out[i][j:j+n] = vecR[1:n] * vecF[1:n];
    }
```

where, *vecI, vecA, vecF* and *vecR* are vectors of size $n$.

*3) Histogram Stretching:* Histogram stretching algorithm is used to stretch the histogram of an image using a transfer function that enhanced the contrast of image. This transfer function (see Fig. 2) can be used as a look-up table. Histogram of an image depicts the contrast of the image. A narrow histogram means low dynamic range which corresponds to the low contrast. A wide histogram means high dynamic range or high contrast. Section III-A2 shows the optimization on histogram stretching using lookup table. Vectorization of histogram stretching using lookup table is used to take the advantage of both optimization and vectorization to achieve higher performance on SIMD based architecture. Sample scalar pseudo code of histogram stretching for single channel is as follows

```
/* Scalar code: Histogram stretching using lookup
   "LUT_stretch" of transfer function */
for(i=0; i<height; i++)
    for(j=0; j<width; j++){
        temp = image[i][j];
        res = LUT_stretch[templ];
        image[i][j] = res;
    }
```

Vectorized pseudo code for above scalar code is:

```
/* Vectorized code: Histogram stretching using
   lookup LUT_stretch */
for(i=0; i<height; i++)
    for(j=0; j<width; j=j+n){
        vecTemp[1:n] = image[i][j:j+n-1];
        /* Selection is used here to get correct values
           from lookup table */
        vecRes[1:n] = LUT_stretch[vecTemp[1:n]];
        image[i][j:j+n-1] = vecRes[1:n];
  }
```

*4) Smoothing:* Smoothing is a process to reduce the effect of noise in image. There are many methods to do smoothing like linear filter includes Uniform filter, Gaussian filter and Triangular filter, and non-linear filter includes Median filter etc.. A sample scalar pseudo code for smoothing using Uniform filter of kernel size $b \times b$ is as follows

```
filter_size = b * b;
for(i= b/2; i<height-b/2; i++)
    for(j=b/2; j<width-b/2; j++){
        avg = 0;
        for(k=-b/2; k<=b/2; k++)
            for(l=-b/2; l<=b/2; l++){
                avg += imageIn[i+k][j+l];
            }
        avg = avg/filter_size;
        imageOut[i][j] = avg;
    }
```

Vectorized pseudo code for above scalar code is:

```
filter_size = b * b;
for(i= b/2; i<height-b/2; i++)
    for(j=b/2; j<width-b/2; j++){
        vecAvg[1:n] = 0;
        for(k=-b/2; k<=b/2; k++)
            for(l=-b/2; l<=b/2; l++){
                vecTemp[1:n] = imageIn[i+k][j+l:j+l+n-1];
                vecAvg[1:n] += vecTemp[1:n];
            }
        vecAvg[1:n] = vecAvg[1:n]/filter_size;
        imageOut[i][j:j+n-1] = vecAvg[1:n];
    }
```

### C. Memory Constraints Environment

Fog removal algorithm has many blocks of image and sizes of these blocks are big enough to not to be fit in low memory space system. Which indicate that the entire block of image cannot be processed in one go. To fit this algorithm on such high memory constrained environment, tilling approach is used, in which each block is divided into tiles and processed every tile, like that entire block can be processed. DMA is used to move the tiles from SRAM to DRAM and vice-versa. Tilling approach is shown in Fig. 5. It depicts the scenario, in which tiling approach is used to process the entire block of an image stored in SRAM memory space that cannot be processed in small DRAM memory space.

## IV. Results

Simulation and porting of the above proposed approaches are performed on different platforms like CPU and DSP processors and achieved the better performance. We have

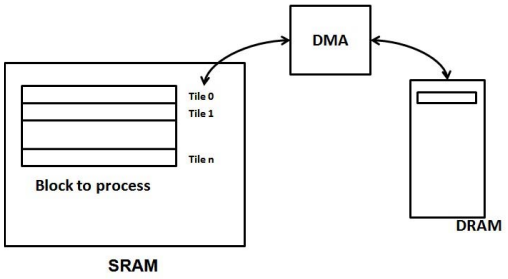


Fig. 5: Tilling approach

analyzed the performance qualitatively and quantitatively. Qualitative results are shown in Figs. 6, 7 and 8. These results depict the test input images and output images with different approaches. Quantitative analysis is shown in Table I, II and III. Performances of these approaches are compared in terms of computation time, contrast gain and perceptual quality matrix (PQM) [11]. Here in Table I, computation time is estimated for OFR, FRPO, FRPV, FRPOV, and FRPOVT on CPU and DSP platforms (OFR - Original Fog Removal by Tripathi et al, FRPO -Fog Removal with Proposed Optimization, FRPV -Fog Removal with Proposed Vectorization, FRPOV -Fog Removal with Proposed Optimization and Vectorization, FRPOVT - Fog Removal with Proposed Optimization Vectorization and Tilling, NA- Not Applicable. All values are in second. Less value indicates high performance).

TABLE I: Comparison between OFR, FRPO, FRPV, FRPOV, and FRPOVT on CPU and DSP platforms.

| **Platforms** | Approaches | | | | |
|---|---|---|---|---|---|
| | **OFR** | **FRPO** | **FRPV** | **FRPOV** | **FROPVT** |
| **CPU i3 3.1 GHz** | 0.22 | 0.12 | NA | NA | NA |
| **DSP 600 MHz** | 6.667 | 3.095 | 0.015 | 0.0113 | NA |
| **DSP 60 MHz low memory** | - | - | 1.667 | 0.933 | 0.106 |

TABLE II: Contrast gain for OFR and FRPOV on CPU and DSP platforms respectively.

| **Image** | **Input** | **OFR** | **FRPOV** |
|---|---|---|---|
| **test1** | 0.0186 | 0.0686 | 0.0717 |
| **test2** | 0.0235 | 0.0641 | 0.0715 |
| **test3** | 0.0648 | 0.0781 | 0.0900 |

TABLE III: Perceptual Quality metric (PQM) for OFR and FRPOV on CPU and DSP platforms respectively.

| **Image** | **OFR** | **FRPOV** |
|---|---|---|
| **test1** | 9.3569 | 9.0726 |
| **test2** | 9.7400 | 9.3978 |
| **test3** | 9.7040 | 9.7797 |

Table I shows the improvement in performance on each individual platform. Vectorization of fog removal algorithm for CPU intel $i3$ processor is not done, that's why values for approaches FRPV, FROPV and FROPVT are $NA$. On DSP $60$ MHz platform values are in minutes for approaches OFR and FRPO, because processor is specially design for vector operation and original fog removal is based on scalar operation, which gives very poor performance on DSP $60$ MHz platform. In the Table I, entries in CPU and DSP $600$ MHz processor for approaches OFR and FRPO, values in FRPO are nearly $2$ times lesser than that of OFR approach for respective processors. From here, it is clear that only proposed optimization gives two times better performance. Values in DSP $600$ MHz processor for approaches FRPV and FRPOV proves the improvement of algorithm over vectorization. Value in DSP $60$ MHz low memory space processor for approach FRPOVT is nearly $9$ times lesser than that of FRPOV , which proves that tilling approach improves the performance in low memory constraint. Performance in terms of contrast gain is shown in Table II. It is noted that clear day or non-foggy images have more contrast in comparison with foggy images. Thus gain in contrast for any fog removal algorithms should be positive. Higher gain in contrast shows better performance. Performance in terms of PQM is shown in Table III, this metric computes (a) discontinuities at block boundaries in horizontal and vertical direction, (b) activity in images expressed by deviations of gradients from their respective block discontinuities, and (c) the no. of zero-crossings in these gradient. According to [11] the PQM value should be close to 10 for best perceptual quality. Results verify that our optimization approaches do not degrade the performance significantly.

## V. CONCLUSION

In this paper, various approaches are proposed for optimization and vectorization of fog removal algorithm. Results verify that the proposed approaches improve the performance significantly. As proposed optimization techniques give two times of better performance, but causes negligible change in quality (in terms of the contrast gain and PQM), because of the conversion from $24$-bits float to $16$-bits integer to use the lookup table for diffusion. If, realtime performance is a bottleneck then proposed optimization approaches are very effective. Proposed vectorization approaches achieved a much higher improvement on DSP platform, which gives a sign that proposed approaches are effective as well as efficient. Proposed tilling approach is also effective in low memory space. Overall the proposed approaches are very effective individually and/or combined. Results confirm that with the proposed optimization and vectorization approaches, performance of the fog removal algorithm is increased up to $90$ fps (approximately) for VGA image on DSP platform. Proposed approaches have wide range of application for realtime implementation of various computer vision and machine learning algorithms. In future, research will focus on the memory optimization for further improvement in the performance.

(a) (b) (c) (d)

Fig. 6: Results of fog removal algorithm of optimization and vectorization. (a) Original 'test1' Image, result of (b) OFR algorithm, (c) FRPO approach, (d) FRPOV approach.

(a) (b) (c) (d)

Fig. 7: Results of fog removal algorithm of optimization and vectorization. (a) Original 'test2' Image, result of (b) OFR algorithm, (c) FRPO approach, (d) FRPOV approach.

(a) (b) (c) (d)

Fig. 8: Results of fog removal algorithm of optimization and vectorization. (a) Original 'test3' Image, result of (b) OFR algorithm, (c) FRPO approach, (d) FRPOV approach.

## REFERENCES


[1] A.K. Tripathi and S. Mukhopadhyay, "Single image fog removal using anisotropic diffusion" in IET image processing Oct, 2011, 6, pp. 966 - 975.

[2] Narasimhan, S. G., and Nayar, S. K., "Chromatic Framework for Vision in Bad Weather", in IEEE Conference on Computer Vision and Pattern Recognition, 2000, 1, pp. 598-605.

[3] Tarel, J. P., and Hautiere, N., "Fast visibility restoration from a single color or gray level image", IEEE International Conference on Computer Vision, 2009, pp. 2201-2208.

[4] Fattal, R., "Single image dehazing", in International Conference on Computer Graphics and Interactive Techniques archive ACM SIGGRAPH", 2008, pp. 1-9.

[5] Tan, R. T., "Visibility in bad weather from a single image", in IEEE conference on Computer Vision and Pattern Recognition, 2008, pp. 1-8.

[6] Kopf, J., Neubert, B., Chen, B., Cohen, M., Cohen-Or, D., Deussen, O., Uyttendaele, M., and Lischinski, D., "Deep photo : Model-based photograph enhancement and viewing", in ACM Transactions on Graphics, 2008, 27, 5, pp. 116:1-116:10.

[7] Perona, P. Malik, J., "Scale-space and edge detection using anisotropic diffusion", in Pattern Analysis and Machine Intelligence IEEE Transactions, 1990, 12, 7, pp. 629 - 639.

[8] Joachim Weickert, "Anisotropic Diffusion in Image Processing" Germany, ECMI Series, Teubner-Verlag, Stuttgart, 1998.

[9] M. Weinhardt and W. Luk, "Pipeline vectorization" in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, November 2006, 20, 2, pp 234-248.

[10] S. Biswas, N. R. Pal and S. K. Pal, "Smoothing Of Digital Images Using The Concept Of Diffusion Process" in Pattern Recognition, Elsevier , March, 1996.

[11] Z. Wang, H. R. Sheikh, and A. C. Bovik, "No-Reference Perceptual Quality Assessment of JPEG Compressed Images", in IEEE International Conference on Image Processing, 2002, vol. 1, pp. 477-480.