A Study on Acceleration of Image Smoothing and FPGA Implementation of Image Enhancement and Haze Removal as an Application of Image Smoothing

Submitted to
Graduate School of Information Science and Technology
Hokkaido University

March 2019

Dabwitso KASAUKA

Abstract

In recent years, much research interest in digital image and video processing has developed focusing on various aspects. Some of these include image smoothing, enhancement and haze removal. With continued advancement in development and deployment of mobile consumer electronics capable of capturing images and streaming video, the need for image processing technique optimization geared to support real-time processing is increasing. Based on this, we investigate image smoothing techniques, Retinex-based image enhancement and haze removal optimization for real-time processing. We further propose an FPGA architecture to support efficient real-time processing. Our research work has been done in two segments; 1) Image smoothing, 2) Retinex-based enhancement and haze removal with FPGA implementation. Our research in the first segment compliments the second segment.

Image smoothing decomposition techniques basically separate a digital image into its structure and texture layers, hence the term smoothing as the structure can exist apart from details. Such techniques have found relevance in various image processing applications such as edge detection, detail enhancement, image tonal mapping, image compression artifact removal, image noise removal, and in computer vision object imaging/ robotics. It has also found application in the field of optical measurement as a calibration method in image sensing. In this thesis, implementation of image smoothing using spatial iterative methods is investigated. We formulate a smoothing algorithm suitable for spatial implementation and compare our proposed multigrid preconditioned conjugate gradient implementation with existing smoothing algorithms. Furthermore, we take into account the image boundaries and eliminate wrap around errors which are inherent in some existing smoothing algorithms including those implemented using Fast Fourier Transform solvers. According to experimental results obtained using various image datasets, our approach computes smoothed output in competitively low processing time. Some existing methods such as based on bilateral filtering, tree filtering and weighted least squares, converge to a solution faster than our proposed approach. However, according to qualitative results obtained, our approach produces better smoothing results. A full HD image is processed in 0.85 secs. In further optimization, we

incorporated image downsampling in one case, and in another case implemented flow optimization by pre-calculating and loading the computational costly Laplacian operator matrix from memory. In the case of downsampling, processing time was reduced by approximately 21.6 %. However, a trade-off between processing time and smoothing quality was encountered in this case. Downsampling optimization may be adopted in applications utilizing low processing power and memory hardware. In the case of flow optimization, processing time was reduced by approximately 46.1 % against existing smoothing algorithms. The smoothing quality in this case is not affected as this simply involves computation flow optimization. This optimization approach is useful and applicable to constant resolution input stream, extending application to video processing.

Image enhancement and haze removal optimization are key research topic in intelligible information gathering and classification. Environmental illumination conditions play a major role in influencing visual perception and scene classification. The quality of images and video taken from outdoor scenes is influenced by scattering of light which occurs before reaching the camera sensor. The amount of scattering depends on the distance between the scene points and the sensor, making degradation spatial-variant. In haze (fog) weather, an elevated presence of atmospheric particles such as water-droplets results in more scattering, resulting in low contrast and colour fidelity images. Scattering is caused by two basic phenomena, which are attenuation and airlight. Haze removal depends upon the unknown depth information. This particularly makes haze removal a challenging task. Haze removal is highly desired in computer vision applications. It not only serves to significantly increase the visibility of the scene and correct the colour shift, it can also benefit many vision algorithms and advanced image editing. Both Retinex-based image enhancement and haze removal are computation costly. Considering real-time processing in applications such as monitoring systems, autonomous cars, and live streaming systems, there still remains much room for the development of efficient hardware implementation of image enhancement and haze removal. Motivate by this, we propose an architecture supporting both real-time Retinex-based image enhancement and haze removal, at low memory and process overhead utilizing a single module. The implementation results reveal that just 1 % logic circuits overhead is required to support Retinex-based image enhancement in single mode and haze removal based on Retinex model. This reduction in computation complexity by using a single module reduces the processing and memory implications especially in mobile consumer electronics, as opposed to implementing them individually using different modules. Furthermore, we utilize image enhancement for transmission map estimation instead of soft matting, thereby avoiding further computation complexity which would affect our goal of realizing high frame-rate

real time processing. Our FPGA implementation, operating at an optimum frequency of 125 MHz with 5.67 M total block memory bit size, supports WUXGA (1,920x1,200) 60 fps as well as 1080p60 color input. The maximum logic utilization and number of registers used are 3212 and 3648, respectively. At high frequency of 240 MHz, our approach has the capability of processing 4K video at 30 fps. We compare our approach to existing state-of-the-art algorithms, both quantitative and qualitatively. Performing PSNR and SSIM tests on multiple inputs, it is observed in average that our approach provides better results. In terms of hardware architecture performance evaluation with existing FPGA architectures, our approach provides the highest throughput of 125 Mpixels/s, utilizing 9 line buffers of 240 width size.

Acknowledgements

I thank my Lord and Savor Jesus Christ, for it is through Him that all things have been made possible. By His love and grace, I am alive to complete my study program in great health and success.

I hereby acknowledge Prof. Y. Miyanaga for his support and guidance though my period of research in Information Communication Networks lab.

I further acknowledge associate professor H. Tsutsui for working close to me through research, direction and refinement. By him, I have learnt valuable aspects of investigative research and innovation.

I further extend my appreciation to associate professor M. Convertino, professor T. Ohgane and professor K. Saito for their support during my study program. Their insight and counsel has been valuable, helping me to complete my program.

I thank my loving wife Chileshe Kasauka for her great support and encouragement during my entire period of study. Without her support, I would not enjoy this success.

I also thank the GI-CoRE GSB, Hokkaido University for fruitful discussions. This work is supported in parts by the Ministry of Internal Affairs and Communications for SCOPE Program (185001003). This work is also supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc., Cadence Design Systems, Inc., and Mentor Graphics, Inc.

Publications

Transactions

1. Dabwitso Kasauka, Kenta Sugiyama, Hiroshi Tsutsui, Hiroyuki Okuhata, Yoshikazu Miyanaga, “An Architecture for Real-time Retinex-based Image Enhancement and Haze Removal and its FPGA Implementation,” IEICE Transaction Fundamentals of Electronics, Communications and Computer Sciences, Vol.E102-A, No.6, June 2019.

Conference Papers with Referee

1. Dabwitso Kasauka, Hiroshi Tsutsui, Seijiro Imai, Takashi Imagawa, Hiroyuki Okuhata, Yoshikazu Miyanaga, ”Image Smoothing in the Spatial Domain Using Multigrid Conjugate Gradient Methods Based on Accelerated Iterative Shrinkage,” Proceedings of Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pp. 1-5, Dec. 2016.

2. Seijiro Imai, Dabwitso Kasauka, Hiroshi Tsutsui, Takashi Imagawa, Hiroyuki Okuhata, Yoshikazu Miyanaga, ”Processing Time Reduction of Tone Mapping Based on Iterative Shrinkage Smoothing Using Parallel Processing,” Proceedings of International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), pp. 1-4, Oct. 2016.

3. Dabwitso Kasauka, Hiroshi Tsutsui, Hiroyuki Okuhata, Takashi Imagawa, Yoshikazu Miyanaga, “Image Smoothing Using Spatial Iterative Methods Based on Accelerated Iterative Shrinkage,” Proceedings of Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pp. 779-783, 16-19 Dec. 2015.

4. Dabwitso Kasauka, Hiroshi Tsutsui, Hiroyuki Okuhata, Yoshikazu Miyanaga,

”Computational Cost Analysis and Implementation of Accelerated Iterative Shrinkage Smoothing,” Proceedings of Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pp. 1-4, 9-12 Dec. 2014.

Conference Papers without Referee

1. Seijiro Imai, Dabwitso Kasauka, Takashi Imagawa, Hiroshi Tsutsui, Hiroyuki Okuhata, Yoshikazu Miyanaga, ”An Evaluation of Frame-based Parallel Processing for Iterative Shrinkage Smoothing Using Multi-core Processor,” Proceedings of IEICE Hokkaido Section Student Council Internet Symposium, Feb. 2016 (in Japanese).

2. Seijiro Imai, Dabwitso Kasauka, Takashi Imagawa, Hiroshi Tsutsui, Hiroyuki Okuhata, Yoshikazu Miyanaga, ”Processing Time Reduction of Iterative Shrinkage Smoothing Using Parallel Processing,” IEICE Technical Report, Vol. 115, No. 348, pp. 57-60, SIS2015-39, 3-4 Dec. 2015 (in Japanese).

3. Seijiro Imai, Dabwitso Kasauka, Takashi Imagawa, Hiroshi Tsutsui, Hiroyuki Okuhata, Yoshikazu Miyanaga, ”Computational Cost Reduction of Iterative Shrinkage Smoothing Based Image Enhancement,” IEICE Society Conference, p. 144, A-20-4, Sep. 2015 (in Japanese).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

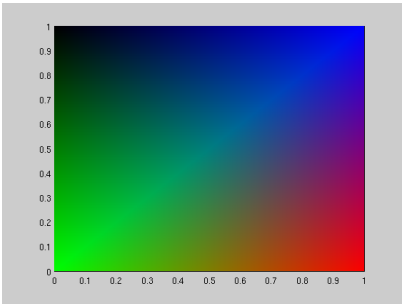
1.1 Background

Digital image processing is basically the manipulation of a picture image for various applications. It focuses on: improvement of pictorial information for human interpretation; processing of image data for storage, transmission and representation for autonomous machine perception (e.g. robotics).

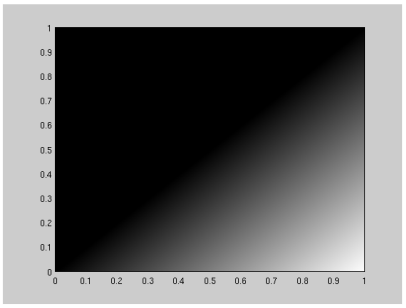
A image is basically a collection of points called pixels, of which values represent intensity of light ranging from black to white (i.e., 0 to 256 respectively). In the case of color images, generally RGB format, each pixel location is represented by a set of three values, which in combination produce the illusion of color. RGB means an image is represented by three individual layers containing red, green and blue information ranging from 0 to 256. This concept of a color image composed of layers of individual colors is illustrated in Fig. 1.1.

It can be noted in each individual layer, Fig. 1.1((b))((c))((d)), that the color represented therein has a greater intensity (256) at the point of origin (vertex of the rectangular image) and fades in towards the center where the intensity reduces (approaches 0). In the area about the center, it can be noted that none of the individual colors can be distinguished, however new shades of color are created. This is due to at least two or all three color layers contributing a relatively significant value to the pixel set. This in a nutshell is how images represent color captured in nature.

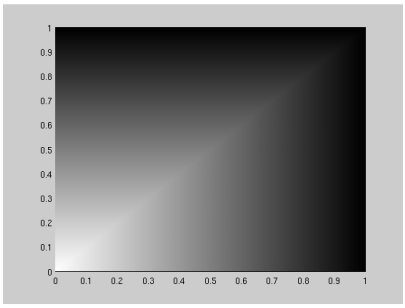
There multiply variations of image types: binary images whose pixel values is either 0 or 1(black or white) only; gray-scale images whose pixels are represented by only one value per pixel which ranges between 0 and 256, that is from black to white with shades of gray in between; color images whose pixels are represented by a set of generally three values, each ranging from 0 to 256, in the case of RGB color space which is the common representation of color images. An example of



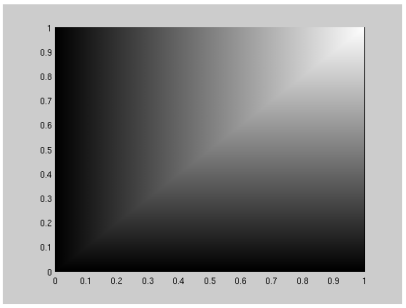
((a)) Basic RGB image



((b)) red layer



((c)) green layer



((d)) blue layer

Figure 1.1: Illustration of layers combined to create a single color image

a color, binary and gray-scale image representation of the same image is shown in Fig. 1.2((a)),((b)) and ((c)), respectively.

Digital image processing has found usage in various applications and fields. Some examples of these are:

- Human computer interfaces
- Medical visualization systems
- Image enhancement
- Image reconstruction
- Noise removal
- Law enforcement

((a)) Color image

((b)) Binary image

((c)) Gray-scale image

Figure 1.2: Examples of image representation

- Manufacturing industrial inspection

The key stages in digital image processing are shown in Fig. 1.3.

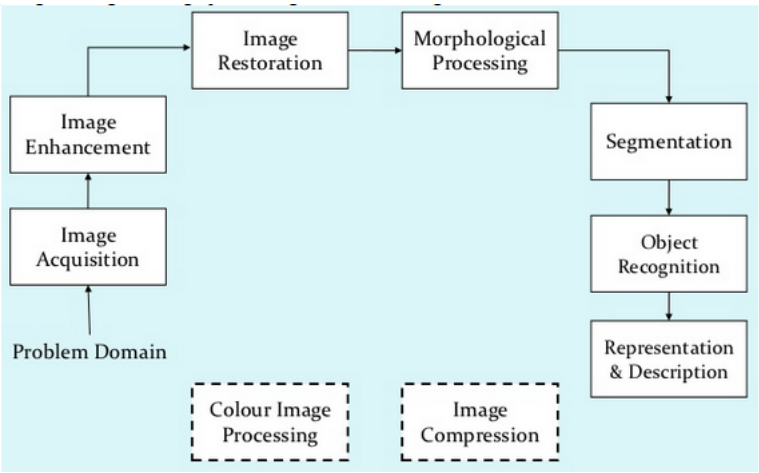


Figure 1.3: Components of digital image processing

Recently, much research interest has developed in image smoothing, image enhancement and haze removal techniques. With the growing need for efficient algorithms suitable for real-time processing and hardware optimization, we have been motivated to investigate this topics and formulate processing and memory efficient algorithms.

In image smoothing, the main challenge is to efficiently decompose the image based on homogeneous and non-homogeneous regions independently. Hence, the smoothing algorithm should correctly classify salent edges. Without correct classification and processing, salent edges are smoothed out resulting in sharpness reduction. Hence, a study on edge classification and homogenous processing is required in order to develop a smoothing algorithm. Furthermore, computational

costly classification methods are not recommended due to increase in processing time. Therefore, a compromise between smoothing fidelity (quality) and computation complexity is considered in algorithm design.

Due to algorithmic complexity, Retinex-based image enhancement and haze removal are implemented separately. Developing a new single module architecture to efficiently support both at low overhead hardware resource cost is a challenge. This requires a study on hardware architecture considering the specific characteristic of the algorithms. At the same time, design efficiency is necessary in order to realize a low memory and processing complexity. Furthermore, optimization to support real-time processing at low complexity cost should be taken into account.

1.2 Objectives of this thesis

The objective of this dissertation is to investigate and propose an accelerated image smoothing algorithm, and to design an FPGA architecture that efficiently supports both Retinex-based enhancement and haze removal using a single module. In both cases, we further focus on real-time processing. To approach this goal, primary issues to study in this thesis are;

- Efficient image decomposition, edge classification and boundary constraints,
- Minimization cost function optimization and iterative solvers,
- Retinex model, haze model and related estimation functions,
- hardware architecture design and optimization.

Firstly, Chapter 2 focuses on investigating image smoothing. In Section 2.2, we review some existing image smoothing algorithms based on weighted least squares, LO gradient minimization and multiscale detail decomposition. Image smoothing presents a complex minimization problem, which can be tackled in frequency or spatial domain. Frequency domain based solvers provide a faster computation convergence compared to spatial domain solvers. However, in certain conditions FFT solvers introduce wrap around errors about the boundaries of the smoothed image due to the inherent nature of FFT. Spatial domain solvers provide algorithmic control on the way image boundaries are handled. This is investigated and presented in Section 2.5. In Section 2.3, we investigate spatial iterative methods, identifying multigrid congugate gradient method as a suitable method due to the large system matrix problem to solve. We also provide comparitive results in Section 2.5 with other spatial solvers. Now the main focus is to propose an efficient problem formulation in order to solve the complex image smoothing problem. In Section 2.4, we

outline our proposed algorithm, which is implemented using multigrid conjugate gradient method. Compared with existing state-of-the-art smoothing algorithms, our approach is competitive in terms of processing time and smoothing quality. We manage to produce a better trade-off balance compared to other algorithms. With the goal to present an accelerated smoothing algorithm, we introduce two implementation optimization techniques.

- Scaling optimization: By introducing downsampling into the algorithm, we managed to reduce the processing time by approximately 21.6 %. In addition to processing time reduction, the use of downsampled images on which to perform image smoothing reduces the memory requirement. However, a trade-off between processing time and smoothing quality is introduced. By using simple downsampling algorithms, processing time advantage is gain at the expense of smoothing quality. The use of more complex scaling algorithms would improve the quality with less processing time gain.

- Flow optimization: This involves the pre-calculation and storage of computationally costly Laplacian operator matrix, which depends only on the resolution of the input. This is applicable to constant resolution input streams, extending its application to video processing. By pre-calculating and storing this operator only once, inputing images having the same resolution results in a processing time reduction of approximately 46.1 %.

The selection of which optimization technique to use depends on the requirements of the application, including the hardware resources proposed.

Secondly, Chapter 3 investigates Retinex-based enhancement and haze removal with a proposed FPGA architecture design. Section 3.2 briefly highlight some related existing algorithms. Retinex image enhancement and haze removal are two computational complex algorithms, which are often times investigated and implemented independently. In Section 3.4, we propose an efficient FPGA architecture capable of supporting both Retinex-base enhancement and haze removal using a single module. Our further objective is to provide a real-time processing architecture capable of processing 1920×1200 input at 60 fps. According to experimental results, our architecture supports this desired input, with capability of supporting 4K video at 30 fps. The hardware parameters used are; frequency of 125 MHz, 9 line buffers used for filtering each with a width of 240, 3648 registers and 366 RAM blocks. Using PSNR and structural similarity index matrix (SSIM), we evaluate the performance of our proposed algorithm and FPGA architecture against existing algorithms. Both quantitative and qualitative results reveal that our approach provides better results. Finally, Chapter 4 summarizes and concludes this thesis.

Chapter 2

Image Smoothing

2.1 Introduction

Image smoothing is basically the removal of fine details while maintaining the integrity of the major image structure. According to [3], image smoothing can be defined as a process of effectively sharpening major edges by increasing the steepness of transition while eliminating a manageable degree of low-amplitude structures. While in [4], it is stated that many computer vision tasks require smoothing as a preprocessing operation to reduce image noise. Many applications in image processing and computer vision require decomposing an image into a piecewise smooth base layer and a detail layer. The base layer captures the main structural information, while the detail layer contains the residual smaller scale details in the image.

Digital image smoothing research has become increasingly popular in recent years. A good smoothing algorithm should have the ability of reducing noise (undesired components), while at the same time preserving important details, such as edges. Edge preserving can be achieved via energy-minimizing, surface fitting, and weighted averaging [4]. Several edge-preserving smoothing methods have been proposed [5–15], and can be classified into two groups . The first group consists of the edge-preserving filters that explicitly compute a filtering output as a weighted average [16]. Bilateral filtering is a basic example, widely adopted for its simplicity and effectiveness in removing noise-like structures. These are typically efficient techniques, often giving a linear-time complexity dependent on the number of image pixels only. As highlighted in [16], a common limitation of these essentially local filters [5, 12, 17–27] is that they cannot completely resolve the ambiguity regarding whether or not to smooth certain edges.

The second group of edge-preserving smoothing methods are based on global

optimizations, solving for a globally optimal solution usually involving minimization cost function with a prior smoothness term [28–35]. These techniques provide better results compared to the local filter group discussed early. However, there is a trade-off between quality and computation complexity. More computation resources are required in order to implement smoothing techniques using global optimization, mainly arising from solving a large linear system. Hence, local filtering techniques perform faster compared to global filtering techniques while sacrificing on quality.

Image smoothing can be implemented using both frequency and spatial domain techniques. In frequency domain, high frequencies correspond to quickly changing regions (edges), while low frequencies correspond to slowly changing regions (smooth regions). To understand edge aware smoothing, let us first look as two most basic building block filters, namely low pass and high pass filters.

- Low pass filter leaves the low frequencies unchanged. This in it's simplest form smooths the image; as blocked high frequencies correspond to sharp intensity changes, i.e. to the fine-scale details and noise in the spatial domain image. This is good for removing noise (sudden discontinuities in an image), but blurs the image as its edges are equally modified as they are regions of discontinuity.

- A high pass filter leaves the high frequencies unchanged. This thereby sharpens the edges and is good for edge detection. Areas of rather constant gray-level consist of mainly low frequencies and are therefore suppressed.

Hence, in layman's terms, edge aware smoothing employs techniques which have a hybrid characteristic of both a low and high pass filters, tuned according to a specific application. Fig. 2.1 illustrates the concept of manipulation of frequencies in frequency domain, in the case of applying low pass and high pass filtering, as well as an edge-aware image smoothing technique(our subject of interest).

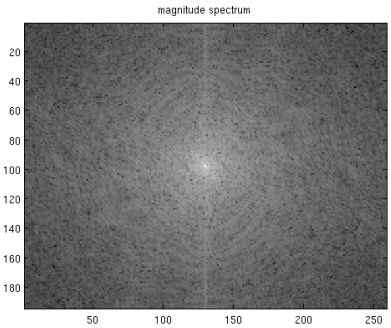
As can be observed simply from Fig. 2.1((g)) and ((h)), edge-aware smoothing filters somehow combine desirable characteristics of both low and high pass filters. From henceforth, when we mention image smoothing, we refer to edge-aware smoothing techniques.

In the case of spatial domain implementation of image smoothing, what is involved is the direct manipulation of image pixels in spatial domain. As many researchers have developed and implemented image smoothing using frequency domain techniques/ solvers, hence-from our motivation to implement image smoothing in spatial domain in order to create a baseline of performance.

One would ask, why are we interested in image smoothing as the smoothing output shown in Fig. 2.1((a)) is not as visually appealing as the original image in

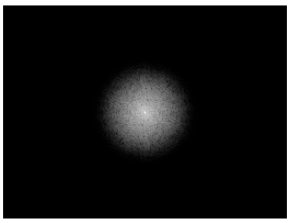
Fig. 2.1((g))? Well, as illustrated in [1, 3, 4, 36], image smoothing has applications in detail enhancement, HDR tone mapping, edge enhancement and extraction, image abstraction and pencil sketching, clip-art compression artifact removal, noise removal and layer-based contrast manipulation, just to highlight a few. According to [37], smoothing can be combined with segmentation techniques in order to produce superior segmentation results, important to many computer vision applications. Some of these applications are shown in Fig. 2.2.

((a)) Original input image

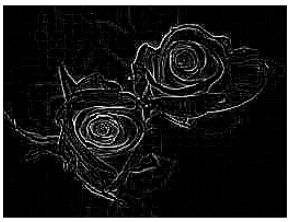


((b)) FFT magnitude spectrum of (a)

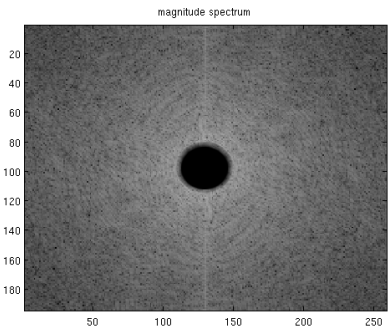
((c)) low pass filtered image



((d)) low pass filter applied to (b)

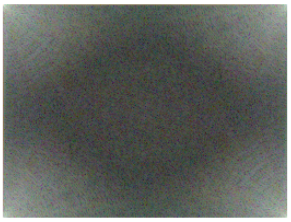


((e)) High pass filtered image



((f)) High pass filter applied to (b)

((g)) edge-aware smoothed image



((h)) Edge-aware filter applied to (b)

Figure 2.1: FFT illustration of image filtering, showing how filter masks affect an image

((a)) Detail enhancement

((b)) Clip-art compression artifact removal (image obtained from [3])

((c)) Edge extraction process stabilized by image smoothing (image obtained from [3])

Figure 2.2: Example applications of image smoothing

2.2 Related Works

There are various image smoothing techniques proposed by researchers, some of which are outlined in [4, 31, 38]. Taking a leaf from *novel smoothing technique* research developer [4, 5, 39–41], who utilized information from existing smoothing techniques upon which they built, we henceforth decided to identify an already existing algorithm by which we can achieve our desired objective. Our criteria of selection, upon investigative research, simply comes down to the algorithm possessing a fast convergence rate (least iterations) while producing a good edge-aware smoothed output, significantly void of blur or artifacts. We also kept in mind the flexibility of an algorithm to be modeled and solved using a spatial solver.

We briefly describe some related works focusing on image smoothing techniques in the following subsections.

2.2.1 Fast Global Image Smoothing Based on Weighted Least Squares

In a paper presented by Dongbo *et al.* [16], a technique for performing spatially inhomogeneous edge-preserving image smoothing is outlined. Focusing on sparse Laplacian matrices consisting of a data term and a prior term (typically defined using four or eight neighbors for 2D image), such global objective functions are solved by approximating the solution of an intensive large linear system. This linear system, defined over a *d*-dimensional spatial domain is approximated by solving a sequence of 1D sub-systems.

Given an input image f and a guidance image g, a desired output u is obtained by minimizing the following weighted least squares (WLS) energy function:

$$J(u) = \sum_{p} \left((u_p - f_p)^2 + \lambda \sum_{q \in N_{(p)}} w_{p,q}\,(g)\,(u_p - u_q)^2 \right) \tag{2.2.1}$$

where f_p and g_p represent a color (or grayscale) value of (x, y), and $\Omega_D = \{p = (x, y) | 0 \leq x < W, 0 \leq y < H\} \subset \Re^2$. $N(p)$ is a set of neighbors of a pixel

p, and λ controls the balance between the two terms. Increasing λ results in more smoothing in the output u. The smoothness constraint is adaptively enforced using the spatially varying weighting function $w_{p,q}(g)$ defined with g. $w_{p,q}$ represents a similarity between two pixels p and q.

Setting the gradient of $J(u)$ to zero, the minimizer u is obtained by solving a linear system based on a large sparse matrix:

$$(I + \lambda A)\, u = f \tag{2.2.2}$$

where u and f denote $S \times 1$ column vectors containing color values of u and f, respectively. **I** denotes an identity matrix and **A** represents a spatially-varying Laplacian matrix witha size of $S \times S$.

In this approach, a solution of an objective function defined on weighted L_2 norm 2.2.1 by decomposing it into each spatial dimension and solving the matrix with a sequence of 1D fast solvers. Alg. 1 shows this smoothing algorithm on a 2D image using the 1D solver.

When the smoothing parameter λ (representing the total amount of spatial smoothing) and the total number of iterations T are given as inputs, the separable 1D smoothing operations are performed along the horizontal and vertical directions with λ_t computed at the t^{th} iteration.

2.2.2 Image Smoothing via L0 Gradient Minimization

In Xu *et al.* [42] [43], an algorithm which sharpens major edges by increasing the steepness of transition whilde eliminating a manageable degree of low-amplitude structures is introduced. This is achieved by using L_0 gradient minimization, which can globally control how many non-zero gradients are resulted. Let input image be denoted by I and output computed image by S. The gradient $\nabla S_p = (\delta_x S_p, \delta_y S_p)^T$ for each pixel p is calculated as color difference between neighboring pixels along the x and y directions. The gradient measure is given by,

$$C(S) = \#\{p \big| \mid \delta_x S_p \mid + \mid \delta_y S_p \mid \neq 0\} \tag{2.2.3}$$

counting p whose magnitude $|\delta_x S_p| + |\delta_y S_p|$ is not zero. With this definition, S is estimated by solving

$$\min_S \left\{ \sum_p (S_p - I_p)^2 + \lambda \cdot C(s) \right\} \tag{2.2.4}$$

To simplify the problem by approximation, auxiliary variables h_p and v_p, corresponding to $\delta_x S_p$ and $\delta_y S_p$ respectively are introduced, rewriting the objective func-

tion as

$$\min_{S,h,v}\left\{\sum_{p}\left(S_p-I_p\right)^2+\beta\left(\left(\delta_x S_p-h_p\right)^2+\left(\delta_y S_p-v_p\right)^2\right)\right\} \tag{2.2.5}$$

where $C(h,v)=\#\{p\big||h_p|+|v_p|\neq 0\}$ and β is an automatically adapting parameter to control the similarity between variables (h,v) and their corresponding gradients. Applying Fast Fourier Transform yields a solution

$$S=\mathcal{F}^{-}1\left\{\frac{\mathcal{F}(1)+\beta(\mathcal{F}(\delta_x)*\mathcal{F}(h)+\mathcal{F}(\delta_y)*\mathcal{F}(v))}{\mathcal{F}(1)+\beta(\mathcal{F}(\delta_x)*\mathcal{F}(\delta_x)+\mathcal{F}(\delta_y)*\mathcal{F}(\delta_y))}\right\} \tag{2.2.6}$$

where $\mathcal{F}$ is the FFT operator and $\mathcal{F}()*$ denotes the complex conjugate. $\mathcal{F}(1)$ is the Fourier Transform of the delta function. Solving for (h,v), the following is computed

$$E_p=\left\{\left(h_p-\delta_x S_p\right)^2+\frac{\lambda}{\beta}H\left(|h_p|+|v_p|\right)\right\} \tag{2.2.7}$$

which reaches its minimum E_p^* under the condition

$$(h_p,v_p)=\begin{cases}(0,0) & (\delta_x S_p)^2+(\delta_y S_p)^2\leq\lambda/\beta\\ (\delta_x S_p,\delta_y S_p) & \text{otherwise}\end{cases} \tag{2.2.8}$$

2.2.3 Fast Multi-scale Detail Decomposition via Accelerated Iterative Shrinkage

In this section, we briefly describe the underlaying principles involved in [1]. This algorithm performs edge-aware smoothing by running very few gradient shrinkage-reconstruction iterations. The basis of this algorithm is the use of first order proximal operators [44] as a way to accelerate the shrinkage scheme. According to [1], a smoothing problem is formulated as

$$\underset{u}{\operatorname{argmin}}\frac{\lambda}{2}\left\|u-g\right\|_2^2+\psi(\nabla u), \tag{2.2.9}$$

where g is input image, u is smoothed outpur, $\psi(\nabla u)$ is a gradient function, and λ is a positive regularization term. Problem (2.2.9) can be simplified by applying half-quadratic form, and solved by iterative minimization as follows,

$$v^{(k+1)} \leftarrow \underset{v}{\operatorname{argmin}}\,\psi(v)+\frac{\beta}{2}\left\|\nabla u^{(k)}-v\right\|_2^2, \tag{2.2.10}$$

$$u^{(k+1)} \leftarrow \underset{u}{\operatorname{argmin}}\,\lambda\left\|u-g\right\|_2^2+\beta\left\|\nabla u-v^{(k+1)}\right\|_2^2, \tag{2.2.11}$$

where $v^{(k+1)}$ corresponds to a shrinkage operation, $u^{(k+1)}$ corresponds to the screened Poisson equation [45], and β is the new regularization term. By applying first order proximal operator, $v^{(k+1)}$ can be simplified to

$$v^{(k+1)} \leftarrow \nabla u^{(k)} \circ f\left(\nabla u^{(k)}\right), \tag{2.2.12}$$

where f is the shrinkage weight. For a color image, the following shrinkage weights can be used,

$$\begin{aligned} f_1(T(\nabla u)) &= 1 - \frac{1}{1 + (T(\nabla u)/\gamma)^{\alpha}}, \\ f_2\left(T(\nabla u)\right) &= 1 - e^{-(T(\nabla u)/\gamma)^{\alpha}}, \end{aligned} \tag{2.2.13}$$

where α and γ are positive parameters, and $T(\nabla u)$ is given by

$$T(\nabla u) = \sqrt{\left(\sum_{k=1}^{\mathrm{ch}} \left|\frac{\partial u_k}{\partial x}\right|\right)^2 + \left(\sum_{k=1}^{\mathrm{ch}} \left|\frac{\partial u_k}{\partial y}\right|\right)^2}, \tag{2.2.14}$$

where u_k corresponds to a color channel of u. From Eq. (2.2.11), $u^{(k+1)}$ is solved using fast Fourier transform and given by

$$u^{(k+1)} \leftarrow \mathcal{F}^{-1}\left(\frac{\mathcal{F}\left(\lambda g - \beta \operatorname{div}(v^{(k+1)})\right)}{\lambda - \beta \operatorname{lap}}\right), \tag{2.2.15}$$

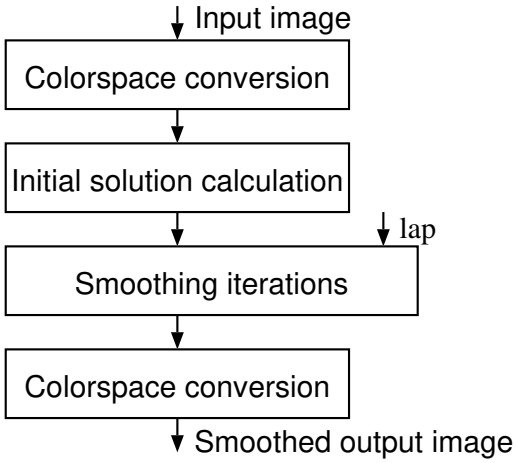
where div is the discrete divergence, and $\mathcal{F}$ and $\mathcal{F}^{-1}$ are forward and inverse fast Fourier transforms, respectively. In Eq. (2.2.15), lap is independent of the input image characteristics. It is basically the optical transfer function of a discrete Laplacian filter such as

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

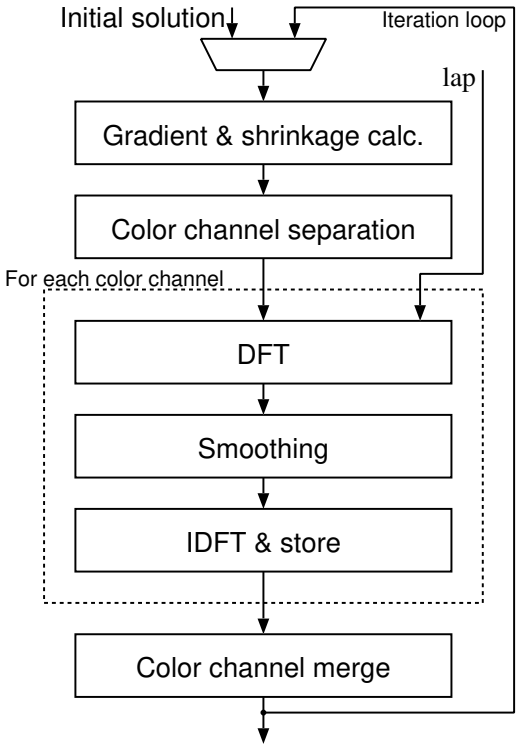
Finally, an initial solution calculation prior to the smoothing iteration is performed as a way to speed up the smoothing process [1]. This is approximated by,

$$u^{(0)} \approx g + \xi \operatorname{div}(\triangledown g - \triangledown \circ f_i(T(\triangledown g))). \tag{2.2.16}$$

According to [46], compared with a monochrome image, the own characteristics of a color image is its color components. If the luminance component and color component of a color image can be respectively described, not only can more useful information be gained from the color image, but also the processing technologies



((a)) Flow of the smoothing algorithm.



((b)) Flow of the smoothing iteration stage in Fig. 2.3(a).

Figure 2.3: Flow of algorithm proposed in [1]

used to process the color image. Hence, in this algorithm, smoothing is performed in CIELab color space.

We summarize the algorithm flow as shown in Fig. 2.3((a)), with detailed expansion of the smoothing iteration stage represented in Fig. 2.3((b)).

In [47], we performed a computational cost analysis of [1]. It provides a detailed computational breakdown, which serves as a foundation to better understand the processing involved in image smoothing, as it also provides information to researchers interested in altering/ improving the outlined smoothing technique in, presented in this section.

Algorithm 1 Separable global smoother for 2D image smoothing

Target Operation: $u = (I + \lambda A)^{-1} f$
Input: 2D image $f(x, y)$; $f(S \times 1$ vector$)$
Input: 2D guide image $g(x, y)$; $g(S \times 1$ vector $)(g = f$ for image filtering, $g \neq f$ for joint filtering)
Output: 2D image $u(x, y)$; $u(S \times 1$ vector$)$
Parameters and Notations
T: iteration num., $S = HW$: image size
λ: smoothing parameter
Algorithm
Initialize $u \leftarrow f$
for $t = 1 :$ T **do**
  compute λ_t
  for $y = 0 : H - 1$ **do**
    $f^h(x) \leftarrow u(x, y)$ for all $x = 0, \ldots, W - 1$
    if (image filtering) **then**
      $g^h(x) \leftarrow f^h(x)$ for all x compute $w_{x,i}$ using g^h for $i \in N_h(x)$
      build a tridiagonal matrix A_h
      solve $(I_h + \lambda_t A_h)u_h = f_h$
      $u(x, y) \leftarrow u^h(x)$ for all $x = 0, \ldots, W - 1$
    end if
  end for
  for $x = 0 : W - 1$ **do**
    $f^v(y) \leftarrow u(x, y)$ for all $y = 0, \ldots, H - 1$
    if (image filtering) **then**
      $g^v(y) \leftarrow f^v(y)$ for all y
      compute $w_{y,j}$ using g^v for $j \in N_v(y)$
      build a tridiagonal matrix A_v
      solve $(I_v + \lambda_t A_v)u_v = f_v$
      $u(x, y) \leftarrow u^v(y)$ for all $y = 0, \ldots, H - 1$
    end if
  end for
end for

Algorithm 2 L_0 Gradient Minimization

Input: image I, smoothing weight λ
parameters: β_0, β_{max}, and rate κ
Initialization: $S \leftarrow I$, $\beta \leftarrow \beta_0$, $i \leftarrow 0$
while ($\beta \leq \beta_{max}$) **do**
  With $S^{(i)}$, solve for $h_p^{(i)}$ and $v_p^{(i)}$ in Eq. 2.2.8
  With $h^{(i)}$ and $v^{(i)}$, solve for $S^{(i+1)}$ with Eq. 2.2.6
  $\beta \leftarrow \kappa\beta$
  $i++$
end while

2.3 Preliminary

2.3.1 Matrices

A matrix basically is a collection of numbers ordered by rows and columns. It generally represents a collection of information stored or arranged in an orderly fashion. A general matrix A having m rows and n columns can therefore be written as:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix} \tag{2.3.1}$$

where a_ij are elements of matrix A, for $i = 1, 2, 3, \dots, m$ and $j = 1, 2, 3, \dots, n$.

The elements a_ij are scalars, can either be real or complex numbers. We write $A \in \mathbf{R}^{m \times n}$ if A is an $m \times n$ matrix whose elements are real numbers, and $A \in \mathbf{C}^{m \times n}$ if A is composed of complex number elements [48].

The concept of a matrix can be extended to image processing, as an image is basically a collection of well-ordered rows and columns of numbers representing how light or dark an area of that image is in relation to the rest. Fig. 2.4 illustrates a basic shape, represented as a matrix and as an intelligible image, in order to re-enforce the idea of an image basically being a matrix of ordered entries.

Images can have either two or three dimensional matrix structures(matrix of pixel values) . An example of a two dimensional matrix structural image is a gray-scale image (Fig. 1.2((c))) which is only depicted using one matrix layer, i.e. one value per pixel location. In the case of a color image(Fig. 1.2((a))), it is referred to as a three dimensional matrix structures. This implies that it possess multiple (e.g in case of RGB, 3) matrices layered one onto another. Thus a combination of these pixel values on different layers, aligned in one location produces the illusion of color(illustrated in Fig. 1.1, as opposed to a gray-scale image which is represented by one value per pixel location.

Keeping this concept of an image as a matrix structure in mind, manipulation of images can be approached with greater ease. We shall now present a recap of basic

((a)) Basic single layer image

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 200 & 200 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 200 & 200 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 200 & 200 & 0 \\ 0 & 100 & 100 & 150 & 150 & 200 & 200 & 255 & 255 & 0 \\ 0 & 100 & 100 & 150 & 150 & 200 & 200 & 255 & 255 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

((b)) Matrix representation of (a)

Figure 2.4: Illustration of image in its matrix form

matrix and vector operations and properties, in order to equip the reader with basic knowledge to help in understanding of image smoothing.

Matrix properties

Order of Matrix

The *order* of a matrix A is defined in terms of its number of rows and columns, i.e.

$$A_{order} = No.ofrows \times No.ofcolumns$$

This concept when extended to images is analogous to the resolution of an image, being in terms of number of rows and columns of pixel points. For example, a 1920×1080 pixel resolution image basically means it is represented by that many number of rows and columns. Hence, a high resolution image basically is one represented by a great number of rows and columns of pixel point, with the opposite true for low resolution images.

Transpose

The *transpose* of a matrix A is denoted by a prime as A' or a with a superscript as A^T. By transposing, the first row of a matrix A becomes the first column of the

transpose matrix A^T, the second row becomes the second column, and so on. Thus, a transpose operation is represented in Eq. 2.3.2

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, A^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \tag{2.3.2}$$

Square matrix

A *square* matrix is one that has the same number of rows and columns. A square matrix in which all the elements of the principal diagonal are equal to 1 while all other elements are zero is called a *unit* matrix. A unit matrix is commonly referred to as an *identity* matrix I, given as:

$$I_n = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & \dots & 1 \end{bmatrix} \tag{2.3.3}$$

Inverse

Remember, in scalar algebra, the inverse of a number is that number which, when multiplied with the original number yields a product equal to one. Hence, the inverse of x is simply $1/x$. Or, in a different notation, as x^{-1}. In matrix algebra, the *inverse* of a matrix is that matrix which, when multiplied with the original matrix yields an identity matrix I. Hence,

$$AA^{-1} = A^{-1}A = I \tag{2.3.4}$$

In a general extension, consider two square matrices A and B. If

$$AB = BA = I$$

then B is the inverse of A, and can be denoted as A^{-1}.

A matrix must be square to have an inverse, but not all square matrices have an inverse. The inverse of a matrix exists only if the columns are linearly independent. For covariance and correlation matrices, an inverse will always exist, provided that there are more subjects than there are variables and that every variable has a variance greater than zero [49].

Null matrix
A matrix whose elements are all equal to *zero*,i.e. $a_{ij} = 0 \, \forall \, i, j$, is called *null* matrix.

Diagonal matrix
A *diagonal* matrix is a square matrix in which all the elements except the elements of the principal diagonal are zero. Note that what distinguishes it from an identity matrix is that its diagonal elements are not restricted to a value equal 1.

Symmetry
A *symmetric* matrix is a square matrix in which $a_{ij} = a_{ji} \, \forall \, iandj$. In other terms, a matrix is symmetric if it is equal to its transpose, i.e. $A = A^T$. Eq. 2.3.5 clearly illustrates this, where A is symmetric while B is not.

$$A = \begin{bmatrix} 9 & 1 & 5 \\ 1 & 6 & 2 \\ 5 & 2 & 7 \end{bmatrix}, B = \begin{bmatrix} 9 & 1 & 5 \\ 2 & 6 & 2 \\ 5 & 1 & 7 \end{bmatrix} \tag{2.3.5}$$

Vectors
A matrix having only one row (or column) of elements is refered to as a *row vector*(or *column vector*). Hence, an $m \times n$ matrix can be represented by its row vectors of $1 \times n$ dimension, or by column vectors of $m \times 1$ dimension. Given in Eq. 2.3.6 is a row vector x and column vector y representation of an $m \times n$ matrix.

$$x = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \tag{2.3.6}$$

Dense and Sparse matrices
A matrix, depending on the relative quantity of non-zero elements can be defined as either *sparse* of *dense*. A *dense* matrix is one possessing a significant relative quantity of non-zero elements, while a *sparse* matrix possess relatively few non-zero elements. An example of a dense matrix D and a sparse matrix S is given as;

$$D = \begin{bmatrix} * & * & * & * & * & * \\ * & & * & * & & * \\ * & * & * & * & * & * \\ * & * & & * & * & * \\ & * & * & * & * & \\ * & * & & * & * & * \end{bmatrix}, S = \begin{bmatrix} * & & * & & & * \\ & & & * & & \\ * & * & & & * & \\ & & * & & & * \\ * & * & & & * & \\ & & * & & & * \end{bmatrix} \tag{2.3.7}$$

It can be intrinsically observed that more memory storage is required to store a dense matrix as compared to a sparse matrix.

Orthogonality
Only square matrices may be orthogonal matrices, although not all square matrices are orthogonal matrices. An orthogonal matrix satisfies Eq. 2.3.8

$$AA^T = I \tag{2.3.8}$$

Thus, the inverse of an orthogonal matrix is simply the transpose of that matrix. Matrices of eigenvectors are orthogonal matrices [49].

Positive-definitive
A matrix A is *positive-definite* if, for every nonzero vector x,

$$x^T Ax > 0 \tag{2.3.9}$$

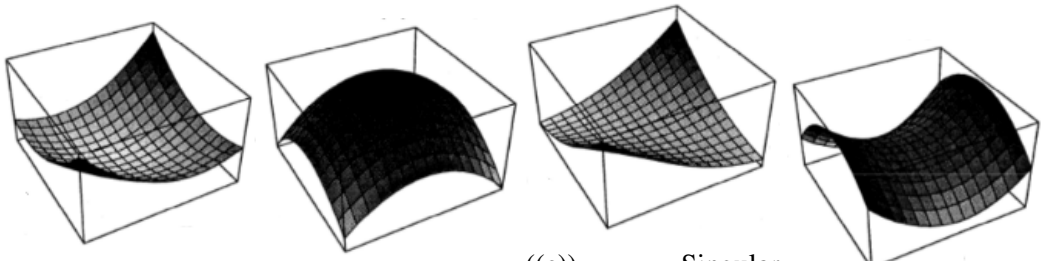
A matrix being positive-definite is a desired quality in minimization problems, as in our case of image smoothing via minimization of a cost function. If a matrix is positive-definite, then a unique minimum point solution exists for its problem. If a matrix is not positive-definite, then it either be *negative-definite*, *singular positive-definite* or *indefinite* matrix. Fig. 2.5 graphically illustrates these definition variations.

Matrix operations

Addition, subtraction and multiplication
Two matrices can be added(subtracted) by adding(subtracting) the corresponding elements of the two matrices, as shown in Eq. 2.3.10

$$\begin{aligned} C &= A \pm B \\ c_{ij} &= a_{ij} \pm b_{ij} \end{aligned} \tag{2.3.10}$$

Note that matrices A, B and C must have the same order. The same holds in the case of subtraction operation. This is shown by:



((a)) Positive-definite matrix ((b)) Negative-definite matrix ((c)) Singular positive-indefinite matrix ((d)) Indefinite matrix

Figure 2.5: Graphic representation of matrix definitiveness

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}, C = \begin{bmatrix} a_{11} \pm b_{11} & a_{12} \pm b_{12} & a_{13} \pm b_{13} \\ a_{21} \pm b_{21} & a_{22} \pm b_{22} & a_{23} \pm b_{23} \\ a_{31} \pm b_{31} & a_{32} \pm b_{32} & a_{33} \pm b_{33} \end{bmatrix}$$

In the case of matrix multiplication, two matrices can be multiplied together provided they are compatible with respect to their orders. The number of columns in the first matrix A must be equal to the number of rows in the second matrix B. That is,

$$\begin{aligned} A_{(n\times p)} \times B_{(m\times q)} &: \textit{ compatible if } p = m. \\ B_{(m\times q)} \times A_{(n\times q)} &: \textit{ compatible if } q = m. \end{aligned} \tag{2.3.11}$$

Hence, the resulting matrix will have the same number of rows as A and the same number of columns as B.

If a matrix is multiplied by a scalar k, each element of the matrix is multiplied by k.

$$kA = \begin{bmatrix} ka_{11} & ka_{12} & ka_{13} \\ ka_{21} & ka_{22} & ka_{23} \\ ka_{31} & ka_{32} & ka_{33} \end{bmatrix} \tag{2.3.12}$$

The following are properties which hold for matrix addition(not including subtraction) and multiplication:

- Matrix addition is commutative,

$$A + B = B + A \tag{2.3.13}$$

- Matrix addition is associative,

$$(A + B) + C = A + (B + C) \tag{2.3.14}$$

- Matrix addition and scalar multiplication are distributive,

$$\begin{aligned} \lambda (A + B) &= \lambda A + \lambda B, \\ (\lambda + \mu) A &= \lambda A + \mu A. \end{aligned} \tag{2.3.15}$$

- Transpose of a sum,

$$(A + B)^T = A^T + B^T \tag{2.3.16}$$

- Transpose of a product,

$$(AB)^T = B^T A^T \tag{2.3.17}$$

As expressed in [48], the seemingly simple properties outlined above are of great value to a programmer. In that, amount of computations can be reduced just by observing them. For example, computing $\lambda A + \lambda B$ requires twice as many operations as opposed to computing $\lambda (A + B)$.

Norms & Inner Products

Vector Norms

Definition 2.2.1. For a vector $x_{n\times 1}$, the *euclidean norm* of x is defined as

$$\begin{aligned} \|x\| &= \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2} = \sqrt{x^T x} \text{ whenever } x \in \mathbf{R}^{n\times 1}, \\ \|x\| &= \left(\sum_{i=1}^{n} \mid x_i \mid^2 \right)^{1/2} = \sqrt{x^* x} \text{ whenever } x \in \mathbf{C}^{n\times 1}. \end{aligned} \tag{2.3.18}$$

For example [50]. if $u = (0; -1; 2; -2; 4)$ and $v = (i; 2; 1 - i; 0; 1 + i)$, then

$$\begin{aligned} \|u\| &= \sqrt{\sum u_i^2} = \sqrt{u^T u} = \sqrt{0 + 1 + 4 + 4 + 16} = 5, \\ \|v\| &= \sqrt{\sum \mid v_i \mid^2} = \sqrt{v^* v} = \sqrt{1 + 4 + 2 + 0 + 2} = 3 \end{aligned}$$

Definition 2.2.2. Standard inner products for $\mathbf{R}^n$ and $\mathbf{C}^n$ are scalar terms defined by,

$$x^T y = \sum_{i=1}^{n} x_i y_i \ \in \mathbf{R}, \text{ and}$$
$$x^* y = \sum_{i=1}^{n} \bar{x}_i y_i \ \in \mathbf{C}, \tag{2.3.19}$$

respectively.

Definition 2.2.3. Given the general *euclidean norm* as $\|.\|$, for $p \geq 1$, the *p-norm* of $x \in \mathbf{C}^n$ is defined as

$$\|x\|_p = \left(\sum_{i=1}^{n} \mid x_i \mid^p \right)^{1/p} \tag{2.3.20}$$

Properties of general vector norms. A norm for a real or complex vector space v is a function $\|.\|$ mapping *V* into **R** that satisfies the following condtions:

$$\begin{aligned} \|x\| \geq 0 \text{ and } \|x\| = 0 &\longleftrightarrow x = 0, \\ \|\alpha x\| = \mid \alpha \mid \|x\| &\text{ for all scalars } \alpha, \\ \|x + y\| &\leq \|x\| + \|y\|, \\ \mid \|x\| - \|y\| \mid &\leq \|x - y\|. \end{aligned} \tag{2.3.21}$$

For detailed proof, refer to [50].

Matrix Norms

The norm of a square matrix A is a non-negative real number denoted $\|A\|$. There are several different ways of defining a matrix norm, but they all share the forllowing properties:

$$\begin{aligned} \|A\| \geq 0 &\text{ for any square matrix } A, \\ \|A\| = 0 &\text{ if and only if the martix } A = 0, \\ \|kA\| = \mid k \mid \|A\|, &\text{ for any scalar } k, \\ \|A + B\| &\leq \|A\| + \|B\|, \\ \|AB\| &\leq \|A\|\|B\|. \end{aligned} \tag{2.3.22}$$

The norm of a matrix is a measure of how large its elements are. It is a way of

determinig the "size" of a matrix that is not necessaryily related to how many rows or columns the matrix has.

Definition 2.2.4. The *1-norm* is given by

$$\|A\|_1 = \max_{1\leq j\leq n}\left(\sum_{i=1}^{n} \mid a_{ij} \mid\right) \tag{2.3.23}$$

which is simply the maximum absolute column sum. Put simply, we can sum the absolute values down each column and then take the biggest answer.

Definition 2.2.5. The ∞*-norm* is given as,

$$\|A\|_\infty = \max_{1\leq i\leq n}\left(\sum_{j=1}^{n} \mid a_{ij} \mid\right) \tag{2.3.24}$$

which is simply the maximum absolute row sum. Put simply, we sum the absolute values along each row and then take the biggest answer.

Definition 2.2.6. The *Euclidean norm* of a matrix A is,

$$\|A\|_E = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(a_{ij})^2} \tag{2.3.25}$$

This is similar to ordinary "Pythagorean" length where the size of a vector is found by taking the square root of the sum of the squares of all the elements. This is the most commonly used norm, and is also refered to as the *2-norm* ($\|.\|_2$). This norm possesse the following specific properties.

$$\|A\|_2 = \max_{\|x\|_2=1}\max_{\|y\|_2=1} \mid y^*Ax \mid . \tag{2.3.26}$$

$$\|A\|_2 = \|A^*\|_2. \tag{2.3.27}$$

$$\|A^*A\|_2 = \|A\|_2^2. \tag{2.3.28}$$

$$\| \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix} \|_2 = \max\{\|A\|_2, \|B\|_2\}. \tag{2.3.29}$$

$$\|U^*AV\|_2 = \|A\|_2 \text{ when } UU^* = I \text{ and } V^*V = I. \tag{2.3.30}$$

Inner Products

Definition 2.2.7. An *inner product* on a real (or complex) vector space is a function that maps each ordered pair of vectors x, y to a real (or complex) scalar $< x|y >$ is given by,

$$x^{'}y = \sum_{j=1}^{p} x_j y_j \tag{2.3.31}$$

such that the following properties hold:

$$\begin{aligned} \langle x|x\rangle \text{ is real with } \langle x|x\rangle = 0 \text{ if } x = 0, \\ \langle x|\alpha y\rangle = \alpha\langle x|y\rangle \text{ for all scalars } \alpha, \\ \langle x|y+z\rangle = \langle x|y\rangle + \langle x|z\rangle, \\ \langle x|y\rangle = \overline{\langle y|x\rangle} (\text{for real spaces, this becomes } \langle x|y\rangle = \langle y|x\rangle). \end{aligned} \tag{2.3.32}$$

Any real or complex vector space that is equipped with an inner product is called an *inner-product space* [50].

Definition 2.2.8. If V is an inner-product space, and if we set $\|.\| = \sqrt{\langle .|. \rangle}$, then

$$|\angle x|y\rangle| \leq \|x\|\|y\| \text{ for all } x, y \in V \tag{2.3.33}$$

Equality holds if and only if $y = \alpha x$ for $\alpha = \langle x|y\rangle / \|x\|^2$.
Hence, a norm on V is defined as

$$\| \star \| = \sqrt{\langle \star|\star \rangle} \tag{2.3.34}$$

By application, the following is realized:

$$\begin{aligned} \|x+y\|^2 &= \langle x+y|x+y\rangle \\ &= \langle x|x\rangle + \langle x|y\rangle + \langle y|x\rangle + \langle y|y\rangle \\ &\leq \|x\|^2 + 2\,\|\langle x|y\rangle\| + \|y\|^2 \\ &\leq (\|x\| + \|y\|)^2 \end{aligned} \tag{2.3.35}$$

Definition 2.2.9. For a given norm $\|x\|$ on a vector space V, there exists an inner product on V such that $\langle x|x\rangle = \|x\|^2$ if and only if the *parallelogram identity*

$$\|x+y\|^2 + \|x-y\|^2 = 2\left(\|x\|^2 + \|y\|^2\right) \tag{2.3.36}$$

holds for all $x, y \in V$.

2.3.2 Iterative Methods

In many large-scale scientific simulation codes, most computation is spent on solving linear equations with sparse coefficient matrices. For this reason, much of the scalable algorithm research and development is aimed at solving these large, sparse linear systems of equations on parallel computers. Sparse linear solvers can be broadly classified as being either direct or iterative. Among the sparse linear solvers, iterative methods are memory scalable and the only choice for large-scale simulations by massively parallel computers. The rate of convergence of iterative methods depends strongly on the spectrum of the coefficient matrix [51].

Direct solvers such as Gaussian Elimination are based on a factorization of the associated sparse matrix. They are extremely robust and would give the exact solution of $Ax = b$ after a finite number of steps without round-off errors. However, their memory requirements grow as a nonlinear function of the matrix size , hence not making them suitable for very large sparse linear problems [51].

In contrast, iterative methods are memory scalable. Iterative methods hence can be effectively implemented on parallel processors. Iterative methods are generally classified as being stationary or non-stationary. Stationary methods are such as the Jacobi method, the Gauss-Seidel method and SOR (Successive Over-relaxation) method. Non-stationary methods are the Krylov subspace methods such as the Conjugate Gradient (CG) method, the Bi-Conjugate Gradient Stabilized (BiCGSTAB) method and the Generalized Minimal Residual (GMRES) method [51].
Stationary iterative methods are iterative methods that can be expressed in a simple form:

$$x^k = Bx^{k-1} + c \tag{2.3.37}$$

where neither B nor c depend upon the iteration count k.

However in the case of non-stationary methods, the computations involve information that changes at each iteration. Typically, constants are computed by taking inner products of residuals or other vectors arising from the iterative method [52].

We shall henceforth briefly look at some of the iterative methods, which are relevant to realizing our goal of performing image smoothing using an iterative method, thereby also comparing its performance to that implemented using FFT. We shall start also by introducing some of the basic, most common iterative methods, which may not be optimized for our goal but provide a good understanding foundation.

Jacobi, Gauss-Seidel & SOR

Jacobi Method

The Jacobi method is based on solving for every variable locally with respect to the other variables; one iteration of the method corresponds to solving for every variable once.
Given an $n \times n$ real matrix A and a real *n-column vector* b, the task is to solve for unknown x in a linear system given by Eq. 2.3.38

$$Ax = b \tag{2.3.38}$$

For simpler understanding of Jacobi method, we shall express it in its vector form as

$$x_{k+1} = D^{-1}\left(E + F\right)x_k + D^{-1}b \tag{2.3.39}$$

where the matrix D is the diagonal matrix of A, E and F are the strict lower and strict upper triangular matrices of A respectively. It is always assumed that the diagonal entries of A are all *nonzero* [53].
In Jacobi method, the $i-th$ equation is independent of others, hence it is also known as the *method of simultaneous displacements*, since the updates could in principle be done simultaneously [52]. The pseudocode is given in Alg. 3: This method is

Algorithm 3 Jacobi Method

Choose an initial guess $x^{(0)}$ to the solution x.
for $k = 1, 2, \ldots$
  for $i = 1, 2, \ldots, n$
    $x_i = 0$
    for $j = 1, 2, \ldots, i-1, i+1, \ldots, n$
      $x_i = x_i + a_{i,j}x_j^{k-1}$
    end for
    $x_i = \left(b_i - x_i\right)/a_{i,i}$
  end for
  $x^k = x$
  check convergence; continue if necessary
end for

easy to understand and implement, but convergence is slow [52].

Gauss-Seidel Method

Gauss-Seidel method is similar to Jacobi method, except that it uses updated values as soon as they are available. In general, if the Jacobi method converges, the Gauss-Seidel method will converge faster than Jacobi method, though still relatively slowly [52].

Gauss-Seidel's computation is a serial computation, as each component of the new iterate depends upon all previously computed components. The update cannot be done simultaneously as in the Jacobi method.
In matrix terms, the definition of Gauss-Seidel method can be expressed as:

$$x_{k+1} = (D - E)^{-1} F x_k + (D - E)^{-1} b \tag{2.3.40}$$

where the matrix D is the diagonal matrix of A, E and F are the strict lower and strict upper triangular matrices of A, respectively.
Eq. 2.3.40 represents forward Gauss-Seidel. In backward Gauss-Seidel, we just replace $D - E$ wiht $D - F$ in the equation. The pseudocode is given in Alg. 4.

Algorithm 4 Gauss-Seidel Method

```
Choose an initial guess x^(0) to the solution x.
for k = 1, 2, ...
  for i = 1, 2, ..., n
    σ = 0
    for j = 1, 2, ..., i − 1
      σ = σ + a_{i,j} x_j^k
    end for
    for j = i + 1, ..., n
      σ = σ + a_{i,j} x_j^{k−1}
    end for
    x_i^k = (b_i − σ) / a_{i,i}
  end for
  check convergence; continue if necessary
end for
```

Successive Over-relaxation (SOR)

This method can be derived from the Gauss-Seidel method by introducing an extrapolation parameter ω. For the optimal choice of ω, SOR may converge faster than Gauss-Seidel by an order of magnitude [52].

The extrapolation term ω takes the form of a weighted average between the previous iterate and the computed Gauss-Seidel iterate successively for each component. In matrix terms, the SOR can be expressed as:

$$x_k = (D - \omega E)^{-1} (\omega F + (1 - \omega) D) x_{k-1} + \omega (D - \omega E)^{-1} b \tag{2.3.41}$$

The pseudocode is given in Alg. 5.

Algorithm 5 SOR method

Choose an initial guess $x^{(0)}$ to the solution x.
for $k = 1, 2, \ldots$
  for $i = 1, 2, \ldots, n$
    $\sigma = 0$
    for $j = i + 1, \ldots, i - 1$
      $\sigma = \sigma + a_{i,j} x_j^k$
    end for
    for $j = i + 1, \ldots, n$
      $\sigma = \sigma + a_{i,j} x_j^{k-1}$
    end for
    $\sigma = (b_i - \sigma) / a_{i,i}$
    $x_i^k = x_i^{k-1} + \omega \left(\sigma - x_i^{k-1} \right)$
  end for
  check convergence; continue if necessary
end for

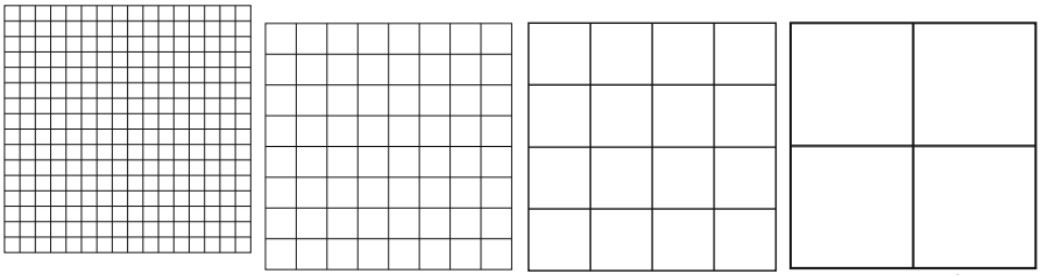
Multigrid Method

Multigrid methods are called scalable or optimal because they can solve a linear system with N unknowns with only $\mathcal{O}(N)$ work. Since this work can be effectively distributed across a parallel machine, multigrid methods are able to solve ever larger problems on proportionally larger parallel computers in essentially constant time, making them an ideal solver for large-scale scientific simulation [54].

Simple iterative methods such as Jacobi and Gauss-Seidel have $\mathcal{O}(N)$ cost per iteration. The number of iterations depends on the condition number of matrix A and on the desired level of accuracy [55]. Hence, as in the case of smoothing problems, in which the condition number can be very large, henceforth requiring far too many iterations to be practical. However, iterative methods such as Jacobi and Gauss-Seidel, when used as smoothers, are an inexpensive way to discover

high-frequency components of the correction in very few iterations. Hence they are used in multigrid methods [55].

Multigrid method is basically a divide-and-conquer algorithm for solving the discrete Poisson equation. It is widely used on other similar ("elliptic") partial differential equations as well. It uses divide-and-conquer in two related senses. First, it obtains an initial solution for an $N \times N$ grid by using $(N/2) \times (N/2)$ grid as an approximation, taking every other grid point from the $N \times N$ grid. The coarse $(N/2) \times (N/2)$ grid is in turn approximated by an $(N/4) \times (N/4)$ grid, and so on recursively. Fig. 2.6 illustrates the concept of multi grids, which will help to better understand it. Hence, obtaining a solution on coarsest grid(Fig. 2.6((d))), which has



((a)) Fine grid($N \times N$) ((b)) $\frac{N}{2} \times \frac{N}{2}$ grid ((c)) $\frac{N}{4} \times \frac{N}{4}$ grid ((d)) $\frac{N}{6} \times \frac{N}{6}$ grid

Figure 2.6: Illustration of multigrid method

fewer unknowns, and using it to approximate the solution of a fine grid(Fig. 2.6((a))) is less computational costly. The second way multigrid uses divide-and-conquer is in *frequency domain*. Considering the *error* in the iterate solution of a general iterative method such as Jacobi and Gauss-Seidel as sinusoidal components, high frequency errors being refered to as *rough error*, while low frequency errors as *smooth error*. According to [56], iterative methods such as Jacobi and Gauss-Seidel nearly remove high frequency errors in a few iterations. However, their slow convergence rate is due to their inability to eliminate smooth errors quickly. Multigrid method addresses this by transferring fine grids to coarse grids, on which the *smooth error* becomes *rough* and thereby removed quickly.

Therefore, what multigrid does on the finest grid $P_{(m)}$(Fig. 2.6((a))), is to damp down the upper half of the frequency components (high frequency) of the error in the solution. On the next coarser grid $P_{(m-1)}$(Fig. 2.6((b))) with half as many points, multigrid damps the upper half of the remaining frequency components in the error. On the next coarser grid $P_{(m-2)}$(Fig. 2.6((c))), the upper half of the remaining frequency components are damped, and so on, until we solve the exact (1 unknown) problem $P_{(1)}$(Fig. 2.6((d))). Fig. 2.7 illustrates the process by which the error com-

ponent is dampened by grid manipulation. This recursive application yields a multigrid *V–cycle*. In the *V-cycle* shown in Fig. 2.8((a)), starting with the finest grid, all subsequent coarser grids are visited only once; in the down-cycle, smoothers damp oscillatory error components at different grid scales; in the up-cycle, the smooth error components remaining on each grid level are corrected using the error approximations on the coarser grids. However, in a *W-cycle* shown in Fig. 2.8((b)), the coarser grids are solved more rigorously in order to reduce residuals as much as possible before going back to the more expensive finer grids.

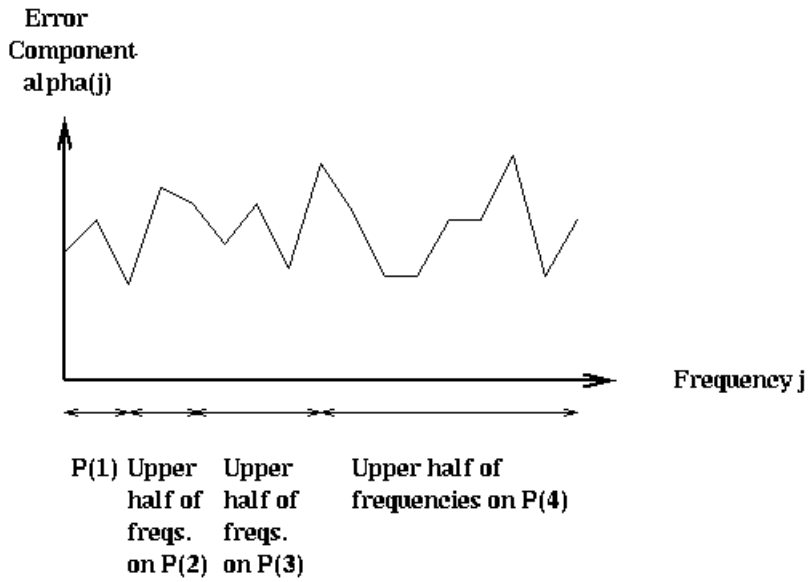
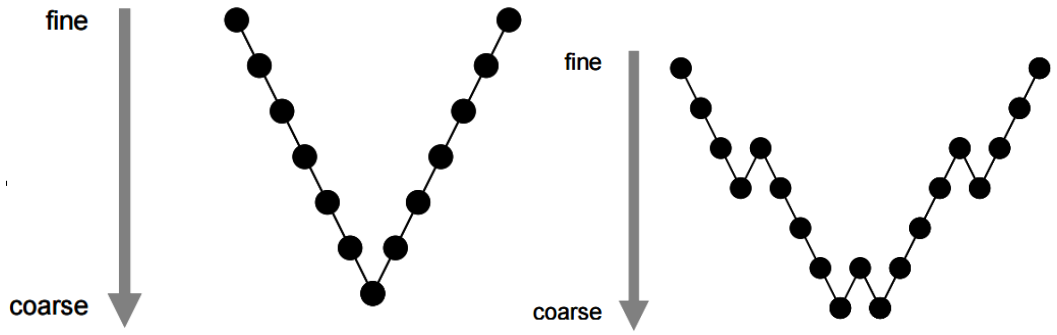


Figure 2.7: Multigrid error elimination, per grid change



((a)) V-Cycle ((b)) W-Cycle

Figure 2.8: Multigrid cycles

Therefore, the multigrid approach was developed in recognition that low-frequency error can be accurately and efficiently removed on a coarser grid. Multigrid method uniformly damps all frequencies of error components with a computational cost that depends only linearly on the problem size, with N unknowns $\mathcal{O}(N)$ [51].

To explain the concept of multigrid, we shall use the simplest form which is a two-grid method, which exploits only two grids: a fine grid and a coarse grid. The two-grid method consists of a *smoothing* step and a *coarse-grid correction* step [57]. Given a linear system problem in Eq. 2.3.42, on a fine grid:

$$A_f x_f = b_f \tag{2.3.42}$$

where the subscript f represents *fine* grid. To better understand the operations of multigrid method on a given linear problem, we shall first state the core operators involved.

- The *restriction* operator; this operator transfers the residual containing low-frequency (or smooth) components from a fine grid onto a coarse grid. Hence, the problem is restricted to a simpler problem, in essence. According to [58], an example of restriction can be symbolized by the following stencils:

$$\begin{bmatrix} 1/4 & 1/2 & 1/2 \\ 1/2 & 1 & 1/2 \\ 1/4 & 1/2 & 1/4 \end{bmatrix}$$

- The *interpolation* operator; this operator transfers the solution obtained on a coarse grid onto a fine grid. This approximation of the solution onto a fine grid has the low-frequency components taken care of on the coarse grid, and thereby acts as a correction to the fine solution (coarse grid correction). Inherently by its name, it prolongates the small scale solution onto a large scale grid. Taking from [58], an example of interpolation (or prolongation) is a standard transfer operation interpolated bi-linearly. This can be symbolized as:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- The *smoother* operator; this relaxes the problem on the fine grid, operating on the initial solution and thereby computing an improved solution. This relaxation is basically the removal of high frequency (or rough) error from the initial solution. A example of a smoothing operator is Jacobi or Gauss-Seidel method.

Algorithm 6 Two-grid Multigrid Method

while (!convergence)
  $x_f = pre\_smoother\,(A_f, b_f, x_f)$; //pre-smoothing
  $r_c = restrict\,(b_f - A_f x_f)$; //coarse-grid correction
  Solve $A_c d_c = r_c$;
  $x_f = x_f + prolongate\,(d_c)$;
  $x_f = post\_smoother\,(A_f, b_f, x_f)$; //post-smoothing
end while

The pseudocode of two-grid multigrid is given in Alg. 6. From this, a simple coarse grid correction can henceforth be given by Eq. 2.3.43:

$$x_f^{(k+1)} = x_f^{(k)} + R^T A_c^{-1} R \left(b - A_f x_f^{(k)} \right) \tag{2.3.43}$$

where the subscript c and f refers to *coarse grid* and *fine grid* respectively, while R^T and R represent the *interpolate/prolongate* and *restrict* operator respectively [51]. Fig. 2.9 gives a graphic flow chart summarizing multigrid method operations.

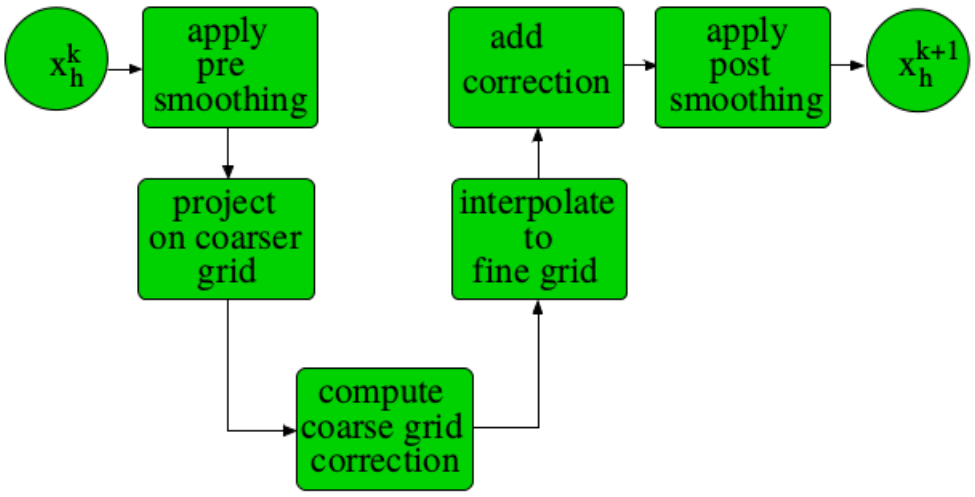


Figure 2.9: Multigrid method flow chart

Multigrid methods can be categorized into two main groups, depending on the method used to construct the coarse grids. These are *geometric* and *algebraic* multigrid.

Geometric Multigrid

According to findings in [59], geometric multigrid methods were originally designed for solving elliptic PDEs, of the form represented in Eq. 2.3.44, by discretiz-

ing them using a hierarchy of regular grids of varying degrees of fineness over the same domain. This hierarchy can be cumbersome to construct [60].

$$-\frac{\partial}{\partial x}\left(v\frac{\partial u}{\partial x}\right)-\frac{\partial}{\partial y}\left(v\frac{\partial u}{\partial y}\right)=f \tag{2.3.44}$$

In geometric multigrid, the geometry of the problem is used to define the various multigrid components. According to [61], geometric multigrid refers to a multigrid method that uses fixed full-octave coarsening along with fixed non-adaptive (uniform) bilinear interpolation.

Algebraic Multigrid

Algebraic multigrid(AMG) method is a generalization of the hierarchical approach of the geometric multigrid method so that it is not dependent on the availability of the meshes used for discretizing a PDE, but can be used as a black-box solver for a given linear system of equations [59]. In [62], it is conferred that in contrast to geometrically based multigrid, algebric multigrid does not require a given problem to be defined on a grid but rather operates directly on (linear sparse) algebraic equations of the form given in Eq. 2.3.38:

$$Ax = b$$

To easily understand, terms such as grids basically refer to simply a set of variables, while coarse-grid discretization corresponds to certain matrix equations of reduced dimensions [62]. According to [51], AMG is suitable for applications with unstructured grids, which can be extended to image smoothing minimization. AMG preconditioner can therefore be derived explicitly from the coefficient matrix A of the system $Ax = b$ [59].

AMG is easier to apply to a generally wide range of linear sparse problem, however the down-side is that efficient parallelization of AMG is much harder than that of geometric multigrid [59].

Conjugate Gradient

Conjugatet gradient (CG) method is the most popular iterative method for solving large systems of linear equations. CG is effective for systems of the form given in Eq. 2.3.38, shown below.

$$Ax = b$$

where x is the unknown vector, b is a known vector, and A is a known sparse(Eq. 2.3.7), symmetric($A = A^T$), positive-definite matrix(Eq. 2.3.9,Fig. 2.5((a))). These systems arise in many important settings, such as finite difference and finite element

methods for solving partial differential equations, structural analysis, circuit analysis, and math homework [63]. Henceforth, we investigate its application in image processing. It was first introduced by M.R. Hestenes & E. Stiefel in 1952, and has since gained popularity. CG method is a Krylov subspace iterative method.

The conjugate gradient method derives its name from the fact that it generates a sequence of conjugate (or orthogonal) vectors. These vectors are the residuals of the iterates. They are also the gradients of a quadratic functional, the minimization of which is equivalent to solving the linear system. CG is an extremely effective method when the coefficient matrix is symmetric positive definite, since storage for only a limited number of vectors is required [52].

Basically, the fundamental underlying structure of conjugate gradient method is:

1. Start with an initial point,
2. determine direction of movement from that point, in step towards solution,
3. move in that direction by desired stepsize to a relative minimum of the objective function, and
4. at the new point, a new direction is determined and the process is repeated, until desired solution is arrived at.

To better understand CG, let us begin with some definitions/ reviews. A is an $n \times n$ matrix, and x and b are row column vectors(i.e., $n \times 1$ vectors), written out fully, Eq. 2.3.45.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{2.3.45}$$

Note: For matrix properties and manipulate used to formulate and implement conjugate gradient method, refer to Sec. 2.3.1.

In conjugate gradient method, it is important for matrix A to be positive-definite (Eq. 2.3.9) as this will ensure a unique solution to the minimization problem, as illustrated graphically in Fig. 2.5((a)), of which image smoothing is a minimization problem.

A minimization problem presented in a problem $Ax = b$ can be likened to a solution of a quadratic problem(solution being at *zero* gradient point). A quadratic form is simply a scalar, quadratic function of a vector with the form given in Eq. 2.3.46.

$$f(x) = \frac{1}{2} x^T A x - b^T x + c \tag{2.3.46}$$

where c is a constant. Solving a problem of the form $Ax = b$ is equivalent to minimizing the quadratic function given in Eq. 2.3.46. In order to find the solution to Eq. 2.3.46, which is a minimum point in the case of minimization, we formulate its derivative as given in Eq. 2.3.47.

$$f^{'}(x) = \frac{1}{2}A^{T}x + \frac{1}{2}Ax - b \tag{2.3.47}$$

Given that A is symmetric, i.e. $A^{T} = A$, this equation reduces to

$$f^{'}(x) = Ax - b$$

Setting the gradient to zero, we obtain Eq. 2.3.38, the linear system we wish to solve [63]. From this formulation above, it is evident that symmetry and positive-definitiveness are very critical to obtaining a solution to our problem.

Conjugate gradient method proceeds by generating vector sequences of *iterates* (i.e., successive approximations to the solution), residuals corresponding to the iterates, and search directions used in updating the iterates and residuals. Although the length of these sequences can become large, only a small number of vectors needs to be kept in memory. In every iteration of the method, two inner products are performed in order to compute update scalars that are defined to make sequence satisfy certain orthogonality conditions [51]. In a nutshell, by simply starting at an initial guess of solution of x, and moving by step-size in a search direction constructed by conjugation of the residual ($r_{(k)} = b–Ax_{(k)}$), a solution is reached.

The iterates $x^{(k)}$, which are the solutions of x at the $k-th$ iteration, are updated in each iteration by a step-size multiple($\alpha_{(k)}$) in the search direction vector $d^{(k)}$:

$$x_{(k)} = x_{(k-1)} + \alpha_{k}d_{(k)} \tag{2.3.48}$$

correspondingly the residuals $r_{(k)} = b - Ax_{(k)}$ are updated as

$$r_{(k)} = r_{(k-1)} - \alpha_{k}q_{(k)} \tag{2.3.49}$$

where,

$$q_{(k)} = Ad_{(k)}$$

The search directions are updated using the residuals, yielding

$$d_{(k)} = r_{(k-1)} + \beta_{k}d_{(k-1)} \tag{2.3.50}$$

where the choice

$$\beta_{k} = \left(r_{(k)}\right)^{T}r_{(k)}\big) \diagup \left(r_{(k-1)}^{T}r_{(k-1)}\right) \tag{2.3.51}$$

ensures that $d_{(k)}$ and $Ad_{(k-1)}$(or equivalently, $r_{(k)}$ and $r_{(k-1)}$) are orthogonal. The choice of,

$$\alpha_k = \left(r_{(k)}^T r_{(k)}\right) \diagup \left(d_{(k)}^T q_{(k)}\right) \tag{2.3.52}$$

minimizes

$$r_{(k)}^T A^{-1} r_{(k)}$$

The pseudocode of conjuagate gradient method is given in Alg. 7.

Algorithm 7 Conjugate Gradient Method

$[x] = CG\,(A, b, x_0)$
$x_0 = 0$; # Or initial guess value
$r_0 = b - Ax_0$; # initial residual
$d_0 = r_0$; # initial search direction
for $k = 1, 2, \ldots$
6: $\alpha_{(k)} = \frac{r_{(k-1)}^T r_{(k-1)}}{d_{(k-1)}^T A d_{(k-1)}}$; #step length
$x_{(k)} = x_{(k-1)} + \alpha_{(k)} d_{(k-1)}$; # approximate solution
$r_{(k)} = r_{(k-1)} - \alpha_{(k)} A d_{(k-1)}$; # residual
check convergence; break if converged
$\beta_{(k)} = \frac{r_{(k)}^T r_{(k)}}{r_{(k-1)}^T r_{(k-1)}}$; #improvement
$d_{(k)} = r_{(k)} + \beta_{(k)} d_{(k-1)}$ # search direction
12: **end for**

Preconditioned Conjugate Gradient

The robustness and the speed of Krylov subspace iterative methods is improved, often dramatically, by preconditioning [59]. Hence, we can greatly improve the performance of CG method by introducing a preconditioner M into the problem $Ax = b$. For mathematical details on preconditioning a general problem $Ax = b$, refer to Chapter 2.3.2.

By this modification, the resulting preconditioned conjugate gradient(PCG) psuedocode is given in Alg. 8. The convergence rate and robustness of the PCG largely depends on how well the preconditioner approximates A.

Iterative methods for solving sparse systems of linear equations are potentially less memory and computation intensive than direct methods, but often experience slow convergence or fail to converge at all.The rate of convergence of iterative methods depends strongly on the spectrum of the coefficient matrix A [51]. Therefore,

Algorithm 8 Preconditioned conjugate gradient method

$[x] = PCG\,(A, b, x_0)$
x_0
$r_0 = b - Ax_0$
$d_0 = Mr_0$
for $k = 1, 2, \ldots$
  $\alpha_k = \frac{\left(d^T_{(k-1)} r_{(k-1)}\right)}{d^T_{(k-1)} A d_{(k-1)}}$
  $x_{(k)} = x_{(k-1)} + \alpha_{(k)} d_{(k-1)}$
  $r_{(k)} = b - Ax_{(k)} = r_{(k-1)} - \alpha_{(k)} A d_{(k-1)}$
  $\tilde{r}_{(k)} = Mr_{(k)}$ //Preconditioning
  check for convergence; continue if necessary
  $\beta_{(k)} = \frac{\left(\tilde{r}^T_{(k)} A d_{(k-1)}\right)}{\left(d^T_{(k-1)} A d_{(k-1)}\right)}$
  $d_{(k)} = \tilde{r}_{(k)} - \beta_{(k)} d_{(k-1)}$
end for

convergence rate is dependent greatly on the condition number of matrix A. Condition number is given by Eq. 2.3.53

$$\kappa\,(A) = \|A^{-1}\| \cdot \|A\| \tag{2.3.53}$$

If A is normal then,

$$\kappa\,(A) = \mid \frac{\lambda_{max}\,(A)}{\lambda_{min}\,(A)} \mid \tag{2.3.54}$$

where $\lambda_{max}\,(A)$ and $\lambda_{min}\,(A)$ are maximal and minimal *eigenvalues* of A respectively.

A low condition number signifies a well-conditioned problem, resulting in good convergence rate, while a high condition number signifies an ill-conditioned problem resulting in poor convergence rate.

Preconditioning is a technique for improving the condition number of a coefficient matrix. The use of a good preconditioner improves the convergence of iterative methods, sufficiently to overcome the extra cost of constructing and applying the preconditioner. Indeed, without a preconditioner the iterative method may even fail to converge [51].
Suppose a preconditioner M which is a symmetric and positive-definite matrix that approximates A, but is easier to invert compared to A. We can now solve $Ax = b$ indirectly by solving Eq. 2.3.55

$$M^{-1}Ax = M^{-1}b \tag{2.3.55}$$

If $\kappa(M^{-1}A) << \kappa(A)$, or if the eigenvalues of M^{-1} are better clustered than those of A, we can iteratively solve Eq. 2.3.55 quicker than the original problem in Eq. 2.3.38 [63].

Practical requirements for successful preconditioning are that the cost of computing M itself must be low and the memory required to compute and apply M must be significantly less than that for solving $Ax = b$ via direct factorization [59]. Hence, selecting a well suited preconditioner is the key to achieving better performance.

Preconditioners can be roughly classified into six groups, according to [59]. We discuss this preconditioner groups in the following sections.

Split-based

Preconditioners based on splitting of the matrix A are basically simple preconditioners based on stationary iterative methods. These are general preconditioners namely, the Jacobi, Gauss-Seidel and SOR. The Jacobi preconditioner is the simplest of them all, as it is a diagonal matrix whose diagonal entries are identical to those of matrix A. Refer to Sec. 2.3.2 for the introduction to these methods as standalone iterative methods. Here we describe them as preconditioners.

Given coefficient matrix A, let us split it into a diagonal matrix D, lower and upper triangular matrices $-E$ and $-F$ respectively, shown in Eq. 2.3.56

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \qquad D = \begin{bmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{nn} \end{bmatrix} \tag{2.3.56}$$

$$E = -\begin{bmatrix} 0 & & & \\ a_{21} & 0 & & \\ \vdots & \vdots & \ddots & \\ a_{n1} & a_{n2} & \dots & 0 \end{bmatrix} \qquad F = -\begin{bmatrix} 0 & a_{12} & \dots & a_{1n} \\ & 0 & \dots & a_{2n} \\ & & \ddots & \vdots \\ & & & 0 \end{bmatrix}$$

Hence, Jacobi method, Gauss-Seidel and SOR are formulated as preconditioning matrices M_{JA}, M_{GS} and M_{SOR} in Eqs. 2.3.57, 2.3.58, and 2.3.59, respectively.

$$M_{JA} = D, \tag{2.3.57}$$

$$M_{GS} = (D + E) D^{-1} (D + F), \tag{2.3.58}$$

$$M_{SOR} = \left(\frac{D}{\omega} + E\right) \frac{\omega D^{-1}}{2 - \omega} \left(\frac{D}{\omega} + F\right), \text{ where } 0 < \omega < 2. \tag{2.3.59}$$

Incomplete factorization

Preconditioners based on Incomplete factorization. This is based on factorization along the lines of a regular Gaussian elimination or *Cholesky* factorization. As opposed to complete factorization methods such as $A = LU$, in which the resulting triangular matrices are much denser than A, and therefore too expensive to compute and store, incomplete factorization drops most of nonzero entries from the triangular factors. This makes such preconditioners conceptually simple but highly effective. Incomplete factorization methods can be broadly categorized depending on the selective nonzero element dropping criteria, of which namely are static-pattern, Threshold-based, and inverse-norm estimate based incomplete factorization [59].

According to [64], IC (incomplete Cholesky) factorization is an important preconditioner in iterative methods involving sparse positive definite(SPDs). IC factorization is the simplest form, belonging to static-pattern incomplete factorization category [59]. IC can be modified by applying strategies, which use information about the values of dropped elements and modify the process to assure the existence of the factorization [65], for example, a strategy which uses only information about the nonzero structure of the matrix to be factored and the structure of the target sparsity pattern for the IC factor [66]. More variants of modified incomplete Cholesky factorization is given in [67]. The Incomplete Cholesky factorization of a positive definite matrix A is $M = LL^T$ where L is a modified lower triangular matrix of A, which is somehow close to the lower triangular matrix of A, but is as sparse as A. Applying the preconditioner M in split form i.e., in terms of L ensures preservation of SPD properties in the resulting problem [68]. One popular way to find incomplete Cholesky factorization matrix is to use the algorithm for finding the exact Cholesky decomposition, except that any entry is set to zero if the corresponding entry in A is also zero. Incomplete Cholesky factorization L of A of size $n \times n$ can be realized using the Alg. 9 Applying the incomplete Cholesky factorization as

Algorithm 9 Incomplete Cholesky

for $i = 1, 2, \ldots, n$
  $L_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} L_{ik}^2\right)^{\frac{1}{2}}$
  for $j = i + 1, \ldots, n$
    $L_{ji} = \frac{1}{L_{ii}\left(a_{ij} - \sum_{k=1}^{i-1} L_{ik} L_{jk}\right)}$
  end for
end for

a preconditioner $M = LL^T$ to the problem in Eq. 2.3.55, yields

$$L^{-1} A L^{-T} u = L^{-1} b, \text{ where } x = L^{-T} u \tag{2.3.60}$$

IC factorization can be applied to preconditioned conjugate gradient as a preconditioner. This method is refered to as ICCG. The pseudocode is given in Alg. 10.

Algorithm 10 Incomplete Cholesky Conjugate Gradient

First compute L, as the preconditioner.
$r_0 = b - Ax_0$
$\tilde{r} = L^{-1}r_0$
for $k = 1, 2, \ldots$
  $z = L^{-1}AL^{-T}\tilde{r}$
  $v = \frac{\left(r_{(k-1)}, r_{(k-1)}\right)}{(\tilde{r}, z)}$
  $x_{(k)} = x_{(k-1)} + v\tilde{r}$
  $r_{(k)} = r_{(k-1)} - vz$
  $\mu = \frac{(r_k, r_k)}{\left(r_{(k-1)}, r_{(k-1)}\right)}$
  $\tilde{r} = r_{(k)} + \mu\tilde{r}$
  if converged **then**
    $x_{(k)} = L^{-T}x_{(k)}$
    return $x_{(k)}$
  end if
end for

Multigrid

According to [51], Multigrid methods tend to be problem-specific solutions and less robust than preconditioned Krylov iterative methods such as CG, IC/ILU iterative methods. Fortunately, it is easy to combine the best features of multigrid and Krylov iterative methods into one algorithm, i.e., multigrid-preconditioned Krylov iterative methods. It has been proven in [51] that the resulting algorithm is robust, efficient and scalable. Hence, such is intrinsically suitable for parallel computing. In our case, we seek to implement it with preconditioned Conjugate gradient method. This combination is achieved by replacing the preconditioning step of the Krylov subspace solver (in our case CG solver) with one iteration of the multigrid algorithm [59]. This is referred to as MCCG method [69].

According to [59], multigrid method can be implemented as a preconditioner in preconditioned conjugate gradient method by replacing the preconditioning step with one iteration of multigrid algorithm, instead of repeating the multigrid cycle k times to solve the problem; treating the approximate solution obtained by an iteration of the multigrid algorithm as the solution with respect to a hypothetical precon-

ditioner matrix. The inclussion of multigrid in PCG method is shown in Alg. 11, collectively known as MGCG.

Algorithm 11 Multigrid Conjugate Gradient

$[x] = MGCG\,(A, b, x_0)$
x_0
$r_0 = b - Ax_0$
$d_0 = r_0$
for $k = 1, 2, \ldots$
  $\alpha_k = \frac{\left(d^T_{(k-1)} r_{(k-1)}\right)}{d^T_{(k-1)} A d_{(k-1)}}$
  $x_{(k)} = x_{(k-1)} + \alpha_{(k)} d_{(k-1)}$
  $r_{(k)} = b - Ax_{(k)} = r_{(k-1)} - \alpha_{(k)} A d_{(k-1)}$
  Relax $\tilde{r}_{(k)} = M r_{(k)}$ using the Multigrid method
  check convergence; continue if necessary
  $\beta_{(k)} = \frac{\left(\tilde{r}^T_{(k)} A d_{(k-1)}\right)}{\left(d^T_{(k-1)} A d_{(k-1)}\right)}$
  $d_{(k)} = \tilde{r}_{(k)} - \beta_{(k)} d_{(k-1)}$
end for

Sparse Approximate Inverse

As opposed to incomplete factorization preconditioners which seek to compute the sparse approximation of the triangular matrices L and U(which constitute of M) of the coefficient matrix A, sparse approximate preconditioners seek to directly compute M^{-1} as an approximation to its inverse A^{-1}. According to [59], there are some important advantages of explicity using an approximation of A^{-1} for preconditioning, rather than using A's approximate factors. These are; avoiding of breakdown in computation which may occur in the case of incomplete factorization preconditioners, in an event of matrix A has small or zero diagonal elements; application of sparse matrix M^{-1} maybe simpler and more easily parallelizable than the forward and back susbstitutions with $\tilde{L}$ and $\tilde{U}$. However, approximating the inverse of A in somecases is difficult, let alone impossible to compute or store as it is most likely dense.

Stochastic

This class of preconditioners involve approximating the solution of linear systems based on random sampling of the coefficient matrix. This class is yet to be ex-

tensively explorered by researchers, as it has somewhat limited classes of linear systems on which it is applicable. However, it has an attractive property of these methods is that they are usually trivially parallelizable [59].

Matrix-Free Methods and Physics-Based

This class of preconditioners is different from those discussed previously, as it does not explicitly depend on the availability of matrix A. In this case, a coefficient matrix-free method, preconditioning is applied implicitly, or the knowledge of the physics of the application is utilized to construct the preconditioner [59]. These preconditioners are of a highly application-specific nature, henceforth are not covered in detail.

2.4 Proposed Algorithm

2.4.1 Problem formulation

Remember from Sec. 2.2 that an image smoothing problem can be represented by Eq. 2.2.9 can be split by using half-quadratic form into Eq. 2.2.11,recapped below.

$$\begin{aligned} v^{(k+1)} &\leftarrow \underset{v}{\operatorname{argmin}}\, \psi(v) + \frac{\beta}{2} \left\| \nabla u^{(k)} - v \right\|_2^2, \\ u^{(k+1)} &\leftarrow \underset{u}{\operatorname{argmin}}\, \lambda \left\| u - g \right\|_2^2 + \beta \left\| \nabla u - v^{(k+1)} \right\|_2^2, \end{aligned}$$

Note that we solve for the shrinkage operation $v^{(k+1)}$ by Eq. 2.2.12 in Sec. 2.2. The part of the problem which we wish to solve using a spatial iterative solver is the screened Poisson equation corresponding to solving for $u^{(k+1)}$, which was originally solved using an FFT solver, Eq. 2.2.15 [1].

According to [45], the screened Poisson equation corresponding to solving for $u^{(k+1)}$ comprises of two components; $\|u–g\|_2^2$ and $\|\nabla u–v^{(k+1)}\|_2^2$ which represent the data term and gradient term respectively. The data term ensures the closeness of the term u (smoothed output) to the term g (original input). While the gradient term ensures the closeness of the gradient of u to a given gradient field $v^{(k+1)}$, which in our case corresponds to a shrinkage operation performed on u according to $v^{(k+1)}$. As introduced in Sec. 2.3.1, remember that we can redefine the terms in $u^{(k+1)}$ by euclidean norm, yielding

$$\left\| u - g \right\|_2^2 = \left(\sqrt{\sum_i \left(u_i - g_i \right)^2} \right)^2 = \sum_i \left(u_i - g_i \right)^2, \quad (2.4.1)$$

$$\left\| \nabla u - v^{(k+1)} \right\|_2^2 = \left(\sqrt{\sum_i \left(\nabla u_i - v_i^{(k+1)} \right)^2} \right)^2 = \sum_i \left(\nabla u_i - v_i^{(k+1)} \right)^2. \quad (2.4.2)$$

Remember according to quadratic functions minimization, a solution is simply the minimum point of the function curve. This in other terms represents the point of

zero gradient (point at which derivative of the function equal zero). Hence, we can solve for the solution u by introducing an arbituary function L representing the problem for solving for $u^{(k+1)}$, shown in Eq. 2.4.3, and finding its derivative equated to zero.

$$L = \lambda \|u - g\|_2^2 + \beta \|\nabla u - v^{(k+1)}\|_2^2 \tag{2.4.3}$$

Note that according to [45], v is a vector-valued function that is generally not a gradient derived from another function, where $v = (v^x, v^y)$. Subscripts in x and y correspond to partial derivatives with respect ot those variables. However, the superscripted v^x and v^y are used to denote the elements of v rather than subscript them, which would incorrectly suggest they are partial derivatives of the same function. In our case, this is a representation of a shrinkage operation.
Eq. 2.4.3 can be simplified by applying the concept of euclidean norm given in Eq. 2.4.1 and Eq. 2.4.2, yielding

$$L = \lambda \left(u - g\right)^2 + \beta \left(\left(u_x - v^x\right)^2 + \left(u_y = v^y\right)^2\right) \tag{2.4.4}$$

which when the derivative is performed, based on the satisfaction of the *Euler-Lagrange*, and equated to zero yields,

$$\lambda u - \beta \left(u_{xx} + u_{yy}\right) = \lambda g - \beta \left(v_x^x + v_y^y\right) \tag{2.4.5}$$

which finally simplifies into,

$$\left(\lambda I - \beta \nabla^2\right) u = -\beta \nabla . v + \lambda g \tag{2.4.6}$$

where ∇^2 is the discrete Laplacian matrix, and I is the identity matrix.

Remember, our goal is to solve the screened Poisson equation corresponding to solving for $u^{(k+1)}$ using an iterative linear solver. Such solvers are suited for problems of the form $Ax = b$, of which Eq. 2.4.6 can be modeled as such, with:

$$\begin{aligned} A &= \left(\lambda I - \beta \nabla^2\right), \\ x &= u, \\ b &= -\beta \nabla . v + \lambda g \end{aligned}$$

Hence, we can now use multigrid and conjugate gradient methods to solve for a solution u, which is the smoothed output. We henceforth experimentally performed our proposed extension of image smoothing by two distinct procedures:

1. Calculate shrinkage operator v based on [1], using Eq. 2.2.12.
2. Implement image smoothing problem in Eq. 2.4.6 by multigrid and (preconditioned) conjugate gradient iterative method. We apply various preconditioners, of which are reviewed in detail in Sec. 2.3.2.

The proposed smoothing algorithm is summarized in Alg. 12.

Algorithm 12 Proposed Smoothing Implementation

Data: Input image g, parameters λ, β and
convergence stopping criteria ε.
Result: Smoothed image u.
Step 1: Calculate gradient field v,
use it to compute $b = \lambda g - \beta\nabla \cdot v$ and store.
Step 2: Calculate $A = \lambda I - \beta\nabla^2$ and store it.
Step 3:
while (!convergence) **do**
  Solve $Ax = b$ using iterative method.
end while
Solution $u = x$

2.4.2 Boundary Processing

In some existing smoothing algorithms, especially implementations utilizing FFT solvers, boundary processing results in wrap-around errors in the resulting smoothing output in applications requiring higher smoothing strength. One of the justifications of utilizing spatial methods is the freedom to model boundary conditions better than in frequency domain methods. In FFT solvers, wrap-around errors occur along image boundaries due to the periodicity implied by the discrete Fourier transformation. To avoid wrap-around errors, one of the solutions is to introduce padding along the boundaries. However, this increased the processing time and memory requirement. In our approach, we manipulate the Laplacian operator in-order to eliminate wrap-around errors without increasing complexity.

The Laplacian operator we use in this paper is the following 5 point 3×3 kernel in the form of 2-D convolution,

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}. \tag{2.4.7}$$

Here, we assume that $f(x, y)$ is a pixel of position (x, y) in an image of size $M \times N$, which means that x and y must be $0, 1, \ldots, M-1$ and $0, 1, \ldots, N-1$, respectively. To process the image boundaries, we need to define $f(x, y)$ for $x < 0$, $x \geq M$, $y < 0$, and $y \geq N$. When we define,

- $f(-1, y) = f(M-1, y)$,
- $f(M, y) = f(0, y)$,

- $f(x,-1) = f(x,N-1)$, and
- $f(x,N) = f(x,0)$,

this kernel can be represented simply in a matrix notation form by the following, assuming that input image is 3×3

$$\begin{pmatrix} -4 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -4 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & -4 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & -4 \end{pmatrix}, \tag{2.4.8}$$

If opposite boundaries are assumed to be continous and the above matrix is used as the Laplacian operator, wrap-around errors occur similar to the processing in frequency domain. Therefore, we consider the following image boundary contraint to avoid boundary connectivity violations

- $f(-1,y) = f(0,y)$,
- $f(M,y) = f(M-1,y)$,
- $f(x,-1) = f(x,0)$, and
- $f(x,N) = f(x,N-1)$.

In this case, the Laplacian operator can be represented in the following form assuming that input image is 3×3,

$$\begin{pmatrix} -4 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & -4 & 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 2 & -4 & 1 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 & 0 & -4 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 2 & -4 \end{pmatrix}. \tag{2.4.9}$$

Now apart from our test case representation for the sake of easier understanding, in general the size of the Laplacian operator matrix is $MN \times MN$, where M and N are the dimensions of the input image. For example, in the case of an input image of size 1920×1080, the Laplacian operator is approximately $2Mega \times 2Mega$ (i.e,4T elements). It should be noted that this matrix is sparse, where there are at most five non-zero values in each row. Therefore, this does not have a significant impact of storage.

2.5 Implementation

In this section we investigate and compare the performance of the image smoothing algorithm using various spatial iterative methods. Furthermore, we present the effect of boundary processing choice on the smoothed result. We also present key optimization techniques tailored towards constant input resolution streams (e.g video inputs), taking advantage of pre-calculation and storage of computational costly components to further reduce processing time.

2.5.1 Multigrid

Algebraic multigrid method (AMG) is a general form of multigrid, which solely derives required information from the coefficient matrix A, as opposed to geometric multigrid.

We present experimental results based on v-cycle implementation for both non-preconditioned and preconditioned algebraic multigrid. Gauss-Seidel was used as the smoother to remove high frequency errors easily. Gauss-Seidel was preferred to Jacobi method as it has a superior convergence rate, as discussed in Sec. 2.3.2.

In the case of preconditioning, CG method was used as a preconditioner. Fig. 2.10 shows experimental results obtained by performing image smoothing by AMG with and without applying a preconditioner, with a stopping criteria of tolerance $\varepsilon = 10^{-6}$, which was as default. At this stage, every other parameter was kept constant.

Note that we tweaked this iterative method by first observing performance changes produced by each individual parameter, henceforth combining individual optimization to get the overall optimal conditions for image smoothing.

As it can be noted from Fig. 2.10, applying a preconditioner does improve the convergence rate of AMG, which conforms to the expectations of preconditioning as discussed in Chapter 2.3.2. Performing smoothing with a preconditioner on a full HD image(1920×1080 = 2.07 megapixels) may be an improvement compared to AMG without preconditioning, however, the processing time taken of 130 secs would not be practical for time sensitive applications. Hence, further optimization was performed to improve performance of AMG.

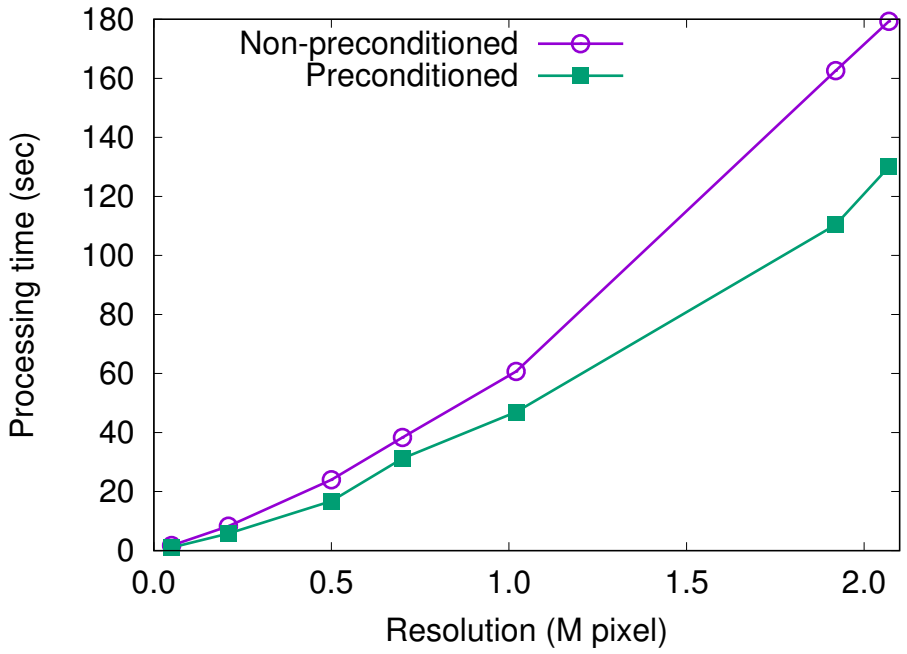


Figure 2.10: Result of algebraic multigrid method (AMG). Resolution vs time.

As we are dealing with a multigrid method, for further optimization we ran test to determine the optimal number of grids, ensuring removal of unnecessary grid computations. Our test image was a 256×256 RGB image. Table 2.1 illustrates the optimal number of grids and the associated processing time. From this, we notice that only 6 grid levels are necessary, with an optimal processing time of 0.66 seconds.

In image smoothing applications, in which details are not a priority but the fidelity of the overall structure(edges), we can relax the tolerance condition in a bid to further optimize this iterative method. Tolerance basically implies how strict we set conditions for convergence. This means that if the tolerance is small then the strict conditions are to be satisfied in order to reach convergence, while a large value of tolerance relaxes the conditions necessary for convergence. Hence, with very low values of tolerance an exact solution can be achieved, as in the case of FFT. However, not all applications, such as image smoothing require an exact solution. Hence, we can relax tolerance in AMG to tailor it to our image smoothing application, in a bid to attain further computationa l cost optimization.

Figures 2.11 and 2.12 depict the effect of tolerance on convergence rate, and the PSNR with reference to default tolerance $\varepsilon = 10^{-6}$, respectively. From Fig. 2.11, it can be noted that relaxing the tolerance leads to a reduction in the overall processing

Table 2.1: Preconditioned AMG coarsening level vs processing time (tolerance: $\varepsilon = 10^{-6}$, image size: 256×256).

Coarsest	Number of grids	Processing time (sec)
128×128	2	1.78
64×64	3	1.26
32×32	4	0.76
16×16	5	0.69
8×8	6	0.66
4×4	7	0.66
2×2	8	0.66

time.

From experimenting with various image resolutions,it was noted that high tolerance levels of 10^{-2} and 10^{-1} affect the output smoothing quality of higher resolution images more than low resolution images. Hence, we applied two different tolerance thresholds between low resolution and high resolution images. For low resolution images, we had a higher degree of freedom in that even a tolerance of 10^{-1} was sufficient for image smoothing, while a tolerance of 10^{-3} is sufficient.

Table 2.2: Optimized AMG for smoothing application

Resolution (Mpixel)	Non-optimized (sec)	Optimized (sec)	PSNR (dB)
0.05	1.8	0.45	84
0.21	9.0	1.62	85.1
0.50	22.5	4.02	106.3
0.70	39.6	5.55	113.4
1.02	64.8	8.22	108.9
1.92	142.3	15.88	109.5
2.07	185.7	17.18	112.3

Finally, by applying various optimization tweaks to AMG method, we obtained results given in Table 2.2. We can see a great improvement has been achieved through applying a preconditioner, grid and tolerance optimization, with computation time reducing from 185.7 secs to 17.18 secs in the case of a full HD image.

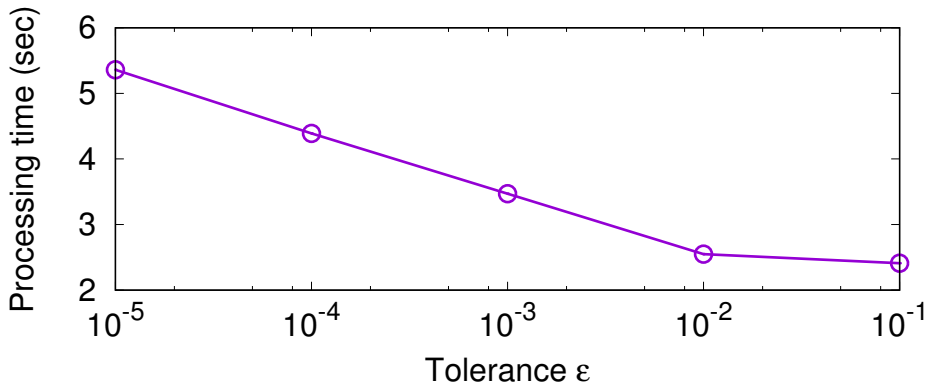


Figure 2.11: AMG: Tolerance vs processing time.

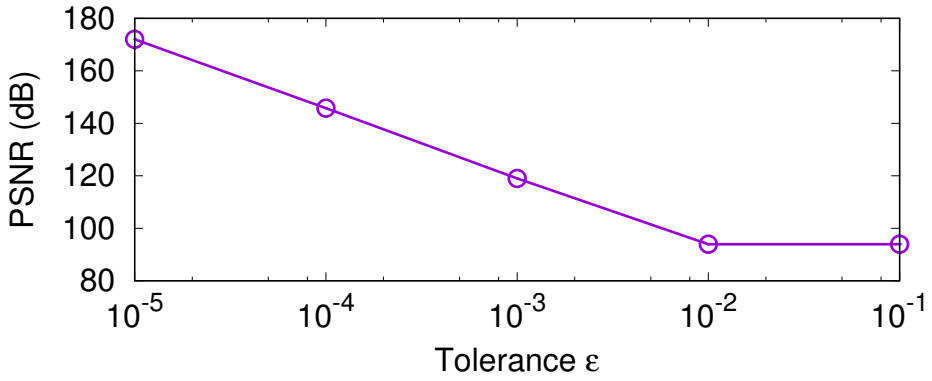


Figure 2.12: AMG: Tolerance vs PSNR.

2.5.2 Conjugate Gradient

Conjugate gradient method (CG) is a Krylov space iterative method. There are many variants, some of which are application specific. We performed image smoothing by conjugate gradient method, incomplete Cholesky factorization preconditioned conjugate gradient (ICCG) and multigrid preconditioned conjugate gradient method (MGCG).

Figure 2.13 shows the performance of non-preconditioned conjugate gradient method for various resolutions and tolerance values. Image smoothing using conjugate gradient has a relatively linear characteristic in relation to image resolution verse computational cost, as can be seen.

By relaxing the tolerance, convergence rate was improved linearly across all the used image resolutions. For images having resolutions lower than 1.5 megapixel, a tolerance level as low as 10^{-1} was sufficient for image smoothing purpose. How-

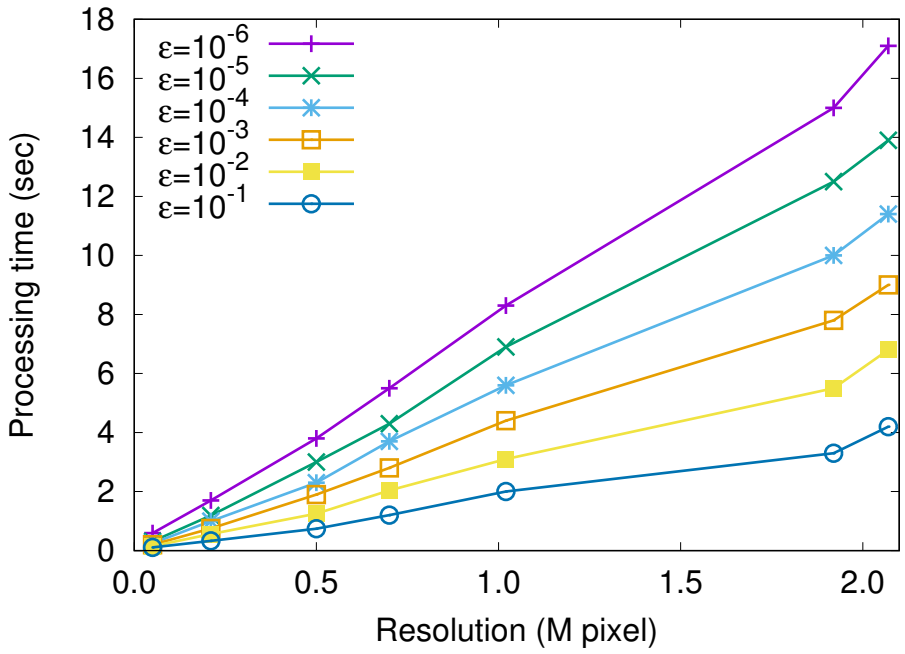


Figure 2.13: Result of conjugate gradient method (CG). Resolution vs time.

ever, images with resolutions equal or above 1.5 megapixels required a stricter threshold of 10^{-3} in order to sufficiently perform smoothing. Figure 2.14 depicts a better understanding of the tolerance changes in terms of PSNR, with reference to the default tolerance result. In image smoothing, a PSNR of 30dB or more represents an acceptable level of fidelity.

It can be noticed that convergence rate of CG method is more superior to AMG method. This is so because of the a-orthogonal nature of CG method, hence its convergence does not slag as a solution is approached.

ICCG

As discussed in Chapt. 2.3.2, introducing a preconditioner to a problem being solved by spatial iterative methods improves the conditioning. Hence, in order to improve the condition number of the coefficient matrix A, thereby clustering the eigen values, we first introduced an *incomplete Cholesky factorization* preconditioner into CG method. This is collectively referred to as ICCG.

Figure 2.15 shows the performance of image smoothing by ICCG, for various image resolutions. As in case of the previous experiments, tolerance was relaxed in order to obtain optimal conditions sufficient for image smoothing. As supported

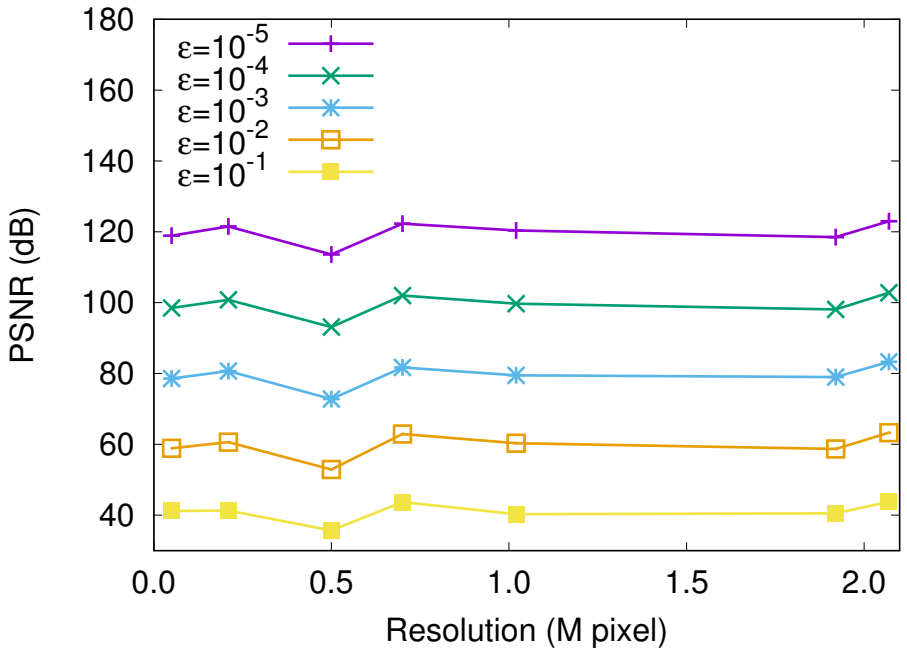


Figure 2.14: Result of conjugate gradient method (CG). Resolution vs PSNR.

by Sec. 2.3.2, applying IC preconditioner greatly improved the performance of CG. For a full HD image smoothing process, the computational time was reduced from 17.1 to 8.8 secs at a strict default tolerance of $\varepsilon = 10^{-6}$.

2.5.3 MGCG

We further went on to incorporate multigrid as a preconditioner, yielding results shown in Fig. 2.16. We applied two-grid, v-cycle multigrid method, relaxing the problem once per CG iteration. Clearly far more superior results were obtained, with the computation time reducing from 17.1 to 4.4 secs for a full HD image at default tolerance $\varepsilon = 10^{-6}$, compared to non-preconditioned CG. Hence by combining robustness of MG method with a-orthogonal properties of CG, MGCG has proven to effectively perform image smoothing with a reduced computational time.

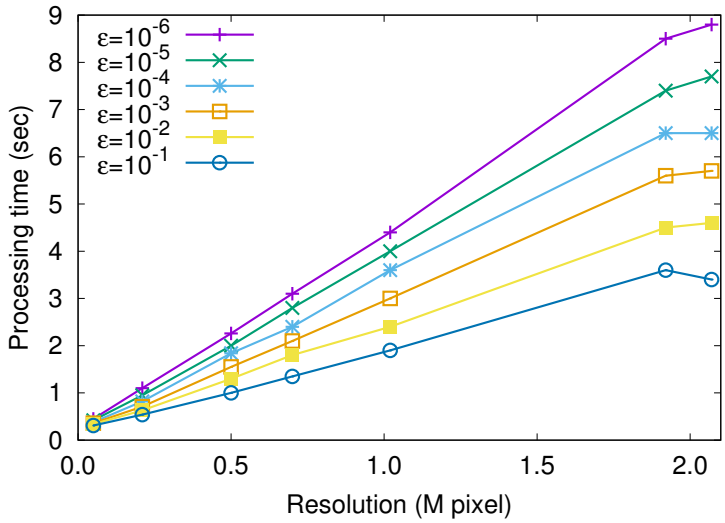


Figure 2.15: Result of incomplete Cholesky factorization preconditioned conjugate gradient method (ICCG). Resolution vs processing time.

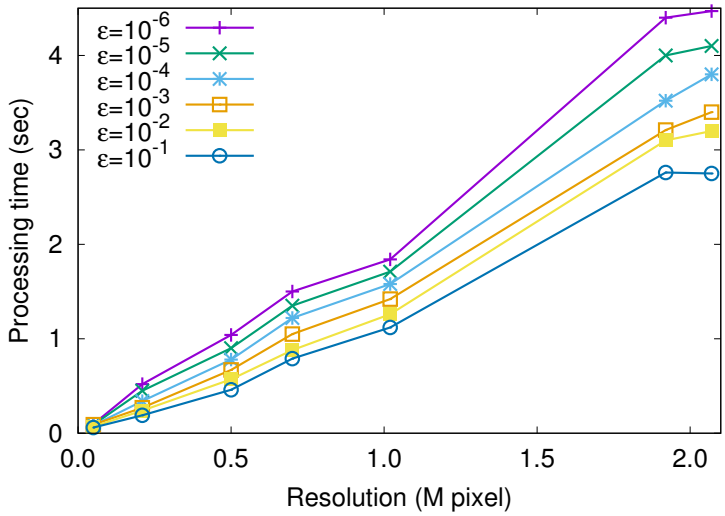


Figure 2.16: Result of multigrid preconditioned conjugate gradient method (MGCG). Resolution vs processing time.

2.5.4 Evaluation

In this subsection, we evaluate the performance of our proposed approach against frequency domain algorithms and existing state-of-the-art smoothing techniques such as [16, 31, 40, 70, 71]. All experiments shown in this section were performed on Intel Core i7 CPU @3.4 MHz.

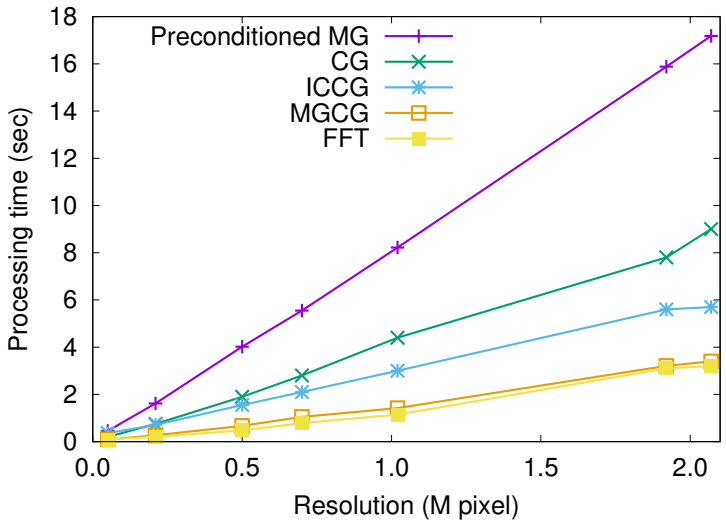


Figure 2.17: Performance comparison

Figure 2.17 illustrates the computation cost of both spatial iterative methods and FFT solvers. For the iterative solvers, the outlined results were taken at tolerance of 10^{-2}, yielding acceptable results. It can be seen that FFT and MGCG at the specified condition are very closely competitive in terms of computation time without compromising quality. In applications involving image smoothing that do not require a strict tolerance criteria, such as 10^{-1}, MGCG converges faster than FFT in such cases, hence being scalable to tailor for an application.

Table 2.3 shows simulation results obtained at various input resolutions, also comparing our approach with existing state-of-the-art smoothing techniques. Fig-

ures 2.24 to 2.31 show visual output of our proposed approach against other existing methods. These results show that our approach not only presents a good balance

Table 2.3: Processing time (secs) comparision of our proposed approach with existing algorithms.

Method/Resolution	0.05M	0.21M	0.5M	0.7M	1.02M	1.92M	2.07M
FastGlobal [16]	0.026	0.028	0.032	0.044	0.061	0.144	0.169
Bilateral [40]	0.097	0.109	0.106	0.139	0.162	0.232	0.254
TreeFiltering [71]	0.149	0.166	0.197	0.246	0.310	0.760	0.838
Proposed	0.176	0.196	0.242	0.289	0.378	0.771	0.850
Non linear [70]	0.245	0.398	0.468	0.585	0.735	1.431	1.611
Lo Gradient [3]	1.065	0.980	1.539	1.708	2.243	4.101	5.596
WLS Filtering [31]	1.616	1.797	2.229	2.694	3.981	6.531	6.748

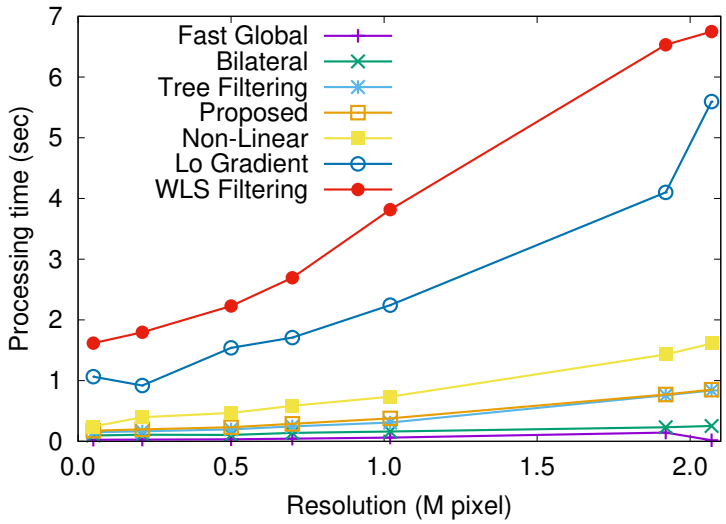


Figure 2.18: Processing time comparison using data in Table 2.3

between complexity and quality, but also presents a compatitive approach with more flexibility and scalability depending on the application requirements and resources available.

Boundary processing

We present the results and compare boundary performance between existing FFT smoothing algorithms and our proposed approach. Figure 2.19:(a), (b) and (c)

show the input image, and smoothing outputs obtained from existing algorithms and our approach, respectively. Our boundary conditions implemented are modeled in Section 2.4. In the case of FFT implementations, it can be seen that so-called wrap-around errors occur along the image boundaries due to the periodicity implied by frequency domain processing. In applications requiring the smoothing strength to be relatively great, these undesired artifacts are more predominant. This wrap-

((a)) Input. ((b)) FFT. ((c)) Proposed.

Figure 2.19: Image smoothing boundary processing results. The input image is obtained from `publicdomainpictures.net` (image 8363),

around error can be suppressed using padding. However, this increases the image size significantly in order to completely remove these artifacts, thereby impacting performance. In our proposed approach, padding is not necessary as we enforce better boundary conditions mathematically when generating the Laplacian operator matrix.

In the following subsections, we outline two optimization techniques. One is suitable for constant input resolution feeds and the other for low processing and memory hardware.

Flow Optimization

In this subsection, we present application specific optimization. In applications which have a constant input resolution stream such as in the case of video processing, further processing optimization is possible. Taking advantage of spatial domain processing and identifying that calculating $\lambda I - \beta\nabla^2$ of the smoothing problem $(\lambda I - \beta\nabla^2)\, u = -\beta\nabla.v + \lambda g$ only depends on the input's resolution (size), this computational costly stage can be pre-calculated and stored in memory. Being sparse, the memory storage required is almost trivial. Hence, by applying the flow optimization shown in Fig. 2.20, we achieve a performance improvement of approximately 46% over existing smoothing algorithms considered in this thesis. This alteration does not affect the smoothing quality as shown in Fig. 2.22.

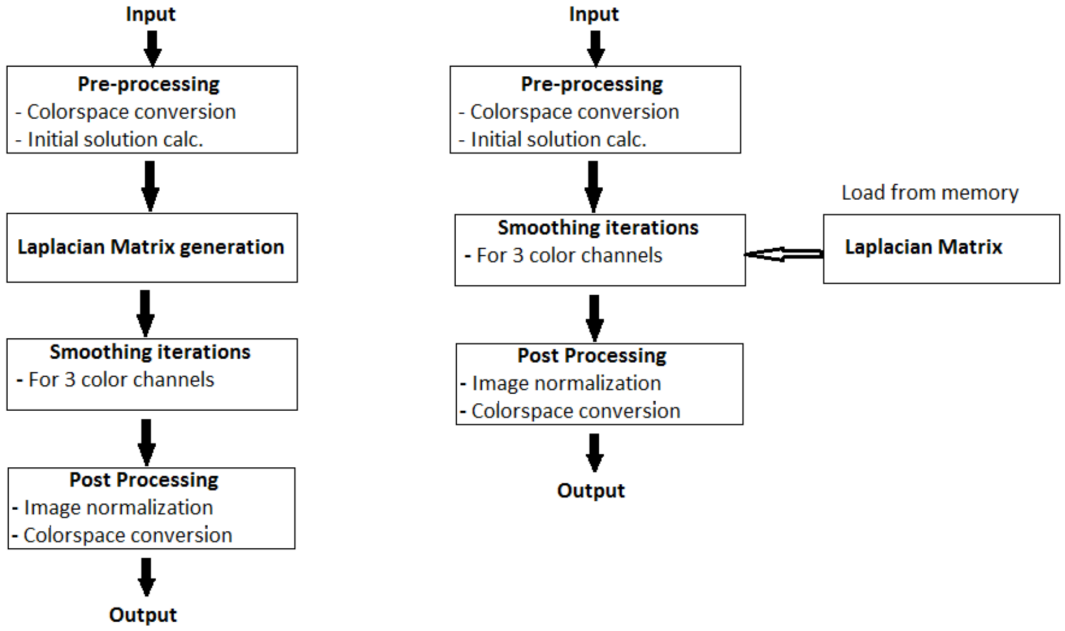


Figure 2.20: Flow optimization chart.

Scaling Optimization

As can be observed from the results, the presented image smoothing implementations possess a relatively linear relationship between input resolution and processing time. This implies that larger image resolutions require more time to be processed.

With this, we decided to incorporate image scaling algorithms into the image smoothing algorithm to support lightweight hardware with low processing power and memory. By image scaling, we simply mean the acceptable reduction in image resolution, on which image smoothing is applied, with finally interpolating the result to the original resolution and performing similarity control. This is illustrated in Fig. 2.23.

Such an implementation results in a trade-off between image smoothing quality and computational time. Hence, it should be noted that using complex scaling algorithms do thrive to maintain the smoothing quality, but may not provide much advantage in terms of computation cost, as their complexity affects overall processing time. On the other hand, simple scaling algorithms are cheaper to implement at the cost of quality. Therefore, including image scaling in image smoothing has its pros and cons, which can be exploited depending on an application and hardware.

Figures 2.21 and 2.22 summarizes the processing time performance gain and

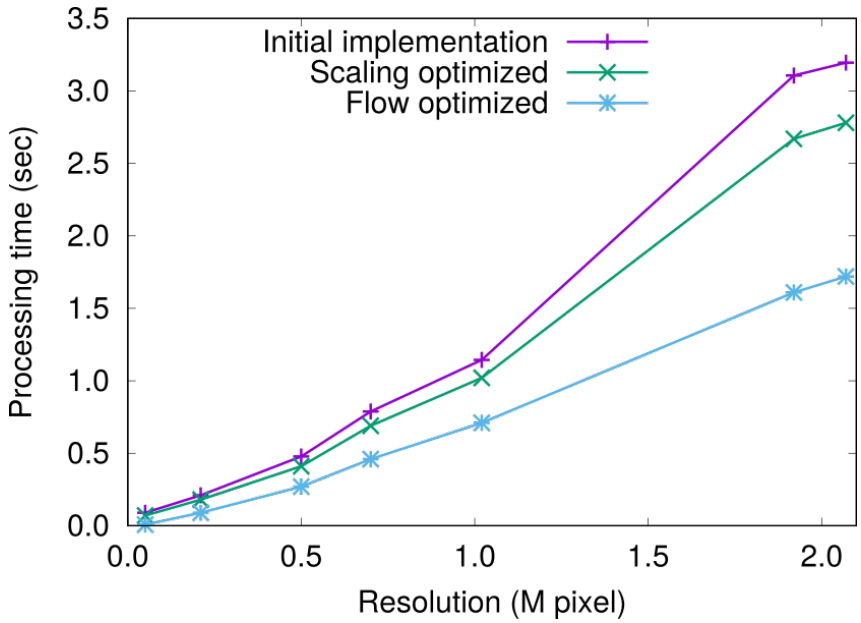


Figure 2.21: Computation time benefit of flow and scaling optimization techniques

smoothing quality of the two optimization techniques compared with the base proposed algorithm, respectively.

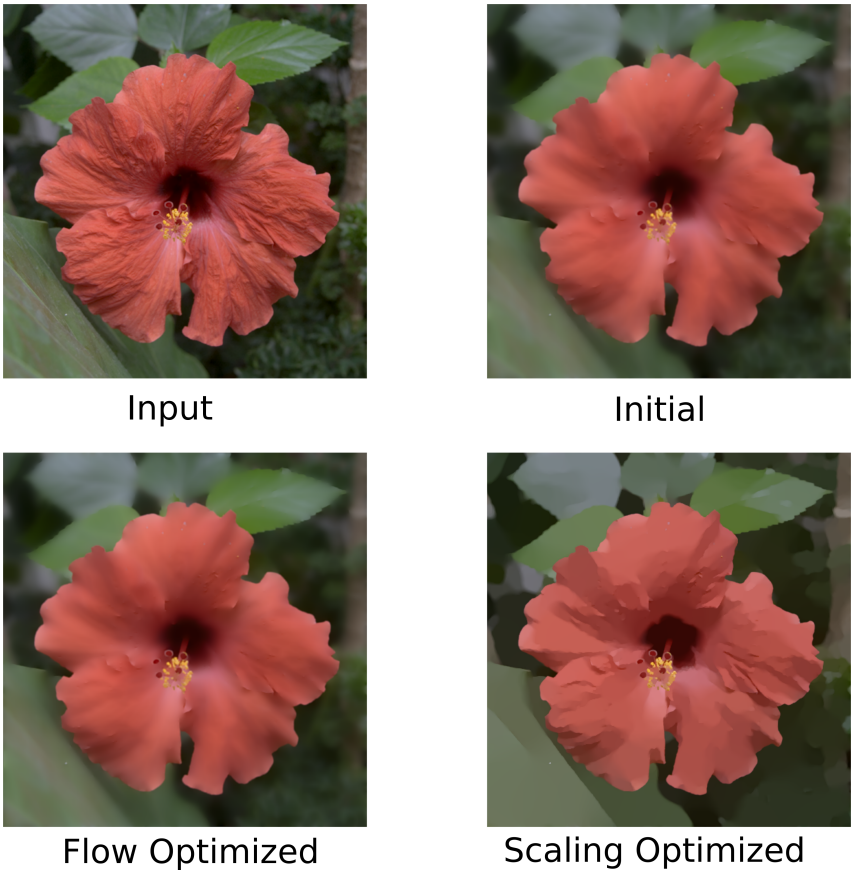


Figure 2.22: Smoothing quality of optimization techniques

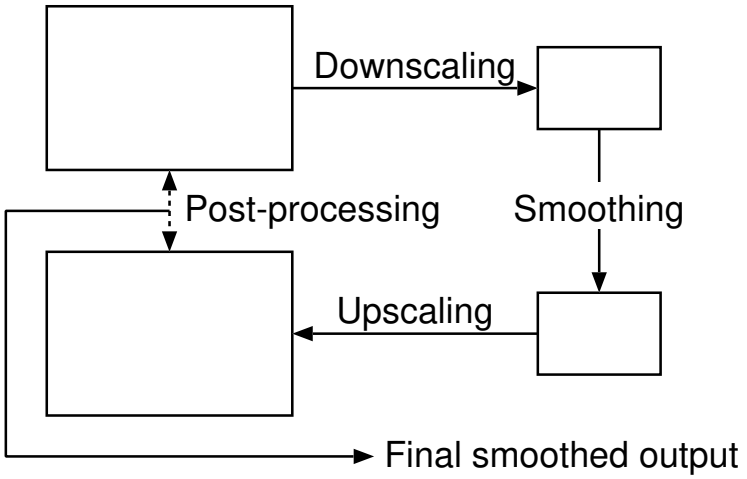
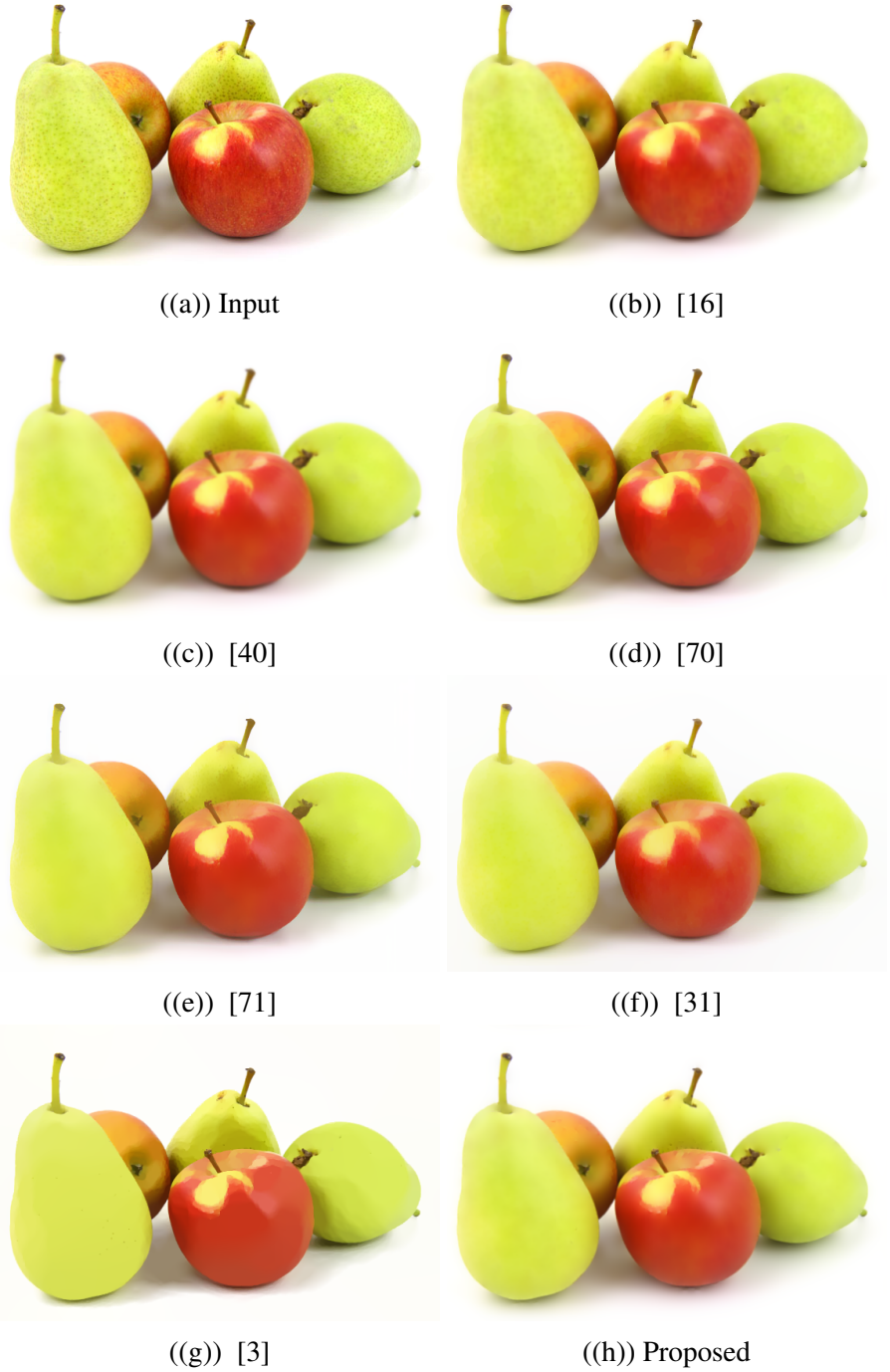


Figure 2.23: Scaling optimization flow chart



((a)) Input ((b)) [16]

((c)) [40] ((d)) [70]

((e)) [71] ((f)) [31]

((g)) [3] ((h)) Proposed

Figure 2.24: Smoothing simulation results. Image from `publicdomainpictures.net` (image 3421)

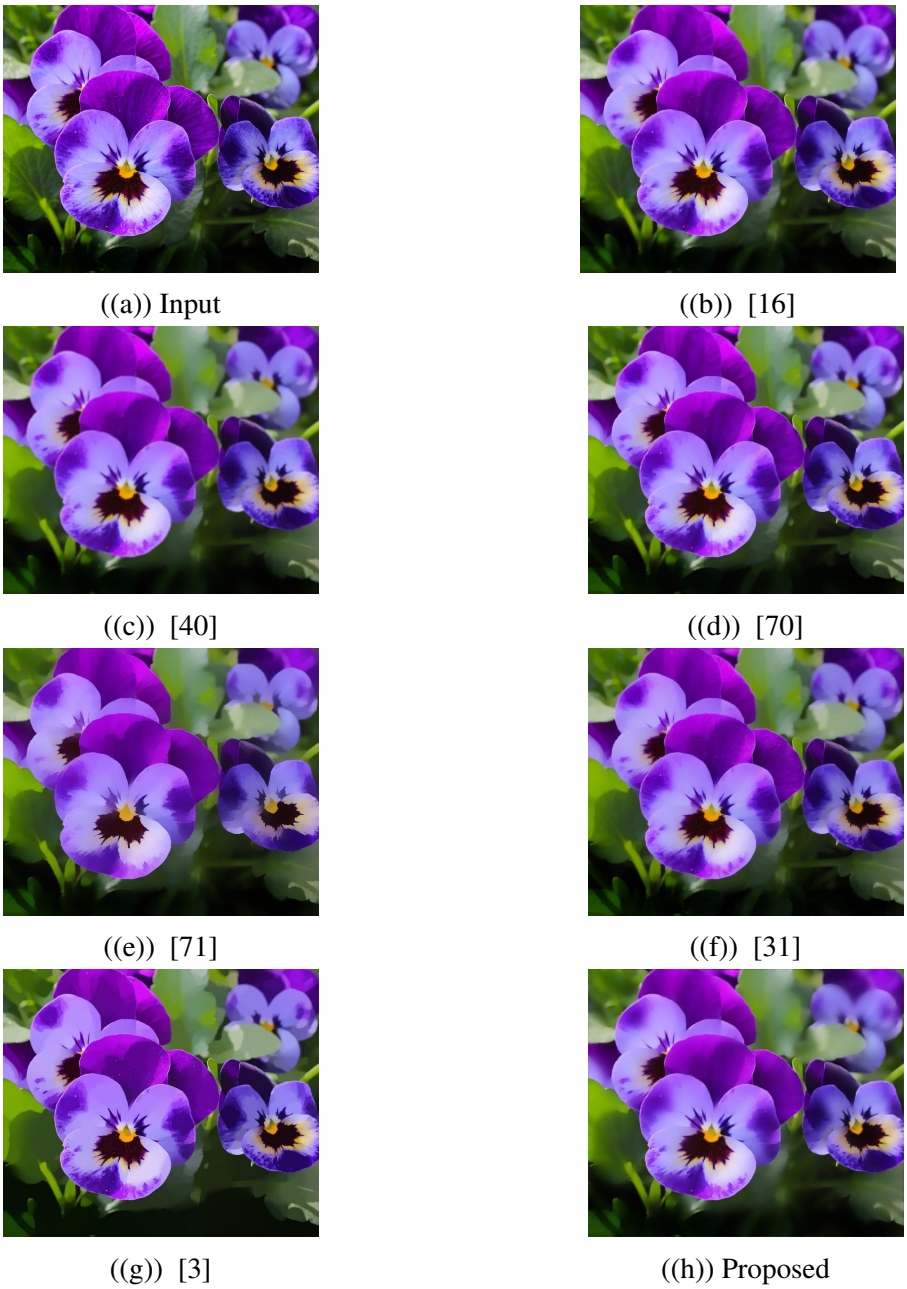
((a)) Input ((b)) [16]

((c)) [40] ((d)) [70]

((e)) [71] ((f)) [31]

((g)) [3] ((h)) Proposed

Figure 2.25: Smoothing simulation results



((a)) Input ((b)) [16]

((c)) [40] ((d)) [70]

((e)) [71] ((f)) [31]

((g)) [3] ((h)) Proposed

Figure 2.26: Smoothing simulation results. image from internet

((a)) Input

((b)) [16]

((c)) [40]

((d)) [70]

((e)) [71]

((f)) [31]

((g)) [3]

((h)) Proposed

Figure 2.27: Smoothing simulation results

((a)) Input ((b)) [16]

((c)) [40] ((d)) [70]

((e)) [71] ((f)) [31]

((g)) [3] ((h)) Proposed

Figure 2.28: Smoothing simulation results

((a)) Input

((b)) [16]

((c)) [40]

((d)) [70]

((e)) [71]

((f)) [31]

((g)) [3]

((h)) Proposed

Figure 2.29: Smoothing simulation results. image from internet

((a)) Input ((b)) [16]

((c)) [40] ((d)) [70]

((e)) [71] ((f)) [31]

((g)) [3] ((h)) Proposed

Figure 2.30: Smoothing simulation results. image from internet

((a)) Input ((b)) [16]

((c)) [40] ((d)) [70]

((e)) [71] ((f)) [31]

((g)) [3] ((h)) Proposed

Figure 2.31: Smoothing simulation results. image from internet

2.6 Conclusion

We successfully implemented image smoothing using spatial iterative methods. It is observed that MGCG has superior performance among the iterative solvers methods demonstrated in the experiments. When applied to a full HD image, MGCG converged in 3.4 secs, while ICCG in 5.7 secs and CG in 9 secs. Both qualitative and quantitative analysis show that our approach provides a good balance between complexity and quality compared with existing algorithms. Furthermore, We investigated two implementation optimization techniques to further accelerate our proposed method.

The first was using image downsampling in order to reduce the resolution required to processing in smoothing algorithm. By utilizing downsampling, we managed to reduce the processing time by approximately 21.6 %. In addition to processing time reduction, the use of downsampled images on which to perform image smoothing reduces the memory requirement. However, a trade-off between processing time and smoothing quality is introduced. By using simple downsampling algorithms, processing time advantage is gain at the expense of smoothing quality. This optimization is suitable for low processing power and memory hardware applications.

The second approach involved the pre-calculation and storage of the computationally costly Laplacian operator matrix, which depends only on the resolution of the input.By pre-calculating and storing this operator only once, inputing images having the same resolution results in a processing time reduction of approximately 46.1 %. This optimization is suitable for same resolution input streams, extending its application to video processing.

Iterative solvers provide the user greater degree of flexibility, as various components which affect processing time can be tweaked to achieve an application specific desired outcome. It can be concluded that iterative methods are scalable and flexible.

Finally, our approach includes image boundary constraint which prevents introduction of wrap around errors which are inherent to FFT smoothing implementations under extreme smoothing conditions.

Chapter 3

Retinex-based Enhancement and Haze Removal with FPGA Implementation

3.1 Introduction

Digital image and video processing plays an essential role in modern day consumer electronics, with the increasing demand in digital media driven by current social trends. With continued advancement in digital imaging applications, real-time image (video) enhancement and haze removal are among key research topics influencing consumer electronics.

Image enhancement schemes can be categorized into two groups; adaptive and non-adaptive schemes. Non-adaptive schemes compensate each pixel value uniformly based on given equations [72], while adaptive schemes refer surrounding pixels to reproduce a high quality image. Retinex theory [73] [74] is a well-known adaptive image enhancement scheme, its variant which we shall consider in this paper. Haze removal methods can be categorized as; single and multiple image schemes. Single-image schemes are more popular, requiring less overhead.

The quality of images and video taken from outdoor scenes is influenced by scattering of light which occurs before reaching the camera sensor. The amount of scattering depends on the distance between the scene points and the sensor, making degradation spatial-variant [75]. In haze (fog) weather, an elevated presence of atmospheric particles such as water-droplets results in more scattering, resulting in low contrast and color fidelity images. Scattering is caused by two basic phenomena, which are attenuation and airlight. According to [75, 76], haze removal depends upon the unknown depth information. This particularly makes haze removal

a challenging task. Numerous haze removal methods have been proposed [77–83] in recent years with significant advancements. Most dehazing methods use a variety of visual cues to capture deterministic and statistical properties of haze images [84]. Haze removal is highly desired in computer vision applications. It not only serves to significantly increase the visibility of the scene and correct the color shift, it can also benefit many vision algorithms and advanced image editing.

Both Retinex-based image enhancement and haze removal are computation costly. Considering real-time processing in applications such as monitoring systems, autonomous cars, and live streaming systems, there still remains much room for the development of efficient hardware implementation of image enhancement and haze removal. Motivate by this, in this paper we propose an architecture supporting both real-time Retinex-based image enhancement and haze removal, at low memory and process overhead utilizing a single module.

Our proposed implementation and architecture efficiently supports both Retinex-based image enhancement and haze removal. Efficiently leveraging the similarity between Retinex-based image enhancement and haze removal, and modifying the process, we present a novel architecture optimized for both processes at low overhead cost.

3.2 Related Works

Various researchers have proposed algorithms to address image enhancement and haze removal, commonly independent of each other. Considering Retinex based image enhancement, Shen and Hwang [85] presented a color image processing using a robust envelope to improve the visual appearance of an image. Guo et al. [86] introduced a visibility restoration method for a single image using Retinex algorithm on luminance component, while Fattal [87] presented a novel transmission estimation method to increase scene visibility and recover haze-free image. Marsi and Giovanni [88] proposed and FPGA implementation for illuminance-reflectance video enhancement in a single module. In Shiau et al. [89], hardware implementation of haze removal is presented. They, [89], proposed an 11-stage pipelined hardware architecture. However, these existing algorithms highlighted require high memory and computation, more so at higher resolutions. Furthermore, most of these algorithms are optimized for either enhancement or haze removal only.

Furthermore, Ren, Wenqi, et al. [84] proposed a multi-scale convolutional neural network dehazing method. In this proposal, a holistic prediction of the transmission map using a dataset trained neural network is utilized. In this case training is required in order to learn mapping, which is a complex task. In [90], an end-to-end image dehazing method called Densely Connected Pyramid Dehazing Network (DCPDN) is proposed. This jointly learns the transmission map, atmospheric light and dehazing all together by directly embedding the atmospheric scattering model into the network. By this, the method follows the physics-driven scattering model for dehazing. Dataset training is required in this implementation as is in [84].

Galdran, Adrian, et al. [91] presents a dual relationship between image dehazing and non-uniform illumination separation, applying Retinex operation on an inverted image followed by another image inversion in order to obtain a dehazed output. It is generally concluded that Retinex and dehazing can be connected by a simple linear relationship. The outcome of this was to demonstrate the general usability of existing Retinex implementations for haze removal based on a simple linear relationship, not to provide output performance gain over existing approaches.

3.2.1 Single Image Dehazing via Multi-Scale Convolutional Neural Networks

Ren *et al.* [84] proposes a multi-scale deep neural network for single-image dehazing by learning the mapping between hazy images and their corresponding transmission maps. The proposed algorithm consists of a coarse-scale net which predicts a holistic transmission map based on the entire image, and a fine-scale net which refines results locally. This technique requires dataset training, as neural networks are utilized in order to achieve haze removal.

For each scene, the transmission map $t(x)$ is estimated based on a multi-scale convolutional neural network (MCNN). The coarse structure of the scene transmission map for each image is obtained from the coarse-scale network, and then refined by the fine-scale network. Both coarse and fine scale networks are applied to the original input hazy image. The output of the coarse network is passed to the fine network as additional information, hence refining the coarse prediction with details.

In order to learn the mapping between hazy images and corresponding transmission maps, minimizing the loss between the reconstructed transmission $t_i(x)$ and the corresponding ground truth map $t_i^*(x)$ is performed by

$$L\left(t_i(x), t_i^*(x)\right) = \frac{1}{q}\sum_{i=1}^{q}||t_i(x) - t_i^*(x)||^2 \tag{3.2.1}$$

where q is the number of hazy images in the training set. Equation 3.2.1 is minimized using stochastic gradient descent method with the backpropagation learning rule.

The atmospheric light, **A**, is estimated by selecting 0.1% darkest pixels in a transmission map $t(x)$. After computing $t(x)$ and **A**, the final scene radiance $J(x)$ is recovered by

$$J(x) = \frac{I(x) - A}{\max\{0.1, t(x)\}} + A \tag{3.2.2}$$

3.2.2 On the Duality Between Retinex and Image Dehazing

In the work done by Galdran *et al.* [91], the basic relationship between Retinex enhancement and haze removal is investigated. While image enhancement and haze removal are two apparently unrelated problems, their goal is to show that they can be connected by a simple linear relationship. They present theoretic proof that Retinex on inverted intensities is a solution to image dehazing problem. This is based on an observation that most Retinex-based algorithms have the characteristic feature of always increasing image brightness, which turns them into ideal candidates for

effective image dehazing by directly applying Retinex to a hazy image whose intensities have been inverted.

Their main contribution is a formal proof of the following direct relationship between Retinex and haze removal:

$$\text{Dehazing}(I) = 1 - \text{Retinex}(1 - I) \tag{3.2.3}$$

According to this research, existing Retinex-based algorithms can be adopted to dehaze images directly by incorporating two intensity-inversion operations, without modifying Retinex-based algorithms. It should be noted that the goal of this work is not to produce results largely improving those of haze removal state-of-the-art algorithms, but to demonstrate the general usability of existing Retinex implementations for the task of haze removal. Furthermore, the presented simple linear relationship between Retinex and haze removal does not take atmospheric light, **A**, into account as this results in a more complex problem. Hence, the output results by ignoring **A** lead to a disparate color recovery in different images.

3.3 Preliminary

3.3.1 Retinex-based Image Enhancement

The Retinex theory [92] deals with compensation for illumination effects in images. This introduces the lightness and color perception of the human visual system, and is based on the property of the color constancy phenomenon, in that humans can recognize and match colors under a wide range of different illuminations. This theory decomposes an input image $I(\boldsymbol{x})$ into two different images, defined by

$$I(\boldsymbol{x}) = L(\boldsymbol{x})J(\boldsymbol{x}), \tag{3.3.1}$$

where $L(\boldsymbol{x})$ and $J(\boldsymbol{x})$ is the illumination image and reflectance image, repectively. The benefits of such decomposition include the possibility of removing illumination effects, enhancing image edges, and correcting the colors in images by removing illumination induced color shifts [86].

Image enhancement can be achieved by extracting $L(\boldsymbol{x})$ from $I(\boldsymbol{x})$ in order to generate $J(\boldsymbol{x})$, as an illumination-independent image. The logarithmic expression of the reflectance image $J(\boldsymbol{x})$ can be expressed by

$$\begin{aligned} J(\boldsymbol{x}) &= \frac{I(\boldsymbol{x})}{L(\boldsymbol{x})}, \\ j(\boldsymbol{x}) &= i(\boldsymbol{x}) - l(\boldsymbol{x}). \end{aligned} \tag{3.3.2}$$

where $i = \log I$, $l = \log L$, and $j = \log J$. Fig. 3.1 highlights the general flow chart of Retinex-based algorithms.

Several illumination models are proposed so far based on Retinex theory, such as Path-based [73], Center/Surround based [74] and Variational model [93], just to mention a few. Path-based algorithms are the primary methods in Retinex theory [92], having high computation complexity with a need for too many parameters [73]. The initial implementation relied on stochastic theory, with subsequent research utilizing multi-resolution image pyramids [94] [95].

Center/Surround algorithms are based on assumption that the illumination component tends to vary smoothly, while reflectance changes at sharp edges [74]. Therefore the output relectance values can be obtained by subtracting a blurred given image [96]. Center/surround algorithms, including SSR (single scale Retinex) and MSR (multiscale Retinex) [97] are easily implemented but require a large number of parameters.

Path and center based models are easily implemented but require a large number of parameters. Hence, these were not considered in our FPGA implementation as the require more memory and computation resources than proposed.

The variational model [93](Kimmel's variational model), assumes spatial smoothness of the illumination field. In addition, knowledge of the limited dynamic range of the reflectance is used as a constraint in the recovery process [96]. A modification of this variant was implemented in this paper, recognizing variational model as one of the most suitable models for practical applications in terms of computational cost and image quality, suitable for our real-time FPGA architecture [2].

The variational model algorithm is constructed to minimize the following penalty function,

$$F[l] = \int_{\Omega} \left(|\nabla l|^2 + \alpha (i-l)^2 + \beta |\nabla (i-l)|^2 \right) d\boldsymbol{x}, \tag{3.3.3}$$

where α and β are weight parameters, i and l represent the logarithmic expression of input image I and illumination image L, respectively. Penalty terms, $|\nabla l|^2$, $(i-l)^2$, and $|\nabla (i-l)|^2$ represent spatial smoothness of the illumination image, closeness between l and i, and spatial smoothness of the reflectance image j, respectively. The illumination image l which minimizes the penalty $F[l]$ is iteratively calculated using a projected normalized steepest descent algorithm.

Fig. 3.1 illustrates the flow of the Retinex image enhancement with illumination correction.

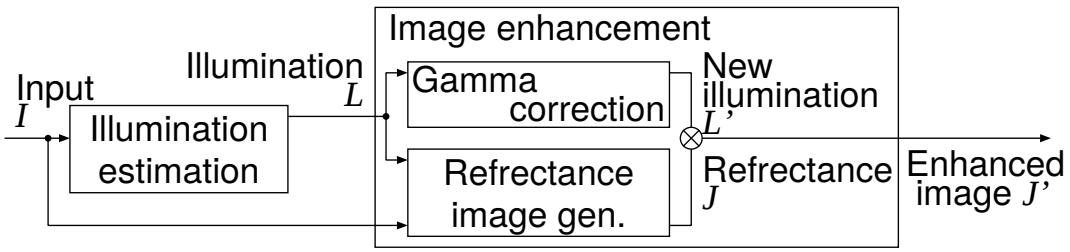


Figure 3.1: The flow of the Retinex image enhancement

By utilizing such adaptive image enhancement methods, *halo* artifacts are observed in the enhanced images. These are caused because such methods utilize the

constraint that the illumination image should be spatially smooth. When the illumination is estimated in the regions around the edge with this constraint, these regions in reflectance image tend to be either over-enhanced or insufficiently enhanced. Hence there are two types of halo artifacts; positive and negative. In the variational model, positive halo artifacts are successfully suppressed using a constraint $i \leq l$ in iterative calculation while leaving negative halo artifacts present [98]. Various halo effect suppression techniques have been investigated in [85, 88], which however are computation costly. In [98] we proposed a halo artifacts reduction method, with a small area overhead.

3.3.2 Haze Removal

The haze image model [82, 87, 99, 100], which consists of direct attenuation model and airlight model is generally expressed by,

$$I(\boldsymbol{x}) \;=\; J(\boldsymbol{x})T(\boldsymbol{x}) + A(1 - T(\boldsymbol{x})), \tag{3.3.4}$$

where I is the observed luminance representing the input haze image, J is the scene radiance representing the restored haze-free image, T is the medium transmission describing the portion of the light that is not scattered and reaches the camera, and A is the global atmospheric light. The goal of haze removal is to recover J from I using estimated T and A by,

$$J(\boldsymbol{x}) \;=\; \frac{I(\boldsymbol{x}) - A}{T(\boldsymbol{x})} + A, \tag{3.3.5}$$

In general, T and A are estimated using dark channel prior [75]. The dark channel prior is a kind of statistics of the haze-free outdoor images. It is based on an observation that most local patches in the haze-free outdoor images contain some pixels which have very low intensities in at least one color channel. Hence the minimum intensity in such a patch should have a very low value. In [75], the *dark channel* of an arbitrary image J is defined as

$$J^{\mathrm{dark}}(x) \;=\; \min_{\boldsymbol{y}\in\Omega(\boldsymbol{x})} \left(\min_{c\in\{\mathrm{R,G,B}\}} J^c(\boldsymbol{y}) \right), \tag{3.3.6}$$

where J^c is the color channel of J comprising of RGB components, and $\Omega(\boldsymbol{x})$ depicts a local patch centered at $\boldsymbol{x}$. The low intensity of the dark channels is due to shadows, colorful objects or surfaces and dark objects in images. According to the observation in [75], if J is a haze-free outdoor image, the intensity of J^{dark} is low and tends to be zero except for the sky region in an image. Due to additive airlight,

a haze image is brighter than its haze-free version. Hence the dark channel of the haze image will have higher intensity in regions with denser haze. Therefore, the intensity of the dark channel is a rought approximation of the thickness of the haze.

In [75], the transmission T is determined using soft matting. However, this approach requires a high computation cost. Motivated by this, some approaches use edge-preserving smoothing such as bilateral filters for estimating T with reasonable processing cost [101]. In our approach, we use edge-preserving smoothing based on the cost minimization function in Eq. (3.3.3) and [102] to generate the transmission T instead of soft matting. Hence, in a complimentary approach, we use Retinex-based image enhancement to suppliment haze removal at a low overhead resource cost.

3.4 Proposed Architecture

The block diagram of our proposed FPGA architecture is shown in Fig. 3.2 and Fig. 3.5. The logic of this architecture is shown in Fig. 3.3. This architecture accommodates both Retinex-based enhancement and haze removal using a single module, with a low overhead resource cost as opposed to using separate modules.

This architecture consists of three parts (Figs. 3.2 and 3.3); (1) Gaussian pyramid generation part, (2) illumination/transmission estimation part, and (3) image enhancement/haze-removal part.

We utilize Gaussian pyramid downsampling in order to realize low block memory size hardware requirement. Considering spatial smoothness characteristic of the illumination field, the effects of downsampling are tolerable.

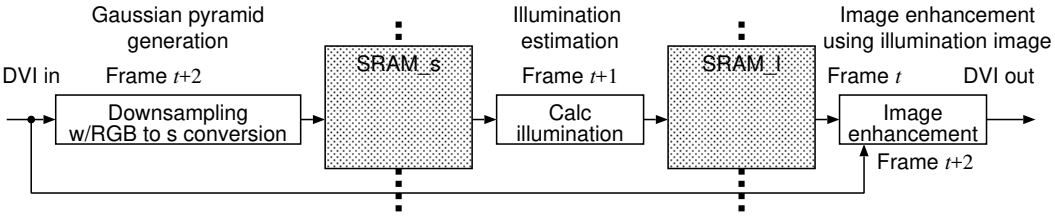


Figure 3.2: The block diagram of the proposed architecture [2].

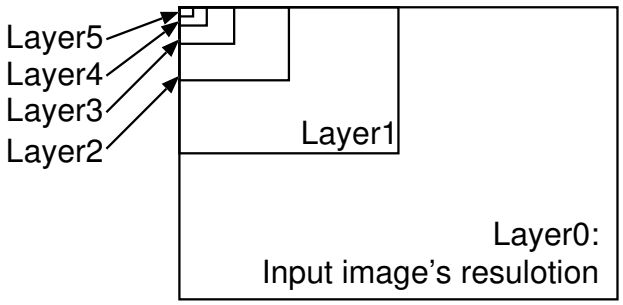


Figure 3.3: Layer hierarchy for illumination estimation.

Illumination and transmission estimation are performed on layers 5, 4, and 3

of Fig. 3.3, enabling accelerated iterations with low memory requirements. Figure 3.3 illustrates the scaling relationship between successive downsampled image layers, which are used as iterative inputs in the estimation process. Therefore by downsampling, the size of the buffers required are significantly reduced since the size of layers 3, 4 and 5 are $1/64$, $1/256$ and $1/1024$ of the resolution of the input image, respectively. The adaptation of Gaussian in our approach presents a computational efficient approximation, especially for FPGA. The use of double buffering in Fig. 3.5 prevents memory access conflict. The original resolution is reconstructed using bicubic interpolation. In order to combat blur effect inherent to bicubic interpolation, we implement the constraint $i \leq l$. Fig. 3.4 illustrates blur edge handling by this constraint.

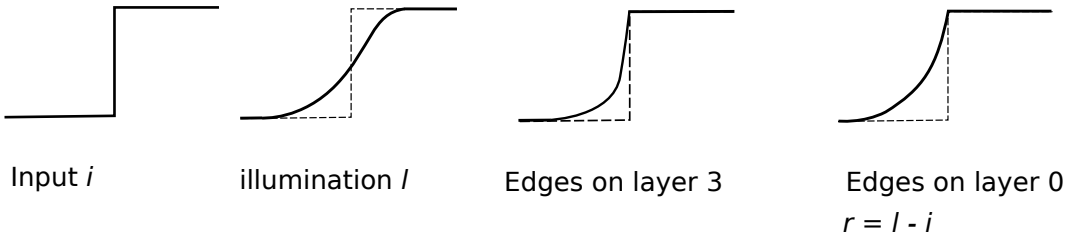


Figure 3.4: Edge preservation using constraint $i \leq l$

In Fig. 3.5, showing the illumination and transmission estimation module of Fig. 3.2, we have the following:

$$\begin{aligned}
G_A(\boldsymbol{x},\boldsymbol{y}) &= \nabla l_{j-1}^{(k)}(x,y), \\
G_B(\boldsymbol{x},\boldsymbol{y}) &= \nabla s^{(k)}(x,y), \\
\mu_A &= \sum_x \sum_y G(x,y)^2, \\
\mu_B &= -\sum_x \sum_y G(x,y)\nabla G(x,y), \\
\mu_{NSD} &= \frac{\mu_A}{\alpha\mu_A} + (1+\beta)\,\mu_A, \\
G(\boldsymbol{x},\boldsymbol{y}) &= -G_A(x,y) + \\
&\quad \alpha\left(l_{j-1}^{(k)}(x,y) - s^{(k)}(x,y)\right) + \\
&\quad \beta\left(G_A(x,y) - G_B(x,y)\right)
\end{aligned} \tag{3.4.1}$$

where *k* is the layer number, *j* the iteration index is $j = 1, 2, ..., T_k$ and *s* represents decimated image. The illumination/ transmission estimation is given by,

$$l_j^{(k)} = \max\left(l_{j-1}^{(k)}(x,y) - \mu_{NSD}G(x,y), s^{(k)}(x,y)\right) \tag{3.4.2}$$

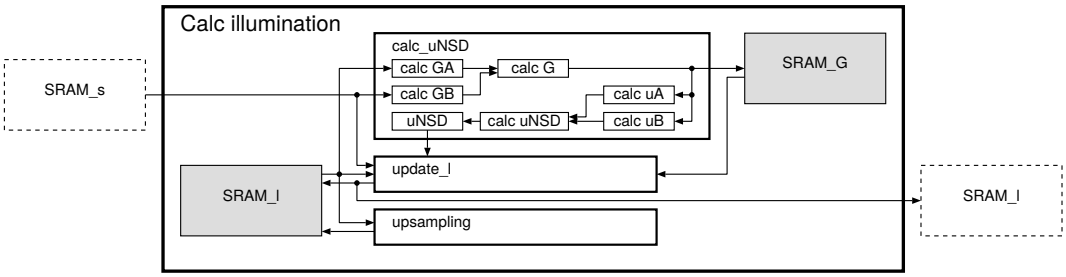


Figure 3.5: Expanded view of Illumination and transmission estimation module

This module has a throughput of 1 pixel/cycle, achieved by utilizing line buffers for laplacian calculation, and double buffering of SRAM_I.

The components of our proposed architecture work in a pipeline manner, each component processing its corresponding frame. In this paper we do not utilize an external frame buffer to storing input frames temporally. To compensate for this, we leverage the close similarity characteristic of consecutive frames enabling estimation component reuse. As shown in the Fig. 3.2, an input frame $t+2$ is enhanced by using the illumination/ transmission estimated from a preceding frame t, without latency. If an external frame buffer which stores two successive input frames is used, each input frame can be enhanced with the corresponding estimated illumination with a latency of two frames delay. Hence, the advantage of our approach, further aided by our implementation of Gaussian pyramid downsampling, is that no latency in frame processing is introduced, making real-time processing more feasible.

In the Gaussian pyramid generation part, an RGB image is converted to HSV colorspace. The V component given by Eq. 3.4.3 is used as the initial estimation of the illumination image, L.

$$I^{\mathrm{V}}(\boldsymbol{x}) \;=\; \max_{c\in\{\mathrm{R,G,B}\}} I^{c}, \tag{3.4.3}$$

Illumination component is estimated iteratively based on Eq. 3.3.3, by using Eq. 3.4.3 as an initial estimate argument. Considering Eq. (3.3.6), calculation of the dark channel involves minimization over each pixel, over a local patch with transmission $T(x)$ estimated using Eq. 3.3.3, in conjunction with our previously proposed minimization technique in [102].

From the definitions of Eqs. (3.3.3) and (3.3.4), we observe the following useful relationships between image enhancement and haze removal, which aid in the

FPGA realization at a low overhead resource cost.

$$\begin{aligned} i &= \log(I), \ I\text{: input} \\ l &= \log(L), \ L\text{: illumination/transmission} \\ j &= \log(J), \ J\text{: reflectance/haze-free image} \end{aligned}$$

where, in the case of image enhancement,

$$\begin{aligned} J(\boldsymbol{x}) &= \frac{I(\boldsymbol{x}) - A}{T(\boldsymbol{x})} + A, \\ j(\boldsymbol{x}) &= i(\boldsymbol{x}) - l(\boldsymbol{x}) \\ J(\boldsymbol{x}) &= \exp(-l(\boldsymbol{x}))\, I(\boldsymbol{x}), \end{aligned} \tag{3.4.4}$$

and, for haze removal,

$$\begin{aligned} J(\boldsymbol{x}) &= \frac{I(\boldsymbol{x}) - A}{T(\boldsymbol{x})} + A, \\ J(\boldsymbol{x}) &= \exp(-t(\boldsymbol{x}))\, (I(\boldsymbol{x}) - A) + A, \end{aligned} \tag{3.4.5}$$

We formulate a generalized equation from Eq. 3.4.5 by replacing $t(\boldsymbol{x})$ with $l(\boldsymbol{x})$ based on our use of image enhancement for transmission estimation, yielding

$$J(\boldsymbol{x}) = \exp(-l(\boldsymbol{x}))\, (I(\boldsymbol{x}) - A) + A, \tag{3.4.6}$$

Here, it should be noted that Eq. (3.4.4) is a special case of Eq. (3.4.6), where $A = 0$. Furthermore, Eqs. 3.4.4 and 3.4.6 are efficiently suitable expression for our FPGA implementation, as we do not need to perform calculations in logarithmic space. Hence, this limits the requirement for more hardware resources. This architecture takes advantage of these similarities between Retinex-based image enhancement and haze removal, also using Retinex for transmission map estimation instead of soft matting.

In our setup, users can select any of the following operation modes; $i)$ Retinex-based enhancement, $ii)$ haze removal, or $iii)$ in combination. In the image enhancement mode, max operation is used in the Gaussian pyramid generation part, and A is set zero. In the haze removal mode, min operation is used in the Gaussian pyramid generation part, and A is set by user's input. In the combined mode, the use of two parallel in-built circuits is used to perform the max and min initial operations. In single mode, only either is utilized. In addition, these modes are supported by our previously proposed smoothing technique [102], accelerated based on predefined Gaussian pyramid generated downsamples.

Since it can be regarded that A is relatively stable during many successive frames, we do not employ any automatic A estimation. An approximation of A, such as around the maximum value is set manually, based on initial illumination estimation in Eq. 3.4.3. This is relatively sufficient for our approach. However, it should be noted that in some real-time applications such as on-board car cameras, it is necessary to update A regularly using automatic estimation.

Table 3.1: FPGA implementation result (1,920×1,200 and 60 fps)

	Retinex-based image enhancement	Both image enhancement and haze removal
Family	Cyclone V	
Device	5CSXFC6D6F31C6	
Timing Models	Final	
Logic utilization (in ALMs)	3,179/41,910 (8 %)	3,213/41,910 (8 %)
Total registers	3,616	3,648
Total block memory bits	2.71 M/5.67 M (48 %)	2.71 M/5.67 M (48 %)
Total RAM Blocks	366/553 (66 %)	366/553 (66 %)
Total DSP Blocks	16/112 (14 %)	16/112 (14 %)

3.5 Implementation

We implemented the proposed architecture using Intel Cyclone V FPGA, which is from one of the lowest system cost FPGA series. The operating frequency used was 125 MHz, with a 5.67 M total block memory bit size.

Based on Fig. 3.3, we utilized only layer 5, 4, and 3. Table 3.2 shows the optimum number of iterations used per layer in order to obtain desirable results without introducing blur artifacts. Layer 3 is interpolated from layer 4, with layer

Table 3.2: Guassian Pyramid Generation layer size and iterations

	Size (of the input resolution)	# of Iterations
Layer 5	$1/1024$	30
Layer 4	$1/256$	20
Layer 3	$1/64$	10

4 from layer 5. By this, our FPGA implementation supports frame resolution and frame-per-second of 1,920$\times$1,200 and 60 fps, respectively.

The implementation results are summarized in Table 3.1. In Table 3.1, it can be noted that the required number of bits is 2.71 M. This is due to the architecture utilizing only layer 5, 4, and 3 in Fig. 3.3. The advantages of utilizing these layers instead of larger resolution layers 0, 1 or 2 are lower block memory and less iterative complexity, realizing real-time processing at 60 fps.

It was observed that by using layer sizes larger than layer 3, i.e layers 1 and 2, block memory and bits required increased above 453 and 3.28 M, respectively. This also resulted in a decrease in frame rate performance to 45 fps, moving away from our goal of real-time processing at higher frame rate of at least 60 fps, while maintaining operation at relatively low frequency rate. At a frequency of 240 MHz, with a capability of processing 4K video at 30 fps, memory access conflict occured unpredictably. Also taking our hardware memory into consideration, our architecture choice of not using larger layer sizes resulted in better overall performance.

As can be observed from the Table 3.1, both enhancement and haze removal can be implemented using one module with an overall 1 % overhead of logics between single and combined modes, with logic utilization and registers increasing $3179 \rightarrow$

3212 and $3616 \rightarrow 3648$, respectively. The required RAM blocks and memory bits remain constant in both single and combined modes of operation.

Table 3.3: Software performance comparision

Method	Image Size	Average runtime (sec)
[87]	1920 x 1200	307.6
[103]	1920 x 1200	124.8
[75]	1920 x 1200	96
[89]	1920 x 1200	1.651
[84]	1920 x 1200	0.182
Proposed	1920 x 1200	0.165

Table 3.4: Quantitative comparision (PSNR)

Method	street	books	peppers	tower	toys	Average
[103]	16.29	16.57	16.59	18.82	12.56	16.17
[75]	20.12	13.52	29.21	19.32	18.46	20.13
[104] [105]	20.01	15.84	26.73	21.88	17.56	20.40
[84]	19.20	23.59	12.66	23.31	11.42	18.04
Proposed	20.40	21.92	24.78	24.44	18.17	**21.94**

Table 3.5: Quantitative comparision (SSIM)

Method	street	books	peppers	tower	toys	Average
[103]	0.873	0.825	0.858	0.933	0.687	0.835
[75]	0.943	0.881	0.979	0.95	0.809	0.912
[104] [105]	0.943	0.815	0.978	0.972	0.888	0.919
[84]	0.948	0.968	0.790	0.981	0.672	0.872
Proposed	0.898	0.93	0.959	0.98	0.876	**0.929**

In Figs. 3.7 and 3.8, sample results of our proposed FPGA implementation are presented. Table 3.3 shows a software performance comparision of our proposed method with other related methods. Our software implementation was in C++ on Intel(R) Core(TM) $\mathrm{i}5-4460$ CPU $@3.20\,\mathrm{GHz}$. Our approach is competitive, with less simulated processing time. Table 3.6 some hardware performance comparision results. The referenced related works were tested using input feeds at a resolution of 600×400, while our implementation was tested at 1920×1200.

To futher verify our proposed design, we compared its performance with various state-of-the-art dehazing methods. Synthesized hazy images were used, having

Table 3.6: Hardware performance comparision

Method	FPGA	Freq. (MHz)	buffers	Mpixels/s
[89]	stratix	58.43	6	58.43
[106]	stratix	116	6	116
Proposed	Cyclone v	125	6	125

the respective ground truth images. Figure 3.6 presents qualitative results obtained from our proposed method compared with [103], [75], [104, 105] and [84]. As can be observed visually, our proposal is competitive in haze removal performance. The results of Tarel and Hautire [103] have color distortions. The results of Ren *et al*. [84] retain some elements of haze under heavy haze conditions. Berman *et al*. [104, 105] produces better results compared to [75, 84, 103], however with some visible color shift in certain images. In He *et al*. [75], the output images are darker due to the underlaying assumptions used in dark channel prior. Our method, which is a modification based on [75], does not suffer from this darkening characteristic while maintaining high color fidelity. In table 3.4, the PSNR values obtained using the results from Fig. 3.6 are presented. Our method provides an overall better performance based on the average PSNR value of 21.94. Table 3.5 shows the Structural Similarity metrics (SSIM) using the same dataset presented. Our proposed method achieves competitive results are highlighted.

(a) Ground truth (b) haze input (c) [103] (d) [75]

(e) [104] [105] (f) [84] (g) Proposed

Figure 3.6: Results on sythetic haze

(a) Input $I(\boldsymbol{x})$. (b) Transmission $L(\boldsymbol{x})$ (c) Output $J(\boldsymbol{x})$

Figure 3.7: Proposed haze removal on natural haze

(a) Input $I(\boldsymbol{x})$ (b) Illumination $L(\boldsymbol{x})$ (c) Reflectance $J(\boldsymbol{x})$

Figure 3.8: Proposed Retinex image enhancement [2]

3.6 Conclusion

In this segment, a novel architecture supporting both real-time Retinex-based image enhancement and haze removal is proposed, with emphasis on low memory requirement and processing complexity. The FPGA implementation results show that enhancement and haze removal can be implemented using one module, with 1 % logic circuits overhead cost. By utilizing layers 5, 4, and 3 in Fig. 3.3, our proposed architecture supports real-time processing of 1,920×1,200 at 60 fps, under optimal conditions at 125 MHz frequency. By implementing the constraint $i \leq l$, we were able to preserve edges, which otherwise would have suffered from blur effect due to interpolation on smoothed components. Furthermore, by not using an external frame buffer in Fig. 3.2, our proposed FPGA implementation do not suffer from latency in processing real-time feeds. We also used double buffering to prevent memory access conflict. Our design proves to be competitive with state-of-the-art designs, both qualitatively and quantitatively, shown in tables 3.4 and 3.5.

Chapter 4

Conclusion

4.1 Summary and conclusion

Chapter 1 introduces digital image processing techniques. This also outlines the objectives of this thesis. We propose a accelerated image smoothing algorithm, and further investigate and propose real-time FPGA architecture which supports both Retinex-based enhancement and haze removal using a single module.

Chapter 2 focuses on investigating image smoothing. Image smoothing presents a complex minimization problem, which can be tackled in frequency or spatial domain. Frequency domain based solvers provide a faster computation convergence compared to spatial domain solvers. However, in certain conditions FFT solvers introduce wrap around errors about the boundaries of the smoothed image due to the inherent nature of FFT. Spatial domain solvers provide algorithmic control on the way image boundaries are handled. This is investigated and presented in Section 2.5. We successfully implemented image smoothing using spatial iterative methods, with good smoothing quality. It is observed that MGCG has superior performance among the iterative solvers methods demonstrated in the experiments. When applied to a full HD image, MGCG converged in 3.4 secs, while ICCG in 5.7 secs, CG in 9 secs and multigrid in 17.18 secs. Comparing the performance of our proposed approach with existing algorithms shows that our approach converges in competitive relatively less time while not compromising greatly on smoothing quality as the case of others. We successfully investigated two implementation optimization techniques to further accelerate our proposed method.

The first was using image downsampling in order to reduce the resolution required to processing in smoothing algorithm. By introducing downsampling into the algorithm, we managed to reduce the processing time by approximately 21.6 %. In addition to processing time reduction, the use of downsampled images on

which to perform image smoothing reduces the memory requirement. However, a trade-off between processing time and smoothing quality is introduced. By using simple downsampling algorithms, processing time advantage is gain at the expense of smoothing quality. The use of more complex scaling algorithms would improve the quality with less processing time gain.

The second approach involved the pre-calculation and storage of the computationally costly Laplacian operator matrix, which depends only on the resolution of the input. This is applicable to constant resolution input streams, extending its application to video processing. By pre-calculating and storing this operator only once, inputing images having the same resolution results in a processing time reduction of approximately 46.1 %. The choice of which optimization technique depends on the application parameters and hardware resource availability.

Iterative solvers provide the user greater degree of flexibility, as various components which affect processing time can be tweaked to achieve an application specific desired outcome. It can be concluded that iterative methods are scalable and flexible.

In Chapter 3, a novel architecture supporting both real-time Retinex-based image enhancement and haze removal is proposed, with emphasis on low memory requirement and processing complexity. The FPGA implementation results show that enhancement and haze removal can be implemented using one module, with 1 % logic circuits overhead cost. By utilizing Gaussian pyramid generation, our proposed architecture supports real-time processing of 1,920×1,200 at 60 fps, under optimal conditions at 125 MHz frequency. At frequency of 240 MHz, our architecture is capable of processing 4K video at 30 fps. By implementing the constraint $i \leq l$, we were able to preserve edges, which otherwise would have suffered from blur effect due to interpolation on smoothed components. Furthermore, by not using an external frame buffer in, our proposed FPGA implementation do not suffer from latency in processing real-time feeds. According to experimental results obtained, by evaluating PSNR and SSIM, our approach yields better results compared to existing state-of-the-art algorithms. Comparing hardware performance with existing architectures, our architecture provides the highest throughput of 125 Mpixels/s, utilizing 9 line buffers for filtering with a width of 240.

In conclusion, the objectives of our research to investigate and develop an accelerated image smoothing algorithm, and to design an FPGA architecture capable to support both Retinex-based enhancement and haze removal have been completed successfully. We have evaluated our designs against existing designs, yielding favorable quantitative and qualitative results.

Bibliography

[1] H. Badri, H. Yahia, and D. Aboutajdine, "Fast multi-scale detail decomposition via accelerated iterative shrinkage," in *SIGGRAPH Asia 2013*, 2013.

[2] H. Tsutsui, H. Nakamura, R. Hashimoto, H. Okuhata, and T. Onoye, "An FPGA implementation of real-time Retinex video image enhancement," in *Proceedings of World Automation Congress*, Sep. 2010, pp. 1–6.

[3] L. Xu, C. Lu, Y. Xu, and J. Jia, "Image smoothing via l0 gradient minimization," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 174:1–174:12, Dec. 2011.

[4] Z. He, Y. Zhang, R. Su, and S. Fan, "A novel image smoothing algorithm based on variational decomposition," in *Genetic and Evolutionary Computing (ICGEC), 2010 Fourth International Conference on*. IEEE, 2010, pp. 181–185.

[5] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proceedings of the Sixth International Conference on Computer Vision*, ser. ICCV '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 839–.

[6] J. Shen, S. Fang, H. Zhao, X. Jin, and H. Sun, "Fast approximation of trilateral filter for tone mapping using a signal processing approach," *Signal Processing*, vol. 89, pp. 901–907, 05 2009.

[7] B. Weiss, "Fast median and bilateral filtering," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 519–526, Jul. 2006. [Online]. Available: http://doi.acm.org/10.1145/1141911.1141918

[8] ——, "Fast median and bilateral filtering," in *ACM SIGGRAPH 2006 Papers*, ser. SIGGRAPH '06. New York, NY, USA: ACM, 2006, pp. 519–526. [Online]. Available: http://doi.acm.org/10.1145/1179352.1141918

[9] J. Chen, S. Paris, and F. Durand, "Real-time edge-aware image processing with the bilateral grid," in *ACM SIGGRAPH 2007 Papers*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1275808.1276506

[10] ——, "Real-time edge-aware image processing with the bilateral grid," *ACM Trans. Graph.*, vol. 26, no. 3, Jul. 2007. [Online]. Available: http://doi.acm.org/10.1145/1276377.1276506

[11] P. Choudhury and J. Tumblin, "The trilateral filter for high contrast images and meshes." 01 2003, pp. 186–196.

[12] J. Baek and D. Jacobs, "Accelerating spatially varying gaussian filters," *ACM Trans. Graph.*, vol. 29, p. 169, 12 2010.

[13] M. Kass and J. Solomon, "Smoothed local histogram filters," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 100:1–100:10, Jul. 2010. [Online]. Available: http://doi.acm.org/10.1145/1778765.1778837

[14] ——, "Smoothed local histogram filters," in *ACM SIGGRAPH 2010 Papers*, ser. SIGGRAPH '10. New York, NY, USA: ACM, 2010, pp. 100:1–100:10. [Online]. Available: http://doi.acm.org/10.1145/1833349.1778837

[15] K. Subr, C. Soler, and F. Durand, "Edge-preserving multiscale image decomposition based on local extrema," *ACM Transactions on Graphics*, vol. 28, 12 2009.

[16] D. Min, S. Choi, J. Lu, B. Ham, K. Sohn, and M. Do, "Fast global image smoothing based on weighted least squares," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 23, 10 2014.

[17] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 7, pp. 629–639, Jul. 1990. [Online]. Available: https://doi.org/10.1109/34.56205

[18] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '02. New York, NY, USA: ACM, 2002, pp. 257–266. [Online]. Available: http://doi.acm.org/10.1145/566570.566574

[19] ——, "Fast bilateral filtering for the display of high-dynamic-range images," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 257–266, Jul. 2002. [Online]. Available: http://doi.acm.org/10.1145/566654.566574

[20] F. Porikli, "Constant time o(1) bilateral filtering," 06 2008.

[21] Q. Yang, K.-H. Tan, and N. Ahuja, "Ahuja n.: Real-time o(1) bilateral filtering," vol. 1, 06 2009, pp. 557–564.

[22] A. Adams, J. Baek, and M. Abraham Davis, "Fast high-dimensional filtering using the permutohedral lattice," *Comput. Graph. Forum*, vol. 29, pp. 753–762, 05 2010.

[23] K. He, J. Sun, and X. Tang, "Guided image filtering," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, pp. 1397–1409, 06 2013.

[24] Q. Yang, "Recursive bilateral filtering," in *Proceedings of the 12th European Conference on Computer Vision - Volume Part I*, ser. ECCV'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 399–413. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33718-5_29

[25] J. Lu, K. Shi, D. Min, L. Lin, and M. do, "Cross-based local multipoint filtering," 06 2012, pp. 430–437.

[26] R. Fattal, "Edge-avoiding wavelets and their applications," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 22:1–22:10, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1531326.1531328

[27] ——, "Edge-avoiding wavelets and their applications," in *ACM SIGGRAPH 2009 Papers*, ser. SIGGRAPH '09. New York, NY, USA: ACM, 2009, pp. 22:1–22:10. [Online]. Available: http://doi.acm.org/10.1145/1576246.1531328

[28] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using optimization," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 689–694, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1015706.1015780

[29] ——, "Colorization using optimization," in *ACM SIGGRAPH 2004 Papers*, ser. SIGGRAPH '04. New York, NY, USA: ACM, 2004, pp. 689–694. [Online]. Available: http://doi.acm.org/10.1145/1186562.1015780

[30] D. Lischinski, Z. Farbman, M. Uyttendaele, and R. Szeliski, "Interactive local adjustment of tonal values," *ACM Trans. Graph.*, vol. 25, pp. 646–653, 07 2006.

[31] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski, "Edge-preserving decompositions for multi-scale tone and detail manipulation," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 67:1–67:10, Aug. 2008.

[32] P. Bhat, C. Lawrence Zitnick, M. Cohen, and B. Curless, "Gradientshop: A gradient-domain optimization framework for image and video filtering," *ACM Trans. Graph.*, vol. 29, 01 2010.

[33] L. Xu, Q. Yan, Y. Xia, and J. Jia, "Structure extraction from texture via relative total variation," *ACM Transactions on Graphics (TOG)*, vol. 31, 11 2012.

[34] I. Koutis, G. Miller, and D. Tolliver, "Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing," vol. 115, 01 1970, pp. 1067–1078.

[35] D. Krishnan, R. Fattal, and R. Szeliski, "Efficient preconditioning of laplacian matrices for computer graphics," *ACM Transactions on Graphics (TOG)*, vol. 32, 07 2013.

[36] G. Tanaka, N. Suetake, and E. Uchino, "Image enhancement based on nonlinear smoothing and sharpening for noisy images," *Journal of Advanced Computational Intelligence and Intelligent Informatics (JACIII)*, vol. 14, no. 200207, 2010.

[37] D. G. Lowe, "Organization of smooth image curves at multiple scales," in *International Conference on Computer Vision*, 1988, pp. 558–567.

[38] D. Barash, "A fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 6, pp. 844–847, Jun. 2002.

[39] D. Min, S. Choi, J. Lu, B. Ham, K. Sohn, and M. Do, "Fast global image smoothing based on weighted least squares," 2014.

[40] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," *Int. J. Comput. Vision*, vol. 81, no. 1, pp. 24–52, Jan. 2009.

[41] W. Liu, Y. Duan, K. Shao, and L. Zhang, "Image smoothing based on the mean shift algorithm," in *Control and Automation, 2007. ICCA 2007. IEEE International Conference on*, May 2007, pp. 1349–1353.

[42] L. Xu, C. Lu, Y. Xu, and J. Jia, "Image smoothing via l0 gradient minimization," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 174:1–174:12, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2070781.2024208

[43] ——, "Image smoothing via l0 gradient minimization," in *Proceedings of the 2011 SIGGRAPH Asia Conference*, ser. SA '11. New York, NY, USA: ACM, 2011, pp. 174:1–174:12. [Online]. Available: http://doi.acm.org/10.1145/2024156.2024208

[44] N. Parikh and S. Boyd, "Proximal algorithms," *Found. Trends Optim.*, vol. 1, no. 3, pp. 127–239, Jan. 2014.

[45] P. Bhat, B. Curless, M. Cohen, and C. L. Zitnick, "Fourier analysis of the 2D screened Poisson equation for gradient domain problems," in *Proceedings of the 10th European Conference on Computer Vision: Part II*, ser. ECCV '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 114–128.

[46] Z. Shangdong, A. Zhibin, and C. Xiyuan, "On color image smoothing processing based an color space transformation," in *International Vehicle Electronics Conference (IVEC)*, vol. 1, Sep. 1999, pp. 271–274.

[47] D. Kasauka, H. Tsutsui, H. Okuhata, and Y. Miyanaga, "Computational cost analysis and implementation of accelerated iterative shrinkage smoothing." in *Asia-Pacific Signal and Information Processing Association, 2014 Annual Summit and Conference (APSIPA)*, 2014, pp. 1–4.

[48] *1. Matrices*, ch. 1, pp. 1–22.

[49] R. Bellman, *Introduction to Matrix Analysis (2Nd Ed.)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997.

[50] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[51] K. Nakajima, "Parallel iterative linear solvers with preconditioning for large scale problems," Ph.D. dissertation, Tokyo University, 2002.

[52] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. Philadelphia, PA: SIAM, 1994.

[53] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003.

[54] R. D. Falgout, "An introduction to algebraic multigrid," *Computing in Science and Engg.*, vol. 8, no. 6, pp. 24–33, Nov. 2006.

[55] D. Krishnan and R. Szeliski, "Multigrid and multilevel preconditioners for computational photography," New York University, Tech. Rep., 2011.

[56] G. Strang, "Mathematical methods for engineers ii," MIT OpenCourseWare, 2006.

[57] O. Tatebe, "Mgcg method: A robust and highly parallel iterative method," Ph.D. dissertation, University of Tokyo, 1996.

[58] O. Tatebe and Y. Oyanagi, "Efficient implementation of the multigrid preconditioned conjugate gradient method on distributed memory machines," in *Proceedings of Super computing 94*. IEEE Computing Society, 1994, pp. 194–203.

[59] A. Gupta, "Preconditioners for sparse iterative methods," in *Encyclopedia of Parallel Computing*, 2011, pp. 1615–1624.

[60] D. Lahaye and S. Vandewalle, "An algebraic multigrid method for solving very large electromagnetic systems," *IEEE Trans. Magn*, vol. 34, no. 5, pp. 3327–3330, 1998.

[61] U. Trottenberg and A. Schuller, *Multigrid*. Orlando, FL, USA: Academic Press, Inc., 2001.

[62] K. Stuben, "An introduction to algebraic multigrid," *Multigrid*, pp. 413–532, 2001.

[63] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Pittsburgh, PA, USA, Tech. Rep., 1994.

[64] G. H. Golub and C. F. Van Loan, *Matrix Computations (3rd Ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[65] Y. Robert, "Regular incomplete factorizations of real positive definite matrices," *Linear Algebra and its Applications*, vol. 48, pp. 105–117, 1982.

[66] X. Wang, "Incomplete factorization preconditioning for linear least squares problems," Ph.D. dissertation, Graduate college of the University of Illinois, 1994.

[67] J. H. Yun and Y. D. Han, "Modified incomplete cholesky factorization preconditioners for a symmetric positive definite matrix," *Bull. Korean Math. Soc. 39*, no. 3, pp. 495–509, 2002.

[68] K. R. Kennedy, "Implementing conjugate gradients with incomplete cholesky preconditioning in playa," Master's thesis, Graduate Faculty of Texas Tech University, 2012.

[69] O. Tatebe, "The multigrid preconditioned conjugate gradient method," in *Proceedings of Sixth Copper Mountain Conference on Multigrid Methods*. NASA Conference Publication 3224, 1993, pp. 621–634.

[70] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, pp. 259–268, 11 1992.

[71] L. Bao, Y. Song, Q. Yang, H. Yuan, and G. Wang, "Tree filtering: Efficient structure-preserving smoothing with a minimum spanning tree," *Image Processing, IEEE Transactions on*, vol. 23, pp. 555–569, 02 2014.

[72] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[73] J. McCann, "Lessons learned from mondrians applied to real images and color gamuts." vol. 14, pp. 1–8, Jan. 1999.

[74] D. J. Jobson, Z.-U. Rahman, and G. A. Woodell, "A multi-scale Retinex for bridging the gap between color images and the human observation of scenes," vol. 6, pp. 965–976, Aug. 1997.

[75] K. He, J. Sun, and X. Tang, "Single image haze removal using dark channel prior," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 12, pp. 2341–2353, Dec. 2011.

[76] Y. Wang and B. Wu, "Improved single image dehazing using dark channel prior," in *Proceedings of Intelligent Computing and Intelligent Systems (ICIS)*, vol. 2, Oct. 2010, pp. 789–792.

[77] R. Tan, N. Pettersson, and L. Petersson, "Visibility enhancement for roads with foggy or hazy scenes," 07 2007, pp. 19 – 24.

[78] N. Hautière, J.-P. Tarel, and D. Aubert, "Towards fog-free in-vehicle vision systems through contrast restoration," 06 2007.

[79] L. Caraffa and J.-P. Tarel, "Markov random field model for single image defogging," 06 2013, pp. 994–999.

[80] S.-C. Pei and T.-Y. Lee, "Nighttime haze removal using color transfer pre-processing and dark channel prior," 09 2012, pp. 957–960.

[81] A. Codruta and C. Ancuti, "Single image dehazing by multi-scale fusion," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 22, 05 2013.

[82] R. T. Tan, "Visibility in bad weather from a single image," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2008, pp. 598–605.

[83] Q. Zhu, J. Mai, and L. Shao, "A fast single image haze removal algorithm using color attenuation prior," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 24, 06 2015.

[84] W. Ren, S. Liu, H. Zhang, J. Pan, X. Cao, and M.-H. Yang, "Single image dehazing via multi-scale convolutional neural networks," *European conference on computer vision*, pp. 154–169, 2016.

[85] C.-T. Shen and W.-L. Hwang, "Color image enhancement using Retinex with robust envelope," in *2009 16th IEEE International Conference on Image Processing (ICIP)*, Nov 2009, pp. 3141–3144.

[86] F. Guo, Z. Cai, B. Xie, and J. Tang, "Automatic image haze removal based on luminance component," in *Proceedings of International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)*, Oct. 2010, pp. 1–4.

[87] R. Fattal, "Single image dehazing," in *Proceedings of ACM SIGGRAPH*, 2008.

[88] S. Marsi and G. Ramponi, "A flexible FPGA implementation for illuminance–reflectance video enhancement," *Journal of Real-time Image Processing*, vol. 8, May 2011.

[89] Y. Shiau, H. Yang, P. Chen, and Y. Chuang, "Hardware implementation of a fast and efficient haze removal method," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, no. 8, pp. 1369–1374, Aug 2013.

[90] H. Zhang and V. M. Patel, "Densely connected pyramid dehazing network," in *CVPR*, 2018.

[91] A. Galdran, A. Alvarez-Gila, A. Bria, J. Vazquez-Corral, and M. Bertalmio, "On the duality between retinex and image dehazing," in *CVPR 2018*, Salt Lake City, USA, 2018.

[92] E. H. Land and J. McCann, "Lightness and Retinex theory," *Journal of the Optical Society of America*, vol. 61, pp. 1–11, 02 1971.

[93] R. Kimmel, M. Elad, D. Shaked, R. Keshet, and I. Sobel, "A variational framework for Retinex," *Int. J. Comput. Vision*, vol. 52, no. 1, pp. 7–23, Apr. 2003.

[94] W.-C. Zhang, X.-J. An, and S.-D. Pan, "An improved recursive Retinex for rendering high dynamic range photographs," pp. 121–125, Jul. 2011.

[95] D.-G. Hwang, W.-R. Lee, Y.-J. Oh, and B.-M. Jun, "Frankle-mccann Retinex by shuffling," vol. 7425, pp. 381–388, Aug. 2012.

[96] G. Hou, G. Wang, Z. Pan, B. Huang, H. Yang, and T. Yu, "Image enhancement and restoration: State of the art of variational Retinex models," vol. 44, pp. 445–455, Nov. 2017.

[97] D. J. Jobson, Z. Rahman, and G. A. Woodell, "Properties and performance of a center/surround Retinex," vol. 6, pp. 451–62, Feb. 1997.

[98] H. Tsutsui, S. Yoshikawa, H. Okuhata, and T. Onoye, "Halo artifacts reduction method for variational based realtime Retinex image enhancement," in *Proceedings of The 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference*, Dec 2012, pp. 1–6.

[99] S. G. Narasimhan and S. K. Nayar, "Vision and the atmosphere," *International Journal of Computer Vision*, vol. 48, no. 3, pp. 233–254, 2002.

[100] ——, "Chromatic framework for vision in bad weather," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2000, pp. 598–605.

[101] Y. Ru and G. Tanaka, "Proposal of dehazing method and quantitative index for evaluation of haze removal quality," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100.A, no. 4, pp. 1045–1054, 2017.

[102] D. Kasauka, H. Tsutsui, S. Imai, T. Imagawa, H. Okuhata, and Y. Miyanaga, "Image smoothing in the spatial domain using multigrid conjugate gradient methods based on accelerated iterative shrinkage," in *Proc. Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, Dec. 2016, pp. 1–5.

[103] J. Tarel and N. Hautière, "Fast visibility restoration from a single color or gray level image," in *2009 IEEE 12th International Conference on Computer Vision*, Sept 2009, pp. 2201–2208.

[104] D. Berman, T. Treibitz, and S. Avidan, "Non-local image dehazing," in *IEEE Conf. CVPR*, 2016.

[105] ——, "Air-light estimation using haze-lines," in *IEEE Conf. ICCP*, 2017.

[106] B. Zhang and J. Zhao, "Hardware implementation for real-time haze removal," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–5, 2016.

Copyright notice

Dabwitso Kasauka, Kenta Sugiyama, Hiroshi Tsutsui, Hiroyuki Okuhata, Yoshikazu Miyanaga, “An Architecture for Real-time Retinex-based Image Enhancement and Haze Removal and its FPGA Implementation,” IEICE Transaction Fundamentals of Electronics, Communications and Computer Sciences, June 2019.

Dabwitso Kasauka, Hiroshi Tsutsui, Seijiro Imai, Takashi Imagawa, Hiroyuki Okuhata, Yoshikazu Miyanaga, ”Image Smoothing in the Spatial Domain Using Multigrid Conjugate Gradient Methods Based on Accelerated Iterative Shrinkage,” Proceedings of Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pp. 1-5, Dec. 2016.

Dabwitso Kasauka, Hiroshi Tsutsui, Hiroyuki Okuhata, Takashi Imagawa, Yoshikazu Miyanaga, “Image Smoothing Using Spatial Iterative Methods Based on Accelerated Iterative Shrinkage,” Proceedings of Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pp. 779-783, 16-19 Dec. 2015.

Dabwitso Kasauka, Hiroshi Tsutsui, Hiroyuki Okuhata, Yoshikazu Miyanaga, ”Computational Cost Analysis and Implementation of Accelerated Iterative Shrinkage Smoothing,” Proceedings of Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pp. 1-4, 9-12 Dec. 2014.