

**Bayesian Ultrasound Image Analysis on Graphics
Hardware**

by

Qi Wei

B.E., Beijing Institute of Technology, 2001

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

December 2003

© Qi Wei, 2003

Abstract

In this thesis, we investigate using the new generation of programmable *Graphics Processing Units* (GPUs) which support floating point computations for statistical restoration of ultrasound images. Deconvolution is a widely used method of recovering an image from degradation caused by blurring, and thus increasing the image quality. We present a modified Bayesian 2D deconvolution method which provides better parameter estimation and improves the speed performance over the previous approach. This method lies within the *Joint Maximum A Posteriori* (JMAP) framework and involves three steps. First is the Point Spread Function (PSF) estimation, which uses the Homomorphic method; second, reflectance field estimation uses a *Conjugate Gradient* (CG) optimization algorithm; and third, variance field estimation uses a *Markov Chain Monte Carlo* (MCMC) sampling algorithm.

We implement the 2D method entirely on programmable floating-point graphics hardware, and results are achieved at an interactive rate. In order to avoid read-back from GPU to CPU, we adopt a multi-pass rendering method to realize the iterative model. This indicates the possibility of using the GPU as a coprocessor in the ultrasound imaging system, to improve image quality in real time. Due to the special architecture of GPUs, not all models are suitable for mapping onto them. We also discuss which structures and schemes GPUs favor and which they do not. Experimental results are presented on synthetic and real ultrasound images acquired by a typical diagnostic ultrasound machine.

We believe our research opens the door for many other image processing methods that are otherwise currently impractical, due to time consuming and complicated computations. This is especially important for medical image processing applications.

Contents

Abstract	ii
Contents	iii
List of Tables	vii
List of Figures	ix
Acknowledgements	xv
Dedication	xvi
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Outline	5
2 Background and Related Work	6
2.1 Ultrasound Image Analysis	6
2.1.1 Ultrasound Imaging	6
2.1.2 Ultrasound Image Degradation Model	7
2.1.3 Ultrasound Point Spread Functions	8
2.1.4 Ultrasound Image Restoration	9
2.1.5 Previous Work	10
2.2 Programmable Graphics Processing Unit (GPU)	12

2.2.1	Modern Graphics Architectures	12
2.2.2	GPU Programming Languages	14
2.2.3	Evolution on GPUs - High Precision	15
2.2.4	Previous Work	15
2.2.5	Recent Related Work	17
3	Bayesian 2D Deconvolution Model and Acceleration on CPUs	18
3.1	Statistical Ultrasound Speckle Models	18
3.2	Standard Model Description	21
3.2.1	Introduction	21
3.2.2	Model Derivation	22
3.2.3	Assumptions on PSF	25
3.2.4	Sampling the Reflectance Field	25
3.2.5	Sampling the Variance Field	26
3.3	Improved Model Based on CPU Architecture	28
3.3.1	Ensuring Good Data Locality and Caching Behavior	28
3.3.2	Partially Spatially Variant PSFs	29
3.3.3	Inter-frame Coherence	30
4	Ultrasound Image Restoration within JMAP Framework	32
4.1	Motivation for Modified Model	32
4.2	MAP Estimate Methods Exploration	34
4.2.1	Deterministic Methods	34
4.2.2	Stochastic Methods	34
4.3	JMAP Ultrasound Image Restoration Model	35
4.3.1	Description of Model Elements	35
4.3.2	JMAP Model Description	37
4.4	Solutions to Three Steps	39
4.4.1	PSF Estimation using Homomorphic Transform	39

4.4.2	Reflectance Estimation using CG Method	44
4.4.3	Variance Estimation using MCMC	47
4.5	JMAP Model Discussion	50
4.5.1	Modification Aspects	50
4.5.2	Parameter Configuration	51
4.5.3	EM Algorithms	51
5	Implementation on the GPU	52
5.1	Streaming Implementation Overview	52
5.2	Solutions for Efficient Mapping on GPU	54
5.2.1	Fast Median Search	54
5.2.2	Four Color Packing	56
5.2.3	Fast Convolution with Large Kernels	60
5.2.4	Random Number Generation	67
5.2.5	Two Passes Updating a Markov Random Field	68
5.2.6	Display Unpacked Image	70
5.2.7	Boundary Padding	71
5.3	Summary and Discussion	72
5.3.1	Shader Summary	72
5.3.2	Current Limitations	74
5.3.3	Algorithm Mapping Discussion	75
6	Experiments	78
6.1	Experiments on Synthetic Images	78
6.1.1	Synthetic Image Generation	78
6.1.2	Estimated Point Spread Function	83
6.1.3	Speed Performance	83
6.1.4	Resolution Performance of Estimated Reflectance Field	86
6.1.5	Resolution Performance of Estimated Variance Field	86

6.2	Experiments on Real Image Data	91
6.2.1	Experiment Design	91
6.2.2	Phantom Construction	92
6.2.3	Results on Cylinder Phantom	93
6.2.4	Results of the Cube Phantom	101
7	Conclusions	105
8	Future Work	107
8.1	Extension to General Image Restoration	107
8.2	Nonnegative Constraints	107
8.3	Parameter Configuration	108
8.4	Deconvolution of Astronomical Images at Iterative Rates	108
	Bibliography	109

List of Tables

3.1	Running time (second/iteration) comparison with and without partition.	29
5.1	Minimum numbers of operations to find the median in short sequences, from [1].	56
5.2	FFT performance, from [2]. The program transforms an image into the frequency domain using FFT; then a low-pass filtering is then performed there. Finally, it is transformed back to the time domain using IFFT.	60
5.3	Number of instructions and registers for convolution with different kernels. These numbers are generated by the Cg compiler.	64
5.4	Instruction and register counts for the four pass convolution fragment programs with kernel size 21×21	65
5.5	Convolution running time (in milliseconds) comparison. The target image has 128×128 pixels. The numbers given in <i>Packed A</i> , <i>Packed B</i> , and <i>Packed C</i> columns are experimental results on an NVIDIA FX5600 card, an NVIDIA FX 5800 card, and an NVIDIA Quadro3000 card, respectively. Here <i>N/A</i> means these operations cannot possibly be implemented either because they are not supported (for the OpenGL column) or because of the 1024 maximum instruction limitation on fragment programs.	66

5.6	Shader summary. The complexity for each fragment program is acquired from the Cg compiler and shown here, complementing the texture-shader relationship tables. All the images are rendered with 128×128 pixels.	74
6.1	Running time (in milliseconds) for each step for estimating r	84
6.2	Running time (in milliseconds) for each step for estimating σ^2	84
6.3	Running time (in milliseconds) for each stage and also the total time of each iteration.	84
6.4	Recipe for the phantom medium, from [3].	93

List of Figures

1.1	(a) A raw-data ultrasound image scanned from the cylinder phantom with four fish wires. (b) Phantom picture captured by a digital camera.	2
2.1	A typical graphics pipeline based on the OpenGL API. Arrows represent data streams.	12
2.2	Programmable graphics pipeline. Arrows represent data streams. The application and the two programmable processors are connected by lines, which means the application has control of them. Parameters can be transferred between the application and the processors.	13
3.1	Pseudocode generating a normal random number using the <i>Box-Müller</i> method, from [4].	26
3.2	Independent Metropolis-Hastings algorithm, from [5].	27
3.3	Proposal function on $\sigma_{i,j}^2$, from [6].	27
3.4	Two real images, namely Image1 (on the left) and Image2 (on the right) respectively, in a sequence. They are both of resolution 64×64 .	31
3.5	Convergence acceleration by inter-frame knowledge.	31

4.1	Graphical Model of statistical ultrasound image restoration. Fixed hyper-hyperparameters are represented with <i>boxes</i> . Observed variable is represented with a <i>double circle</i> . Unknown variables are represented with <i>circles</i> . Model parameters, but not hyperparameters, are represented in <i>double boxes</i>	36
4.2	Plot of the prior density function in Equation 4.3.	37
4.3	Flowchart estimating the PSF using cepstrum transforms. The input signal, which is the observed raw image, is represented with a <i>circle</i> . The output, which is the estimated PSF signal, is represented with a <i>double circle</i> . All the other intermediate variables are in <i>ovals</i> . <i>Boxes</i> represent major processing steps, each of which consists of many operations. <i>Arrows</i> represent data flow and the <i>operators</i> bound with them indicate the operations performed on the data.	43
4.4	Pseudocode of the Conjugate Gradient (CG) method, from [7]. . . .	47
5.1	Abstract structure for CPU image processing programs.	52
5.2	Extended fastest C code to search the median of four elements, from [1].	55
5.3	Fragment program code to search the median of four elements. . . .	56
5.4	Color packing by four blocks of the same size. The resulting image has the same size as each block.	57
5.5	Color packing by 4 adjacent neighbors.	58
5.6	Color packing by vectors. The matrix on the left is packed along the row direction and the matrix on the right is packed along the column direction. Each packed fragment is shown in one black solid box. . . .	59
5.7	Four color components in one pixel, represented by four colorful <i>Xs</i> , have different neighbors.	61
5.8	Four PSF textures with proper zero paddings. Here we use a 5×5 kernel for simple illustration.	62

5.9	Left: red PSF texture is color-packed as well. Right: extended PSF texture helps to eliminate conditional checking inside each fragment of which ones are neighbors, and therefore, four colors can be fetched together. Notice the matching of the red PSF texture on the left with the extended supporting region on the right, represented by the solid, rounded rectangle.	63
5.10	Plot on the convolution time comparison in Table 5.5.	66
5.11	Random number accessing on a GPU. The background array is the random2Texture and the random numbers in the shadowed region are fetched and used in the fragment programs.	67
5.12	First-order Markov neighborhood system.	69
5.13	Second-order Markov neighborhood system.	69
5.14	Parallel updating within first-order Markov neighborhood system.	70
5.15	Textures and fragment programs in the reflectance estimation initialization step. The textures are specified by the capital letters on the first row. The operations in the fragment programs are given in the left column.	72
5.16	Textures and programs in the reflectance estimation step.	73
5.17	Textures and programs in the variance σ^2 estimation step.	74
5.18	Restructured MCMC model on GPUs.	77
6.1	Simulated variance field. (a) Image with original intensity. (b) Image of the scaled intensity by a factor of four. (c) Image rescaled using the maximum and minimum intensity values.	79
6.2	Simulated reflectance field. (a) Image with original intensity. (b) Image rescaled using the maximum and minimum intensity values.	79
6.3	Simulated point spread function.	80
6.4	Simulated ultrasound image. (a) Image with original intensity. (b) Image rescaled using the maximum and minimum intensity values.	81

6.5	Comparisons of the pixels along (a) the 43 rd scanline, and (b) the 26 th scanline in different stages.	82
6.6	Estimated PSF from Figure 6.4.	83
6.7	Convergence plots on (a) r and (b) σ^2	85
6.8	Estimated reflectance field. (a) Image with original intensity. (b) Image rescaled using the maximum and minimum intensity values. .	86
6.9	Comparisons on auto-covariance plots on unprocessed image envelope and deconvolved image along (a) the axial direction, and (b) the lateral direction. The solid curves show the auto-covariances on the original image and the dashed curves show the results after deconvolution. The peak value of auto-covariance is normalized to 1.0. . . .	87
6.10	Estimated variance field after 3000 iterations. (a) Image with original intensity. (b) Image of the scaled intensity by a factor of three. (c) Image rescaled using the maximum and minimum intensity values. .	88
6.11	Estimated variance field after (a) 500 iterations; (b) 1000 iterations; (c) 2000 iterations and (d) 6000 iterations. (e), (f), (g), and (h) are the corresponding scaled images, by a factor of three.	88
6.12	Pixels comparison on the 91 st rows in the original variance field, blurred image, and estimated variance field, respectively.	89
6.13	Pixels comparison on the 99 th rows in the original variance field, blurred image, and estimated variance field, respectively.	90
6.14	Pixels comparison on the 108 th rows in the original variance field, blurred image, and estimated variance field, respectively.	90
6.15	Design illustration of the two phantoms. These are the side views. The inclusions, a cylinder in (a) and a cube in (b), are of the same physical properties.	92

6.16	(a) Image of phantom A shown on the screen. This is the image interpolated from the raw signals shown in Figure 6.17. (b) Phantom picture captured by a digital camera.	94
6.17	Ultrasound raw-data images scanned from the cylinder phantom. They are of resolution 234×512 . (a) Original image. (b) Image in which the known inclusion is marked by a black oval.	95
6.18	(a) Raw-data ultrasound image scanned from the cylinder phantom with four fish wires. It is of resolution 234×512 . (b) Phantom picture captured by a digital camera. (c) Camera-captured picture with the object marked by a cycle. (d) The same raw-data ultrasound image as in (a) with the marked object.	96
6.19	(a) A window is specified on the original raw-data image which embraces a cylindrical object. In order to fit the page, the image is resized. (b) A 128×128 small region which corresponds to the window in (a) is extracted from Figure 6.17. (c) This is the same image as in (b) but the object is marked by an oval.	98
6.20	Point spread function estimated from Figure 6.19(b).	98
6.21	Reflectance field estimated from Figure 6.19(b).	99
6.22	Estimated variance fields from Figure 6.19(b) after (a) 500 iterations; (c) 1500 iterations; (e) 3000 iterations; (g) 5000 iterations; (i) 5000 iterations; (k) 6000 iterations and (m) 7000 iterations. The known location and shape of the target feature are marked with a black oval in the corresponding Figures (b), (d), (f), (h), (j), (l), and (n).	100
6.23	(a) An interpolated ultrasound image scanned from the cube phantom. (b) Phantom picture acquired from a digital camera.	101
6.24	(a) Raw-data ultrasound image scanned from the phantom that has a cubic object inside. (b) The same image with object marked by a box.	102

6.25	(a) An object region in the raw-data image has 128×128 pixels. (b) The same image with object marked by a box.	102
6.26	Estimated variance field from Figure 6.25 after (a) 500 iterations; (c) 1500 iterations; (e) 3000 iterations; (g) 5000 iterations; (i) 5000 iterations; (k) 6000 iterations and (m) 7000 iterations. The known location and shape of the target feature is marked with a black box in the corresponding (b), (d), (f), (h), (j), (l), and (n).	104

To my parents Jianping Wei and Yuqing Peng

Chapter 1

Introduction

1.1 Motivation

While diagnostic ultrasound has been used for more than fifty years, it has become one of the most popular medical diagnostic tools in recent years. It has the advantages of safety, low cost and interactivity, compared to X-rays, computed tomography (CT) and magnetic resonance imaging (MRI). Of the multiple modes it provides, *B-mode* scanning, which is also named *brightness mode*, displays images on screen with bright dots representing the ultrasound echoes. The intensity of the dots is determined by the echo amplitude. However, it is well-known that B-mode ultrasound images usually suffer from speckle and noise, which greatly reduce their diagnostic value, and hence, restrict their clinical usefulness in some instances.

Figure 6.18(a)¹ shows an ultrasound image scanned from the phantom, shown in Figure 6.18(b)². It is acquired by a typical ultrasound imaging system. It can be seen that the cylindrical object is not very clear in the ultrasound image because of the speckle and noise.

In the past dozens of years, significant efforts have been made to improve ultrasound image quality and look for better tissue information representations.

¹This image is also used in the experiments in Chapter 6.

²This image is also used in the experiments in Chapter 6.

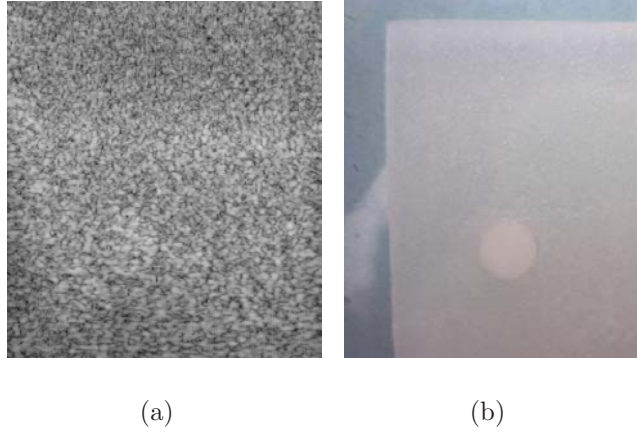


Figure 1.1: (a) A raw-data ultrasound image scanned from the cylinder phantom with four fish wires. (b) Phantom picture captured by a digital camera.

However, due to the complicated interaction introduced by the pulse-echo process, ultrasound image analysis has presented many more difficulties than ordinary image analysis. Simple models such as filtering usually cannot produce satisfying results. Sophisticated models have been developed as well, trying to model the complex process of image formation. However, they have traditionally been very slow, due to the huge amount of computation required. This has seriously inhibited their application. Therefore, developing reliable real time ultrasound image restoration and analysis methods that can better present tissue information is still a challenge to the medical image processing community and the diagnostic ultrasound industry.

Bayesian statistics provide some very powerful tools for image analysis. They are capable of modelling the uncertainty and randomness that commonly exist in complicated image formation and degradation processes. An effective approach for estimating both the reflectance field and the variance field from ultrasound images is proposed in [6]. However, due to the inherent high dimensionality of the data, this takes longer than desired; [6] reports taking 50 minutes to process one image. Resolution performance and speed performance are always a tradeoff. The question is, what is the best way to achieve the best performance with the least cost? This

is the problem that the current research attempts to solve.

This means that since the capability to process images in real time is a precondition for those algorithms to be used in practice, implementation on different architectures deserves thorough investigation. We consider three available hardware architectures: CPU, GPU, and dedicated hardware with special chips. For particular problems, dedicated hardware, such as that using FPGA techniques, can probably provide the best performance. However, this suffers from the disadvantages of being expensive and having a long development period, compared to already available processors such as CPUs and GPUs. GPU architecture and CPU architecture are designed basically for different purposes. While CPUs are dedicated to sequential processing, programmable GPUs are optimized for highly-parallel vertex and fragment shading code [8].

The long term research goal of the current thesis project is to investigate the performance of image processing algorithms on different system architectures. Specifically, we are interested in accelerating conventionally expensive computations and making them suitable for real time use by exploiting features of modern computer graphic hardware architectures and efficient algorithms.

Our idea of using computer graphics hardware to perform medical image restoration and analysis is inspired by the following facts:

- *Graphics processing units* (GPUs) are designed to exceed the performance of conventional CPUs when performing graphics related computations. In recent years, the performance of graphics hardware has increased more rapidly than that of CPUs [9].
- A lot of the features provided by the graphics hardware make it suitable for image processing tasks, which have a lot of similar characteristics to graphics applications.
- The new generation of GPUs provides much more flexibility in programma-

bility. With the development of high-level GPU-oriented programming languages, such as the Nvidia Cg and the OpenGL shading language, design has become more convenient for developers.

- Another noteworthy point is that the support on 32-bit floating point computations eliminates the limitation of insufficient precision, which previously existed. Although for certain problems there do exist scenarios where higher precisions such as 64-bit floating point format are desirable, for a lot of other problems the 32-bit floating point format is precise enough. Therefore, GPUs can be used as computation engines instead of only for traditional rendering and visualization purposes.
- Modern high-performance GPUs are affordable and can be used as coprocessors to improve the performance of the whole medical imaging system. If the image restoration and analysis approaches can be mapped and performed on a GPU, no burden (or less burden if some high-level control is required from CPU) needs to be added to the CPU. In this way, the CPU's resources can be used to do other tasks, such as signal acquisition and sampling, while the GPU does the extra computations and works on the desired outputs.

This thesis presents a modified Bayesian two-dimensional ultrasound image restoration model within the framework of *Joint Maximum A Posteriori* (JMAP). It is capable of estimating reflectance and variance fields from ultrasound images. Since the GPU has a totally different architecture from the CPU, we take special precautions when designing the algorithm. We demonstrate how image analysis can be conducted in a streaming model on GPUs. This is implemented on an NVIDIA Quadro 3000 GPU, and all the computations are carried out in multi-passes during the rendering entirely on the graphics card. It takes advantage of paralleled fragment processor units and high bandwidth-to-texture memory. Compared to the original model and CPU-based implementation, the computational time of each

iteration is reduced dramatically from 1.09 second to 11.49 millisecond. Estimation at interactive rates can be achieved by reasonable assumptions.

1.2 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 gives a review of the current ultrasound image restoration and analysis methods. It also introduces the new generation of programmable floating point graphics processing units (GPUs) and previous graphics hardware related work. Chapter 3 presents a Bayesian two-dimensional ultrasound deconvolution model, which is the base of our modified model. How this model can be improved exploiting the CPU architecture is also discussed in Chapter 3. Our modified Bayesian restoration model, suitable for mapping onto GPUs, is presented in Chapter 4. Special implementation issues to be aware of in order to maximize the performance on GPUs are discussed in Chapter 5. Chapter 6 analyzes the results and the performance of our model implementation on a programmable GPU, both on synthetic images and on acquired ultrasound images. Conclusions, limitations, and future work are discussed in the final two chapters.

Chapter 2

Background and Related Work

This chapter consists of two main sections. Section 2.1 presents the background of ultrasound imaging research, ultrasound image restoration, and previous work in ultrasound image analysis. Section 2.2 illustrates the current graphics architectures and the flexibility brought by the new generation of programmable floating point graphics processing units (GPUs). Recent work on how to utilize programmable graphics hardware in both graphics-related and scientific-computation-related applications is reviewed as well in Section 2.2.

2.1 Ultrasound Image Analysis

2.1.1 Ultrasound Imaging

In a typical ultrasound system, a transducer is used to generate short pulses into tissue. While a pressure wave propagates in the medium, part of it is reflected back at the interface between two different tissue types. This part is called a *back-scattering* echo. It is detected and measured by the transducer. The location of the reflector is determined by the distance that the wave travels. That distance is calculated as the multiplication of the default sound speed in soft tissue and the time in which the wave travels. The amplitude of the echo is utilized to determine

the intensity of the corresponding pixel in the displayed image. This is how a B-scan image is formed.

However, the process described above is the ideal case. In real life, the interaction between the transmitting waves and tissues is very complicated. As stated above, only part of the wave is reflected perpendicularly back to the transducer and measured. Some other parts of the wave might be reflected in other directions, due to the irregular shapes of the interfaces and the random acoustic impedance difference between tissues. Finally, those parts may arrive at other transducer elements and contribute improperly to other waves. The remaining part of the wave goes on travelling more deeply until its energy is exhausted.

A common assumption is that the received signal consists of two parts. The first part is called the *specular signal*, which is the reflected signal at tissue interfaces. The other part is called the *diffuse signal*, resulting from irregular interference by randomly distributed scatterers in the tissues.

It is known that B-scan ultrasound images suffer from low resolution. Homogeneous regions appear to be granular and detailed structures are concealed by artifacts in images. This is because of a variety of reasons, including complicated interactions between the pulse and the tissue, limitations of the hardware in the ultrasound imaging system, and inaccurate signal collection.

2.1.2 Ultrasound Image Degradation Model

There are various ways to model the image degradation process mathematically. The actual ultrasound images result from very complicated processes and involve many undetermined factors. Generally, the observed image is accepted as a distorted version of the ground truth. The inverse problem must be modelled explicitly in order to be solved quantitatively. Inevitably, certain assumptions are enforced.

Generally, image degradation can be modeled as

$$y = \bar{h}x + n \tag{2.1}$$

where \bar{h} is the degradation operator, y is the observed image, x is the true image intensity, and n is the additive noise.

If \bar{h} is not specified, we assume it is identity, and the model reduces to the simplest one that assumes only additive noise existing in the imaging system. Usually \bar{h} is defined to be a convolution operation with the system impulse response as the convolution kernel. Thus we assume a linear system with scaling and superposition properties which simplify the inverse problem. In the following content, we use h to represent the system impulse response and the operator $*$ to notate the convolution convolution. The degradation model is rewritten as

$$y = h * x + n \tag{2.2}$$

As can be seen from the general image degradation model given in Equation 2.2, the ultrasound image degradation process in the spatial domain can be modeled simply as a spatial convolution on the tissue reflectance with the system impulse response. The true tissue reflectance is determined by the tissue type. In the present project, we want to estimate this from blurred and noisy ultrasound images.

2.1.3 Ultrasound Point Spread Functions

The system impulse response h discussed above in the ultrasound degradation model is also often named as *Point Spread Function* (PSF)¹.

It is necessary to explore different factors that influence the system impulse response. In [10], the system response is described as a combination of a smoothing kernel and a distortion kernel. The smoothing kernel is determined by the transducer excitation and the impulse responses during emission and reception. The distortion kernel is both hardware-related and tissue-related. It depends not only on the geometry of the transducer and the spatial extent of the scattered field, but also on aberrations and dispersive attenuation, along with the pulse and echo.

¹These are used interchangeably through out the thesis.

In real medical imaging systems, PSF is not space invariant. Methods such as deconvolution with a one-dimensional spatially varying PSF in [11] have been developed to model variation in PSF, but commonly it is recognized to be hard and may even be impractical for some methods². Consequently, in most models including Husby et al.'s Bayesian model, introduced in Chapter 3 and also in our modified model in Chapter 4, the PSF is considered spatially invariant.

2.1.4 Ultrasound Image Restoration

Most ultrasound image restoration techniques are developed from existing common image restoration approaches, giving extra consideration to the specific physical characteristics inherent to ultrasound images.

Image restoration is a classic problem that has been studied for decades. This is inspired by the fact that various influences from the environment and the imaging system cause the observed images to always suffer from degradation, which makes them deviate from their true features. The major objective of image restoration is to recover the underlying true image information from those corrupted images. It is an inverse problem and is usually ill-posed.

Sometimes, the terms *deblurring* and *deconvolution* have the same meaning as *image restoration* when the degradation operator is assumed to be a convolution³. These terms all refer to the process of reconstructing the true images.

Katsaggelos gave a comprehensive classification of image restoration approaches in [12]:

- direct, recursive, or iterative;
- linear or nonlinear;
- deterministic, hybrid, or stochastic;

²This will be discussed further in Section 3.3.2.

³In this thesis, all three terms are used.

- spatial domain or frequency domain;
- adaptive or nonadaptive.

The categories are not exclusive to each other, which means a typical restoration method may fall into more than one of the categories. In the following section, currently available ultrasound image restoration approaches are explored.

2.1.5 Previous Work

Various attempts have been made to improve qualities of ultrasound images. The objectives are to remove speckle and noise while preserving fine structures.

Many approaches using signal processing techniques have been studied. A directional median filtering method which provides better results than median filtering and low-pass filtering is proposed in [13]. Taxt et al. develops homomorphic deconvolution methods in one-dimension [14], two-dimensions [15], and three-dimensions [16]. In their methods, ultrasound images are deconvolved by a Wiener filter in the frequency domain, using the point spread function estimated by cepstrum transforms. It is a direct method and the image resolution can be improved in real time. Other signal-processing-based approaches include a higher order spectral deconvolution model [10] and one-dimensional deconvolution with spatially varying PSFs by Kalman filter [11]. All these approaches, except the first one, require explicit estimation of the PSF from ultrasound images, and use this to perform blind deconvolution. They have the advantages of high processing speed, but are sensitive to the existence of noise.

Spatial compounding is an effective approach to increasing the signal-to-noise ratio (SNR). Multiple ultrasound image frames from different angles are acquired and then averaged to reduce noise. This approach has been investigated in [17], [18], [19], [20] and [21]. Recently, compounding techniques have been implemented in ultrasound systems [22]. Clinical results show that compounding imaging is a promising technique in practice, which can effectively increase resolution, while suppressing

speckle and noise. Real-time capability is one of its strongest advantages.

Statistical multi-scale approaches have been found to be effective when solving inverse problems, such as image restoration [23] and [24]. The correlations among wavelet coefficients are modelled by proper priors. Multi-scale wavelet principles have been extended for use in ultrasound speckle reduction. [25] applies wavelet shrinkage on the separated noise signal after a logarithmic transform to reduce noise. A Bayesian denoising algorithm is developed based on alpha-stable priors in the wavelet framework [26]. One drawback of wavelet-based methods is that artifacts might be induced due to translation dependence [24]. Another limitation is that determination of good parameters is required.

Bayesian statistical models for the restoration of ultrasound images have been investigated in recent years. Ideally, if a model of ultrasound scattering and degradation processes is available, the true reflectance image can be reconstructed from the observed image by solving the associated inverse problem. A *Maximum A Posteriori* (MAP) deconvolution model, based on discrete Markov Random Fields is developed by Hokland and Kelly [27]. In their approach, both diffuse and specular scattering are modelled explicitly, based on their physical properties. Due to the complexity of their model, [27] reports taking from two to ten hours to process each image on a DECstation 5000/125 workstation. [6] presents another Bayesian two-dimensional deconvolution approach which models diffuse scattering only. Markov Chain Monte Carlo is employed to sample from the posterior and for restoration. This is the model we study in this thesis. We investigate the efficiency performance of the original model and our modified model on different architectures.

2.2 Programmable Graphics Processing Unit (GPU)

2.2.1 Modern Graphics Architectures

Graphics processing units, GPUs for short, are designed specifically to perform graphics-oriented tasks. A GPU is able to process multiple fragments (which are precursors of pixels) in parallel but treat them independently. The parallelism brought by no communication among fragments is one of the most important factors that improve the performance of GPUs. Memory latency can be hidden by this as well.

Without losing generality, a computer graphics pipeline based on the OpenGL API is shown in Figure 2.1.

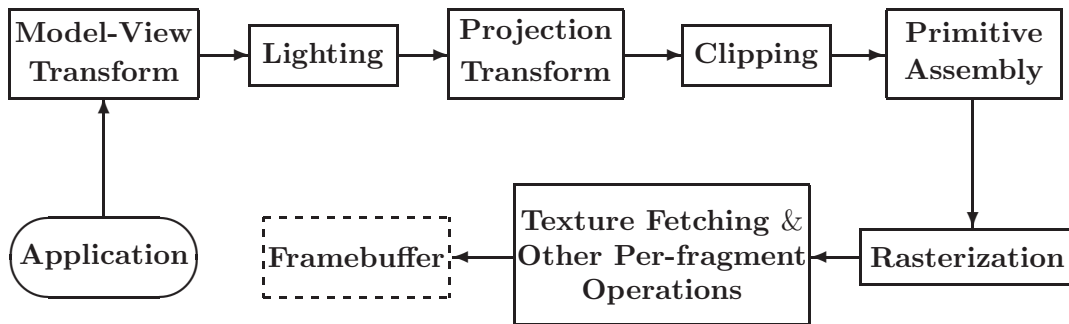


Figure 2.1: A typical graphics pipeline based on the OpenGL API. Arrows represent data streams.

The graphics pipeline consists of sequential stages. GPUs receive 3D graphics vertices from the application. Every vertex has its features, such as 3D positions, orientations, texture coordinates, and colors. Mathematical operations are executed on those data values. Positions for these vertices in homogeneous clip space are calculated by transformations before the *rasterization* stage, which generates fragments by interpolating the attributes of those vertices. The parameters associated with each fragment include colors and texture coordinates. The texture coordinates can be used to fetch the values from pre-stored textures. Those values can be combined with the color attributes associated with the fragment for any computation pur-

poses. The resulting colors are written into the frame buffer by either replacing or blending with the values already in it.

With the development of modern GPUs, some of the originally fixed stages in the graphics pipeline are made explicit to the user and are now programmable. Developers can specify the operations they want based on their specific applications by certain programs, load them onto the graphics card, and execute them during the rendering. This helps developers better utilize the computation available on GPUs. Figure 2.2 shows an abstract programmable graphics pipeline.

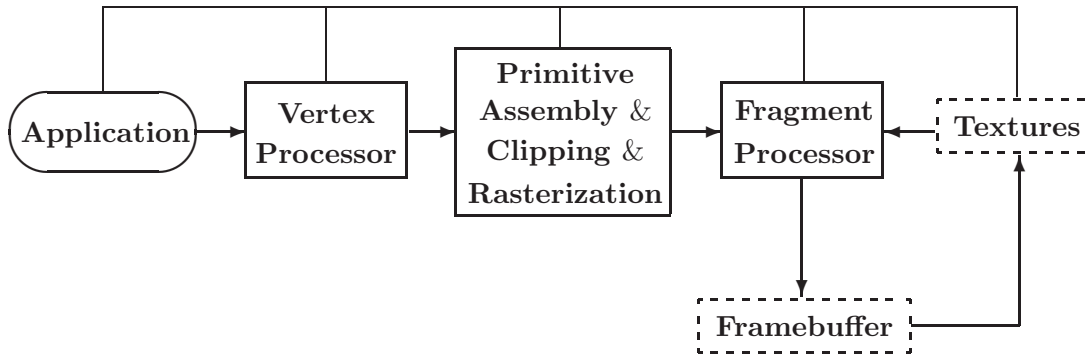


Figure 2.2: Programmable graphics pipeline. Arrows represent data streams. The application and the two programmable processors are connected by lines, which means the application has control of them. Parameters can be transferred between the application and the processors.

There exist two programmable processors taking the place of some of the original fixed functions in the pipeline: the *vertex processor* and the *fragment processor*. Their features are discussed below.

1. *Vertex processor*

As the name indicates, a vertex processor can only operate on vertices, not on primitives. It performs traditional transformation and lighting tasks on the vertices sent in from the application. The programs which define the operations on the vertices at this stage are called *vertex programs* or *vertex shaders*.

2. *Fragment Processor*

The fragment processor performs per-fragment operations that are specified in *fragment programs* or *fragment shaders* on all the fragments passing through it. It supports a variety of inputs from the rasterization stage, including pixel positions, multiple texture coordinates, color values, fog, and point size. It can also accept the constant parameters and texture handlers. Developers have complete control of fragment-wise operations such as shading and texture computations. This is done by a *single instruction multiple data* (SIMD) way and thus, all the fragments have the same operations. The fragment program operates on one fragment; every fragment is invisible to all the others, which means they cannot read each others' attributes. Every fragment can access any value in the textures.

Present programmable graphics hardware supports rich instruction sets for both programmable stages. These instructions are optimized for 4-component vectors, which are the most common data structures in graphics to represent colors and positions. The vertex processor and fragment processor have their own instruction sets with a shared subset. The special graphics architecture enforces some limitations on the available programmability. For example, the current vertex processor does not support texture fetching and the fragment processor cannot perform unrolled loops.

2.2.2 GPU Programming Languages

Currently available GPU program languages can be classified into two: low-level assembly-like languages, such as DirectX 9 Vertex Shader 2.0 and Pixel Shader 2.0, and high-level languages, such as OpenGL shading language, Nvidia Cg, Microsoft High-Level Shader Language (HLSL), and Stanford Real-Time Shading Language.

Our vertex and fragment programs are written in Nvidia's Cg (C for Graphics) shading language. It is important to point out that Cg is functional and not

hardware specific [28]. It is easy to master since the syntax is like the C programming language. Assembly code is generated either at compile time or runtime automatically by the Cg compiler, based on the Cg programs, and is loaded onto GPUs and executes during rendering.

2.2.3 Evolution on GPUs - High Precision

Usually, images are recorded as two-dimensional arrays of pixel values. These are called *intensities* and are typically small integers describing the brightness of the corresponding pixels. The traditional graphics cards use 8 bits for each color value, which is insufficient for many image processing tasks. They are also almost impossible to use for common scientific computing tasks. In 2002, ATI announced the first floating point GPUs that support 24-bit floating point precision (s.e7.m16)⁴. In 2003, Nvidia released their high precision GPUs that provide two floating point formats through out the pipeline, including the framebuffer, textures, vertex processor, fragment processor, and interpolation. The two formats are full 32-bit IEEE floating point (s.e8.m23)⁵ and 16-bit floating point (s.e5.m10)⁶ respectively.

2.2.4 Previous Work

Since the present GPUs can be exploited as fast parallel processors and streaming processors with high precision and flexibility, they can be used as computation engines as well as common visualization engines. Considerable effort has been made to investigate elaborate acceleration algorithms and techniques for graphics hardware.

The latest GPUs have many features that image processing and analysis applications can take advantage of. Hopf and Ertl investigate how to use graphics hardware to accelerate wavelet transforms, three-dimensional separable convolution and morphological operators in [29], [30], and [31]. *Fast Fourier Transform* (FFT)

⁴1 bit for the sign, 7 bits for the exponent and 16 bits for the mantissa.

⁵1 bit for the sign, 8 bits for the exponent and 23 bits for the mantissa.

⁶1 bit for the sign, 5 bits for the exponent and 10 bits for the mantissa.

implementation on GPUs is given in [2]. Although currently the performance of FFT and IFFT on GPUs cannot compete with the performance by highly optimized libraries executed on CPUs, they do open the door for many image processing and signal processing algorithms to be mapped onto GPUs. Nonlinear diffusion on images is presented in [32]. [33] investigates some common image processing problems, such as local mean filtering, color space mapping, local principal component analysis, and anisotropic diffusion filtering on graphics hardware.

With a similar goal as in our research, [34] and [35] examine the capabilities of GPUs in performing medical image processing tasks. Graphics hardware accelerated segmentations on MRI and CT images are investigated, respectively, and both achieve a more than 10 times speedup over the corresponding optimized CPU implementation.

A lot of attention has been given to exploiting the scientific computation capability of GPUs. Due to the special graphics architecture, grid-based computations are suitable to be executed efficiently on GPUs. Matrix-matrix multiplication on a GPU is first presented in [36] using multi-texturing. Later, Hall et al. propose a multichannel block-based matrix multiplication algorithm and discuss bandwidth related issues [37]. [38] presents a general-purpose vector computation framework by utilizing vertex programs and also presents applications on matrix multiplication and solving 3-SAT problem. [39] shows efficient dense matrix multiplication on GPUs. For matrices of size 1000×1000 , full matrix multiplication requires about 0.6 (ms).

The way to perform classical graphics tasks on graphics hardware efficiently has also been investigated. Purcell et al. implement ray tracing in a stream processing model on a GPU. They also predict that with the increasing generality of GPUs, more parallelizable algorithms can be supported on GPUs, which can be viewed as streaming processors [9]. Harris et al. examine programmable graphics hardware and successfully use it in physical simulation and cloud simulation in [40] and [41].

Collision detection on GPUs is demonstrated in [42] and [43].

2.2.5 Recent Related Work

Two recent publications are particularly relevant to our work. Iterative optimizations using the Conjugate Gradient method are discussed in [44] and [45] with different applications and implementation techniques. Krüger and Westermann introduce a representation for vectors and dense, banded- or random-sparse matrices on GPUs in [44]. Using those representations, basic linear algebra operators are implemented. The Conjugate Gradient method is presented with these basic operators as building blocks. [45] implements a multi-grid solver for fluid problems by special sparse matrix storage and element fetching.

We use the Conjugate Gradient method in one stage of our model to estimate the reflectance variable and perform iterative image restoration. This is a new application of the CG method on the graphics hardware. Our implementation is different from the two papers mentioned above in that we do not carry out the matrix vector multiplication, an expensive step in CG. Instead, we take advantage of the special structure of the Hessian matrix and use two-pass convolutions to complete the task. We also develop a strategy to do convolution efficiently on GPUs.

Chapter 3

Bayesian 2D Deconvolution Model and Acceleration on CPUs

3.1 Statistical Ultrasound Speckle Models

Echo signals detected by the ultrasound transducer are a combination of reflected echoes from tissue boundaries with echoes from randomly distributed scatterers in the medium. Statistical models are able to characterize these random properties and help to better explain the scattering phenomena in target media. With these models, the relationship between the medium parameters determined by the underlying tissue structures and the observed images formed by the received signals can be investigated. This enables refining the ultrasound image analysis approaches.

According to [46], there are two ways to represent the received signals: one is the *radio-frequency* (rf) signal and the other is the *magnitude* signal. The magnitude signal is the envelope of the corresponding rf signal. The statistical properties of these two types of signals in medical ultrasound are first discussed in [46]. Dutt conducts a thorough first order statistical analysis on ultrasound echo envelope signals

by a variety of distributions in his Ph.D. thesis [47] and develops a K-distribution-based adaptive filter to reduce ultrasound speckles.

The magnitude signals are commonly used in clinical ultrasound imaging systems to display B-scan ultrasound images. Thus, through out this thesis, we work on image data represented by envelope-detected signals. The term *speckle* is defined as the interference pattern formed in the envelope image in [47]. Here, we present the probability density functions that can model the scatterer randomness. Detailed derivations can be found in either [46] or [47].

The basic two distributions used are Gaussian and Rayleigh, which are developed from the study of laser speckle by Goodman in [48]. They are suitable to model the cases when the number of scatterers is large and the spacing between scatterers is small.

The echo signal from scattering media can be modeled as a sum \mathbf{X} of the backscattering signals from a number of randomly located scatterers in the media [47]. Because of the variations of both the random phase and the random amplitude, \mathbf{X} is a complex phasor. Using the central limit theorem, both X_r and X_i , which are \mathbf{X} 's real and imaginary parts, follow Gaussian distribution:

$$\begin{aligned} p(X_r) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{X_r^2}{2\sigma^2}} \\ p(X_i) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{X_i^2}{2\sigma^2}} \end{aligned} \quad (3.1)$$

Since X_r and X_i are uncorrelated, their joint density function is

$$p(X_r, X_i) = \frac{1}{2\pi\sigma^2} e^{-\frac{X_r^2 + X_i^2}{2\sigma^2}} \quad (3.2)$$

The amplitude X and phase ϕ of the complex signal \mathbf{X} are defined by

$$X = \sqrt{X_r^2 + X_i^2} \quad X \geq 0 \quad (3.3)$$

$$\phi = \tan^{-1} \frac{X_i}{X_r} \quad -\pi \leq \phi \leq \pi \quad (3.4)$$

The corresponding density function in Equation 3.2 in the polar coordinate system can be derived by

$$p_{X\phi}(X, \phi) = p(X_r, X_i) \left| J \left(\begin{array}{c} X_r, X_i \\ X, \phi \end{array} \right) \right| \quad (3.5)$$

where

$$\begin{aligned} J \left(\begin{array}{c} X_r, X_i \\ X, \phi \end{array} \right) &= \text{Det} \left[\begin{array}{cc} \frac{\partial X_r}{\partial X} & \frac{\partial X_i}{\partial X} \\ \frac{\partial X_r}{\partial \phi} & \frac{\partial X_i}{\partial \phi} \end{array} \right] \\ &= \text{Det} \left[\begin{array}{cc} \cos \phi & \sin \phi \\ -X \cdot \sin \phi & X \cdot \cos \phi \end{array} \right] \\ &= X \cdot \cos^2 \phi + X \cdot \sin^2 \phi \\ &= X \end{aligned} \quad (3.6)$$

And thus

$$p_{X\phi}(X, \phi) = \frac{1}{2\pi\sigma^2} e^{-\frac{X^2}{2\sigma^2}} \quad X \geq 0 \quad (3.7)$$

The probability density function of the envelop signal, which is the amplitude X , is the marginal density [47]

$$\begin{aligned} p_X(X) &= \int_{-\pi}^{\pi} p_{X\phi}(X, \phi) d\phi \\ &= \frac{X}{\sigma^2} e^{-\frac{X^2}{2\sigma^2}} \quad X \geq 0 \end{aligned} \quad (3.8)$$

This distribution is known as a Rayleigh distribution [47].

The variance σ^2 in both of the two density functions in Equations 3.2 and 3.8 depend on the mean-square scattering amplitude of the particles in the scattering medium [48]. Other papers refer to this as the “mean backscattering energy” [47]. Based on this physical property, the σ^2 field itself can be used to represent the underlying structures.

Many of the ultrasound image analysis approaches use the Gaussian distribution with the assumption that only the real part of the signal contributes to the displayed images while the imaginary part is discarded. The Gaussian distribution

has good computation properties and thus can be easily incorporated into complex systems. It is adopted in the Bayesian model presented in the next section.

The Rice distribution and K distribution are two variants of the Rayleigh distribution. The former distribution can model the coherent signal, which may be an additional part of the diffuse scattering signal; the latter is capable of dealing with those cases having low scatterer densities. Rice distribution has been used in ultrasound texture analysis (see [49] and [50]), and has been studied intensively as well. More introduction and applications can be found in [47] and [17].

3.2 Standard Model Description

3.2.1 Introduction

In this section, the two-dimensional Bayesian diffuse scattering deconvolution model developed by Husby et al. in [6] is presented. As discussed in the speckle model section in 3.1, the variance field σ^2 has a special feature. It depends on the physical properties of the scattering tissue and can be used as a representative of the underlying tissue structures.

The basic idea behind this Bayesian model is to have σ^2 as the objective parameter and estimate it by using a combination of the speckle model and the degradation model. Let us first look at the variables used in the model. The notations in [6] are followed.

- r - reflectance field
- σ^2 - variance field of r
- y - observed image
- $\beta, \gamma, \delta, \epsilon$ - fixed parameters in the σ^2 prior
- h - point spread function

- n - additive noise
- τ^2 - variance of additive noise n
- ω - coefficient of Markov Random Fields (MRF) neighborhood system

σ^2 is regarded as the major objective variable and it is desirable to be estimated from its posterior. h is assumed to be known in advance; therefore, it can be treated as fixed. We provide full details in the next sections.

3.2.2 Model Derivation

The relationships among unknown variable σ^2 , unknown variable r , and observed image data y build up the whole model.

- $r \longleftrightarrow \sigma^2$

The association between r and σ^2 is determined by the statistical speckle model. The Gaussian distribution is used and the two-dimensional discrete form is given in Equation 3.9, from [6]. Here (i, j) is defined to be an image resolution cell in the two-dimensional target image I .

$$p(r_{i,j}|\sigma_{i,j}^2) = \frac{1}{\sqrt{2\pi\sigma_{i,j}^2}} \exp\left\{-\frac{r_{i,j}^2}{2\sigma_{i,j}^2}\right\}, \quad \forall (i, j) \in I \quad (3.9)$$

- $r \longleftrightarrow y$

Similarly to most of the ultrasound image restoration approaches, the degradation operator is defined to be a convolution of the true image (or the reflectance field r) with the PSF. Referring to the degradation model in Equation 2.2, the likelihood probability density function is defined in Equation 3.10, from [6]. This gives the relationship between r and the observed image y :

$$p(y|r, h, \tau^2) \propto \exp\left\{-\frac{1}{2\tau^2} \sum_{i,j} \left(y_{i,j} - \sum_{k,l} h_{k,l} r_{i-k,j-l}\right)^2\right\} \quad (3.10)$$

- σ^2

The prior is one of the key components in the Bayesian paradigm. The prior on σ^2 used in [6] is described as follows. Apparently, since the variance is related to the average scattering energy, it should vary little in homogeneous tissue regions, which present spatial smoothness. This is different from the reflectance field, which results from the random interference by the echoes from random scatterers. Another natural belief on the variance field is that abrupt changes are expected to be observed at the interfaces of different types of tissues [6]. The prior on σ^2 is defined in Equation 3.11, from [6].

$$p(\sigma^2) \propto \exp \left\{ -\beta \sum_{i,j \sim k,l} \omega_{ij,kl} \phi(\ln(\sigma_{i,j}^2 / \sigma_{k,l}^2)) - \sum_{i,j} (\epsilon / \sigma_{i,j}^2 + (\epsilon + 1) \ln \sigma_{i,j}^2) \right\} \quad (3.11)$$

Here \sim defines a symmetric neighborhood relationship. $i, j \sim k, l$ stands for site (i, j) and (k, l) are neighbors. $\omega_{ij,kl}$ is a set of positive weights that determine the influential strength of these neighbors. The interaction function $\phi(\cdot)$ in Equation 3.11 is given as

$$\phi(u) = \frac{|\frac{u}{\delta}|^\gamma}{1 + |\frac{u}{\delta}|^\gamma} \quad (3.12)$$

In [6], the two terms in the σ^2 prior in Equation 3.11 are named as a structured term and an unstructured term, respectively. Further explanations can be found there.

- $\sigma^2 \longleftrightarrow y$

For computation efficiency reasons, r is considered to be an auxiliary variable in the posterior equation. An assumption is that given r , the observed image y and variance σ^2 are independent of each other. Based on this assumption, the estimation on σ^2 is quite simplified.

Having the discrete model elements, the joint density on r and σ^2 is derived as

$$\begin{aligned} p(r, \sigma^2 | y, h, \beta, \delta, \gamma, \tau^2) &\propto p(y | r, \sigma^2, h, \beta, \delta, \gamma, \tau^2) p(r, \sigma^2 | \beta, \delta, \gamma, \tau^2) \\ &\propto p(y | r, h, \tau^2) p(r | \sigma^2) p(\sigma^2 | \beta, \delta, \gamma) \end{aligned} \quad (3.13)$$

The second proportional equality comes from the assumption that the introduction of r decouples y and σ^2 .

It is important to notice that the three density functions on the right side of Equation 3.13 are degradation likelihood distribution, statistical speckle distribution, and σ^2 prior distribution, respectively. Inserting Equations 3.10, 3.9, and 3.11 into Equation 3.13, we have

$$\begin{aligned} p(r, \sigma^2 | y, h, \beta, \delta, \gamma, \tau^2) &\propto \exp \left\{ - \sum_{i,j} \left(\frac{1}{2\tau^2} \sum_{i,j} \left(y_{i,j} - \sum_{k,l} h_{k,l} r_{i-k,j-l} \right)^2 + \frac{r_{i,j}^2 + 2\epsilon}{2\sigma_{i,j}^2} \right. \right. \\ &\quad \left. \left. + \left(\epsilon + \frac{3}{2} \right) \ln \sigma_{i,j}^2 \right) - \beta \sum_{i,j \sim k,l} \omega_{ij,kl} \phi \left(\ln(\sigma_{i,j}^2 / \sigma_{k,l}^2) \right) \right\} \end{aligned} \quad (3.14)$$

In [6], the posterior distribution is examined using the *Markov Chain Monte Carlo* (MCMC) analysis. Having the joint density of r and σ^2 , the full conditional distributions of $r_{i,j}$ and $\sigma_{i,j}^2$ can be derived, from [6]

$$\begin{aligned} p(r_{i,j} | r_{-(i,j)}, \sigma^2, y, h) \\ &\propto \exp \left\{ - \frac{1}{2\tau^2} \sum_{m,n} \left(y_{m,n} - \sum_{(k,l) \neq (m-i, n-j)} h_{k,l} r_{m-k, n-l} - h_{m-i, n-j} r_{i,j} \right)^2 - \frac{r_{i,j}^2}{2\sigma_{i,j}^2} \right\} \end{aligned} \quad (3.15)$$

$$\begin{aligned} p(\sigma_{i,j}^2 | \sigma_{-(i,j)}^2, r, \beta, \delta, \gamma) \\ &\propto \exp \left\{ - \left(\epsilon + \frac{r_{i,j}^2}{2} \right) \frac{1}{\sigma_{i,j}^2} - \left(\epsilon + \frac{2}{3} \right) \ln \sigma_{i,j}^2 - \beta \sum_{(k,l) \in \partial(i,j)} \omega_{ij,kl} \phi \left(\ln \sigma_{i,j}^2 / \ln \sigma_{k,l}^2 \right) \right\} \end{aligned} \quad (3.16)$$

Here $-(i, j)$ refers to all the other pixels in the image, other than (i, j) itself.

3.2.3 Assumptions on PSF

In Husby et al.'s paper, PSF is modelled as a fixed two-dimensional *sine* signal multiplied by a Gaussian window. What we want to address here is that the recovery of the PSF is a very hard inverse problem itself. In our model, presented in the following chapter, we exploit signal processing methods and calculate an approximation of the PSF directly from the image data. The influence of the inaccurate PSF is discussed in [51]. It is found that small variations in the parameters characterizing the PSF, less than $\pm 25\%$ change of frequency, width, or length, still gave satisfactory results [51].

3.2.4 Sampling the Reflectance Field

Sampling from the posterior full conditional using *Markov Chain Monte Carlo* (MCMC) method usually needs expensive computations. [6] rearranges the terms in the updating equation in order to reduce the number of calculations and facilitate implementation. Equation 3.15 can be approximated as Equation 3.17 by expanding the quadratic term and skipping some terms irrelevant to $r_{i,j}$, from [6].

$$\begin{aligned}
 & p(r_{i,j} | r_{-(i,j)}, \sigma^2, y, h) \\
 & \propto \exp \left\{ -r_{i,j}^2 \left(\frac{1}{2\tau^2} \sum_{m,n} h_{m-i,n-j}^2 + \frac{1}{2\sigma_{i,j}^2} \right) + 2r_{i,j} \left(\frac{1}{2\tau^2} \sum_{m,n} h_{m-i,n-j} \Upsilon_{m,n} \right) \right\}
 \end{aligned} \tag{3.17}$$

where

$$\Upsilon_{i,j} = y_{m,n} - \sum_{(k,l) \neq (m-i,n-j)} h_{k,l} r_{m-k,n-l} = y_{m,n} - \sum_{k,l} h_{k,l} r_{m-k,n-l} + h_{m-i,n-j} r_{i,j}$$
(3.18)

The terms that are independent of $r_{i,j}$ are ignored since they can be cancelled out in the sampling process. The reason for rearranging the equation is that instead of doing the full computation required by Equation 3.15, the values of $\Upsilon_{m,n}$ can be stored in an array. In this way, after updating each pixel, only some entries in $\Upsilon_{m,n}$ are updated, and thus the total number of computations can be notably reduced.

Clearly, Equation 3.17 tells us that the conditional distribution of $r_{i,j}$ follows a Gaussian distribution [6]. Its mean $\mu_{i,j}$ and variance $\kappa_{i,j}^2$ are given in Equations 3.19 and 3.20, respectively, from [6].

$$\mu_{i,j} = \frac{\frac{1}{\tau^2} \sum_{m,n} h_{m-i,n-j} \Upsilon_{m,n}}{\frac{1}{\tau^2} \sum_{m,n} h_{m-i,n-j}^2 + \frac{1}{\sigma_{i,j}^2}} \quad (3.19)$$

$$\kappa_{i,j}^2 = \left[\frac{1}{\tau^2} \sum_{m,n} h_{m-i,n-j}^2 + \frac{1}{\sigma_{i,j}^2} \right]^{-1} \quad (3.20)$$

There are many standard ways to sample from a Gaussian distribution. [6] chooses a Gaussian distribution centered at the current state as a proposal distribution. Given the mean and variance, the updating formula is

$$r_{i,j} = \mu_{i,j} + norm \cdot \kappa_{i,j}^2 \quad (3.21)$$

where $norm$ is a random variable following normal distribution. It is generated by the *Box-Müller* method, given in Figure 3.1.

1. **Generate** $u_1, u_2 \sim U[0,1]$.
2. $norm = \sqrt{-2 \ln(u_2)} \cdot \cos(2\pi u_1)$

Figure 3.1: Pseudocode generating a normal random number using the *Box-Müller* method, from [4].

3.2.5 Sampling the Variance Field

The *Metropolis-Hastings* (MH) algorithm is employed to sample the σ^2 from its conditional probability density function. The MH algorithm has many variations and the *Independent Metropolis Hastings* (IMH) algorithm is one of them. The IMH algorithm is given in Figure 3.2.

The distinctive property of the IMH algorithm over other MH algorithms, such as the *Random Walk* MH algorithm, is that when IMH generates the candidate

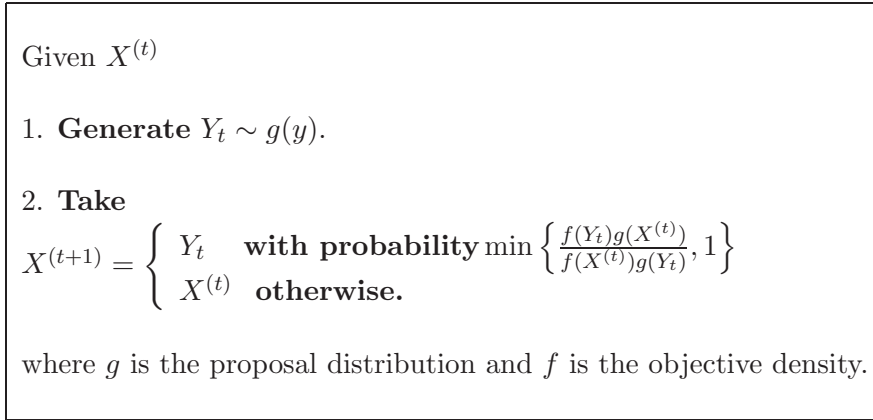


Figure 3.2: Independent Metropolis-Hastings algorithm, from [5].

value for one site, the current value at this site is not taken into account. IMH is the algorithm used in [6] and the proposal function is given in Figure 3.3.

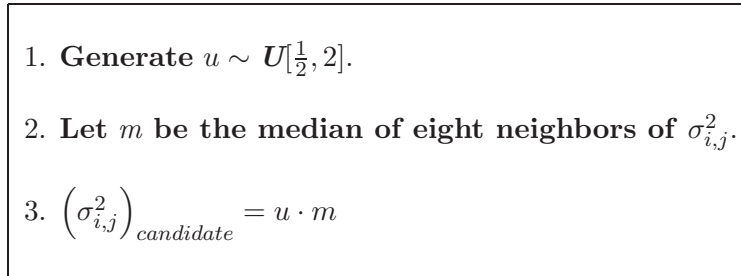


Figure 3.3: Proposal function on $\sigma_{i,j}^2$, from [6].

Specifically, the median of the current pixel's eight neighbors is calculated first. Then, the candidate value of $\sigma_{i,j}^2$ is defined to be a scaled value on the median. The scalar itself is a uniform random number $u \sim U[\frac{1}{2}, 2]$. The MH algorithm follows a sequential updating scheme that requires one site to be processed at a time. Given the neighbors, the state of the current site is independent of the other sites in the two-dimensional Markov Random Field. Therefore, although the conditional density requires the full conditional $p\left(\sigma_{i,j}^2 | \sigma_{-(i,j)}^2, r_{i,j}\right)$, the locality brought by the Markov Random Fields can dramatically reduce the computation, since $p\left(\sigma_{i,j}^2 | \sigma_{-(i,j)}^2, r_{i,j}\right) = p\left(\sigma_{i,j}^2 | \sigma_{\vartheta(i,j)}^2, r_{i,j}\right)$, where $\vartheta(i, j)$ represents the small neighborhood region.

3.3 Improved Model Based on CPU Architecture

Based on the model described above, we describe a *block restoration* method which can simultaneously speed up estimation and improve image quality for a sequence of ultrasound images. We achieve this using three techniques, as described in the following sections.

3.3.1 Ensuring Good Data Locality and Caching Behavior

The efficiency of the program on a CPU is highly related to the CPU cache performance. In order to utilize the CPU cache efficiently, we want the data to be loaded into cache and be reused for some time. In practice, the original image domain is partitioned into non-overlapping, equal-sized blocks. The blocks are processed one by one and the sites inside each block are updated randomly, using MCMC. This method has not yet run in real time, but represents a significant step towards this goal.

The performance of modern processors is very sensitive to the distribution of data within the memory hierarchy. For instance, at 5-1-1-1¹ for 100MHz SDRAM, it takes 32 CPU cycles to read 16-bytes of data from main memory while accessing the same amount of data on a 2-1-1-1² 100MHz L2 cache requires 20 CPU cycles. So, the more times the data is in the L1 and L2 caches, the faster the overall CPU performance is. The L2 cache on an Intel **Pentium III** 1.01GHz processor has a limited size of 256KB. If we have a 128×128 image (which is smaller than those used clinically) and store it in a floating point array, it requires a contiguous storage of 64KB. There are five reference matrices to operate during the estimation. The visiting scheme used is random access, which means we might need the data across

¹These numbers express the performance of the **rated burst mode** of memory on CPUs. The **rated burst mode** is a special technique to compensate for latency. 5-1-1-1 tells us that it takes five clock cycles to read the first bit and one cycle for each bit after that [52].

²2-1-1-1 means that it takes two cycles to read the first bit and one cycle for each bit after that.

the whole image at any time. These can not fit in the L2 cache. Unavoidably, much time is spent on the data communication between the L2 cache and main memory. However, if we use an appropriate block size (in our case 64×64 , which requires 16KB storage), the entire block fits in L2, and we can “crunch on it” for awhile and leave the other blocks in the main memory untouched.

Table 3.1 gives the time comparison with and without image partitioning. Each number on the second and the third column represents the time spent on one iteration. We can see an order of magnitude improvement, which is a great step towards interactive processing.

Image Size	Without Partition	With Partition
128×128	2.1196	1.0940
128×256	14.1189	2.4184
256×256	74.2744	5.3997

Table 3.1: Running time (second/iteration) comparison with and without partition.

Both the two programs are implemented using the **Intel Processing Package** (IPP) and optimized code. They are run on a dual CPU system with two identical PentiumIII processors. Each processor has a 32K L1 cache and a 256KB L2 cache. The clock speed is 1.0GHz and the system bus frequency is 133MHz. The two programs were run for 1000 iterations each to obtain the statistics. The speedup grows greater for higher resolution clinical images.

3.3.2 Partially Spatially Variant PSFs

We add one more pre-process step in the model which estimates the PSF directly from the image data using homomorphic transformation. Assumption on the spatially invariant PSFs is reasonable for many applications where the image degradations are caused by telescopes or cameras.

For ultrasound images, due to attenuation, aberration, and reflection, the shape and frequency of the pulse changes as it interrogates the body. This is par-

ticularly true along the axial direction. An ultrasound pulse contains a range of frequencies, and attenuation is proportional to frequency. Theoretically, the pulse elongates as it penetrates tissue, which lowers the axial resolution. Therefore, spatially variant PSFs would be better approximations of the real degradation.

We observe the incidental benefit brought by the blocking scheme discussed above. The main idea is to estimate a PSF for each subimage, which approximates the blurring process. The PSFs are considered to be located at the centers of blocks. Blurring of each block is taken to be due to the form of the actual PSF at the center. Such PSFs exhibit better local information because each PSF is calculated from that subimage instead of the whole image. Since each subimage after partitioning needs to be processed independently, this multiple PSF strategy does not add extra computation after the PSFs are determined.

3.3.3 Inter-frame Coherence

It is often the case that two images in a sequence have much similarity, so that we may use the underlying tissue information of one image as the initialization of the one next to it, as suggested in “future work” in [6]. We use random initialization on the very first image in the sequence, since we know nothing about it. We can subsequently use a blurred version of a previous variance field in the initialization as the prior knowledge for the next frame. The blurring process may reduce the strength of the prior and reduce the uncertainty of the second image. Such inter-frame speedup is very important for online interactive applications.

Figure 3.4 gives two real ultrasound images. The second image is acquired by moving the probe a little away from where the first image is acquired. They represent two ideal consequent images in a image sequence. Figure 3.5 shows that when the second image is initialized by the variance field generated by the first image, it requires less iterations to converge.

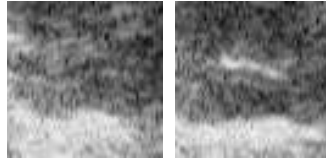


Figure 3.4: Two real images, namely Image1 (on the left) and Image2 (on the right) respectively, in a sequence. They are both of resolution 64×64 .

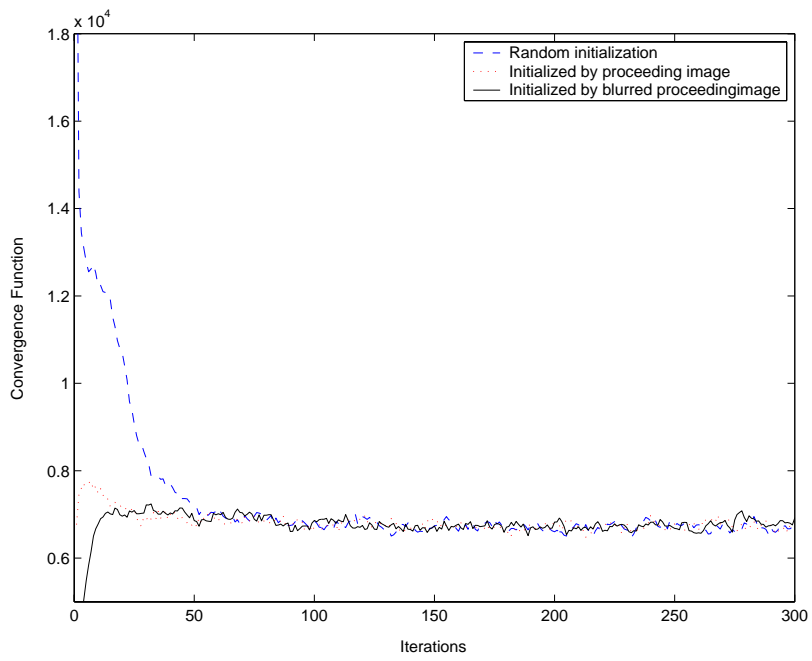


Figure 3.5: Convergence acceleration by inter-frame knowledge.

Chapter 4

Ultrasound Image Restoration within JMAP Framework

In Section 4.1, we first discuss why a modified hybrid model is needed. An introduction of two groups of *Maximum A Posteriori* (MAP) estimation approaches, deterministic methods and stochastic methods, is given in Section 4.2. Our model, which falls in the *Joint Maximum A Posteriori* (JMAP) framework, is presented in Section 4.3. Details of the three steps in the model are discussed in Section 4.4.

Our JMAP model is a modified version based on Husby et al.'s model, presented in Chapter 3. We interpret the model structure from another point of view and incorporate deterministic optimization methods into the model in order to make it suitable for GPU acceleration.

4.1 Motivation for Modified Model

One main drawback of the deconvolution model presented in Chapter 3 is that it takes too long to process images. Although our block restoration method takes advantage of better caching behaviors and improves speed performance, especially for high resolution image processing, it still suffers from the limit of memory bandwidth inherent in the CPU architecture. This is a restriction that cannot be overcome be-

cause of the huge data demand of image processing applications on current CPU architecture. We observe that new generation of GPUs are more suitable for tasks such as iterative image restoration. The relevant features are listed below:

- The streaming GPU architecture favors iterative methods that continually feed data into the processor, one pass after another.
- Many graphics related tasks require fast texture access from video memory. Current GPUs support high memory bandwidth to fulfil this requirement.
- Flexible programmability of GPUs provides the possibility of specifying the pixel operations required by the algorithms.
- Parallel fragment processors on GPUs make them effective processing engines.

Our research goal is to explore the maximum performance that GPUs can achieve for traditionally expensive iterative approaches. Due to the special architectures of GPUs, one precondition for the investigation is that the application models are suitable to be mapped onto GPUs. The Bayesian model discussed in the proceeding chapter requires updating sites sequentially, which makes it hard to program on GPUs. There are two major reasons for this:

1. The current generation of GPUs does not allow *read* and *write* operations on the same site to be executed in one pass.
2. The parallelism of GPUs makes it impossible to control the order of processing pixels.

Further discussions can be found in Chapter 5.

Therefore, certain modifications are necessary if we want to study the performance of this model on GPUs. When looking at the conditional density function on the reflectance variable r , we find that it is the exponential of a regular quadratic form and that it is possible to find the maximum using iterative deterministic optimization approaches. The Steepest Descent iterative restoration is a classic approach

to restoration while the Conjugate Gradient method offers improvement in speed of convergence over the Steepest Descent method and is regarded as one of the best methods of its kind [53]. We use the Conjugate Gradient (CG) method in our model. The CG method can be mapped onto GPUs by using the procedural texture idea. We keep the MCMC estimation approach for σ^2 , as proposed in [6]. By replacing the second order neighborhood system by the first order one, each iteration can be done in two passes on GPUs. Then the whole model can be mapped onto GPUs.

4.2 MAP Estimate Methods Exploration

The *Maximum A Posteriori* (MAP) estimation of the objective variable is computed by maximizing the posterior density function. According to [54], iterative MAP estimation approaches can be classified into two categories: deterministic methods and stochastic methods.

4.2.1 Deterministic Methods

This class of methods has a longer history than the stochastic optimization methods. It provides an estimate of a local probability maximum by “stepping through” the function space in directions intended to locate a maximum [54]. Usually, the objective function is approximated as a quadratic form whose maximum can be calculated analytically. However, if the approximation is inappropriate, it may fail to find the desired solutions. Generally, the success of such approaches depends on the analytical properties of the objective function, such as convexity [5]. The Conjugate Gradient (CG) method is a popular and effective method belonging to this class.

4.2.2 Stochastic Methods

Stochastic methods examine objective function from a probabilistic point of view. They perform a series of stochastic jumps in the variable space from the starting

point [54]. The optimization process exhibits randomness. There are a lot of variations based on different proposal functions that define the jumps. These approaches are well known for imposing fewer restrictions on the objective functions. If the objective function is very complicated and far from being a quadratic form, stochastic approaches are still able to search the global maximum. Usually the computations required are expensive. *Markov Chain Monte Carlo* (MCMC) optimization is a stochastic method that has been applied successfully in many applications, including the Bayesian deconvolution model from [6], introduced in Chapter 3.

4.3 JMAP Ultrasound Image Restoration Model

4.3.1 Description of Model Elements

Generally, unlike ordinary image restoration, our ultrasound analysis model has one more variable that needs to be estimated from the images, namely the variance σ^2 . Figure 4.1 shows a graphic model which specifies the relationship of all the variables and parameters in the analysis model¹.

The new model is interpreted differently from the original model. In our hybrid model, which combines the deterministic and stochastic MAP estimate methods, the scattering reflectance variable r is no longer treated as an auxiliary variable but as the main objective variable. Using Bayes' theorem, the r posterior, which has y as the observed data, is defined as

$$p(r|y) \propto p(y|r)p(r) \tag{4.1}$$

$p(y|r)$ is the likelihood and it represents how the true image information r is degraded into y . It is defined in the same way as it was in [6] and has been given in Section 3.10. $p(r)$ is the prior on r . The statistical speckle model represents our knowledge on the scattering process mathematically. Rayleigh distribution in

¹We follow the notation that is used in the original model for consistency.

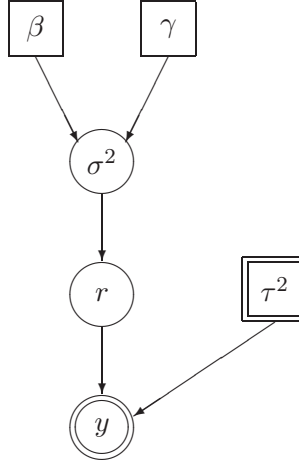


Figure 4.1: Graphical Model of statistical ultrasound image restoration. Fixed hyper-hyperparameters are represented with *boxes*. Observed variable is represented with a *double circle*. Unknown variables are represented with *circles*. Model parameters, but not hyperparameters, are represented in *double boxes*.

Equation 3.8 is selected in our model and is re-formulated as

$$p(r_{i,j}|\sigma_{i,j}^2) = \frac{r_{i,j}}{\sigma_{i,j}^2} \exp\left\{-\frac{r_{i,j}^2}{2\sigma_{i,j}^2}\right\} \quad (4.2)$$

Our belief in r is that by having the knowledge of σ^2 for each site (pixel), the reflectance value on each pixel becomes independent of all others. This is reasonable since r exhibits random characteristics, while σ^2 is highly related to the physical properties of the underlying medium, as described by the statistical scattering models. σ^2 is unknown and it is expected to be estimated as well as r . Since the variance variable σ^2 is introduced into the system by the speckle model, which acts as the prior on r , σ^2 is regarded as r 's hyperparameter. The assumption that given r , y and σ^2 are independent of each other is still valid here and is expressed in the graph by no direct arrow between y and σ^2 .

In order to complete the model, prior knowledge of the hyperparameter σ^2 is required. This has been discussed in Section 3.2.2 in the previous chapter. In short, noise should be suppressed at uniform regions and boundaries should be sharpened at interfaces. We use a non-quadratic prior on σ^2 which incorporates a variation of

the GM (Gemen & McClure) function, given in Equation 4.3. Figure 4.2 shows the two-dimensional plot of this function.

$$\phi(u) = \frac{\sqrt{|u|}}{1 + \sqrt{|u|}} \quad (4.3)$$

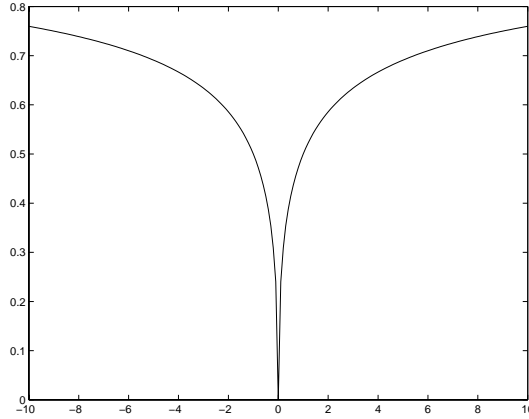


Figure 4.2: Plot of the prior density function in Equation 4.3.

This member of the edge-preserving prior family preserves the discontinuity and does not make the whole region over-smooth (see [55] and [56]). The prior probability density function on σ_2 is thus:

$$p(\sigma^2) \propto \exp \left\{ -\beta \sum_{i,j \sim k,l} \omega_{ij,kl} \phi \left(\frac{\sigma_{i,j}^2 - \sigma_{k,l}^2}{\gamma} \right) \right\} \quad (4.4)$$

4.3.2 JMAP Model Description

Next, we show how to fit our model into the joint MAP framework. In [57], a *joint maximum a posteriori* (JMAP) model is proposed. The basic idea here is to consider the hyperparameters at the same level as the main parameter. Further, the optimization can be implemented by successively maximizing the joint posterior with respect to the main parameter and the hyperparameters. This is very useful if there is more than one unknown variable in a Bayesian model.

Having the likelihood in Equation 3.10, the prior on r in Equation 4.2 and the prior on the hyperparameter σ^2 in Equation 4.4, the joint posterior on r and σ^2

is obtained as

$$p(r, \sigma^2 | y, h, \beta, \gamma, \tau^2) \propto \exp \left\{ - \sum_{i,j} \left(\frac{1}{2\tau^2} \sum_{i,j} \left(y_{i,i} - \sum_{k,l} h_{k,l} r_{i-k,j-l} \right)^2 + \frac{r_{i,j}^2}{2\sigma_{i,j}^2} \right) - \frac{1}{2} \sum_{i,j} (\ln r_{i,j} - \ln \sigma_{i,j}^2) - \beta \sum_{i,j \sim k,l} \omega_{i,j,kl} \cdot \phi \left(\frac{\sigma_{i,j}^2 - \sigma_{k,l}^2}{\gamma} \right) \right\} \quad (4.5)$$

Our problem states that given h (point spread function), y (observed image), β and γ (hyper-hyperparameters), and τ^2 (variance of additive noise), then r (reflectance) and σ^2 (variance of reflectance) are estimated from the joint posterior. This is formulated as

$$(\widehat{r}, \widehat{\sigma^2}) = \arg \max_{r, \sigma^2} (p(r, \sigma^2 | y, h, \beta, \gamma, \tau^2)) \quad (4.6)$$

The estimation on r and σ^2 is performed by an alternating iterative descent procedure. At iteration $n + 1$, $r_{(n+1)}$ is obtained by holding σ^2 at fixed $\sigma_{(n)}^2$. Then $\sigma_{(n+1)}^2$ is calculated from $\sigma_{(n)}^2$ and $r_{(n+1)}$. This is formulated formally as

$$\widehat{r}_{(n+1)} = \arg \max_r \{ p(r, \sigma_{(n)}^2 | y, h, \tau^2) \} \approx \arg \max_r \{ p(r | \widehat{\sigma}_{(n)}^2, y, h, \tau^2) \} \quad (4.7)$$

$$\widehat{\sigma}_{(n+1)}^2 = \arg \max_{\sigma^2} \{ p(\sigma^2, r_{(n+1)} | \beta, \gamma) \} \approx \arg \max_{\sigma^2} \{ p(\sigma^2 | \widehat{r}_{(n+1)}, \beta, \gamma) \} \quad (4.8)$$

Apparently, now we have two explicit, separate optimization problems to solve independently. Instead of using MCMC to sample from the posterior, a deterministic optimization method - Conjugate Gradient (CG) method is employed to restore the reflectance r , of which details are given in Section 4.4.2. The objective function of the variance field σ^2 is far from a quadratic form, which adds difficulties to linear optimization solvers. We choose to perform its estimation using the original MCMC procedure in [6], as presented in Section 4.4.3. In addition, to make this model work we need a preprocessing step to estimate the point spread function h from the observed image y . This is discussed in Section 4.4.1.

The model is setup so that it is well-suited for implementation on GPUs. We see the efficiency and benefit of this in Chapter 5.

4.4 Solutions to Three Steps

4.4.1 PSF Estimation using Homomorphic Transform

In [6], a *sine* modulated circular Gaussian function is used to approximate the PSF. Apparently, the complicated interaction between the tissue and the interrogating pulse adds difficulty to finding good parameters in that function and thus, to making a realistic approximation. There is a great need to estimate the PSF directly from the recorded ultrasound image data.

A two-dimensional homomorphic deconvolution method is proposed in [15] to improve the ultrasound image resolution. The approach belongs to the *Blind Deconvolution* category, which can reconstruct the PSF and deconvolve an image simultaneously. A spatially invariant point spread function is estimated by transforming the recorded image data and operating on it in different domains. Finally, the image is deconvolved by *Wiener filtering* in the frequency domain [15].

We exploit the homomorphic transformation idea. We have the estimated PSF in the frequency domain transform to the spatial domain by an inverse Fast Fourier Transform (IFFT). The resulting signal is regarded as the approximation of the point spread function. Instead of using the PSF in the deterministic deconvolution in the frequency domain as in [14], we apply a two-stage iterative restoration model on ultrasound images. The PSF obtained is used as pre-knowledge. To some extent, the error caused by the assumption in PSF in the Bayesian deconvolution model [6] is reduced. The basic idea behind this PSF estimation approach is given here and detailed descriptions can be found in [14].

The term *cepstrum* was introduced by Bogert et al. and is now an accepted terminology for the inverse Fourier transform of the logarithm of the power spectrum of a signal [58]. First we introduce some of its underlying concepts.

- *Two-dimensional Z-transform*

The 2D Z-transform takes discrete time domain signals into a complex-variable

frequency domain.

$$X(z_1, z_2) = Z \{x(m, n)\} = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x(m, n) z_1^{-m} z_2^{-n} \quad (4.9)$$

- *Two-dimensional Discrete Fourier Transform*

The 2D discrete Fourier transform is a special case of the 2D Z-transform evaluated on unit bi-circle:

$$X(\omega_1, \omega_2) = X(z_1, z_2)|_{z_1=e^{j\omega_1}, z_2=e^{j\omega_2}} = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x(m, n) z_1^{-m} z_2^{-n} \quad (4.10)$$

- *Two-dimensional Complex Logarithm Function*

For $z_1, z_2 \neq 0$, the complex logarithm function $\log(Z)$ is defined as

$$\log(z_1, z_2) = \ln |z_1, z_2| + i (\text{Arg}(z_1, z_2) + 2n\pi) \quad (4.11)$$

where n is an integer and $-\pi < \text{Arg}(z_1, z_2) < \pi$.

- *Two-dimensional Homomorphic Transform*

Homomorphic transform transforms a signal into its *cepstrum*. It is capable of converting a convolution in the spatial domain

$$y(m, n) = h(m, n) * x(m, n) \quad (4.12)$$

into a sum in the cepstrum domain

$$\hat{y}(m, n) = \hat{h}(m, n) + \hat{x}(m, n) \quad (4.13)$$

There are two important properties supporting the homomorphic transform.

1. The *Convolution theorem* tells us that the Fourier transform of the convolution of two signals is equal to the product of their individual Fourier transforms [59]. The convolution of two functions in the time domain is represented by

$$y(m, n) = h(m, n) * x(m, n) \quad (4.14)$$

Apply the Fourier transform on both sides and we have

$$\begin{aligned} F(y(m, n)) &= F(h(m, n) \star x(m, n)) \\ &= F(h(m, n))F(x(m, n)) \end{aligned} \quad (4.15)$$

2. Logarithm multiplication property

$$\log(xy) = \log(x) + \log(y) \quad (4.16)$$

Next, we show how to obtain a signal in the cepstrum domain using homomorphic transform. The image degradation process is modelled as $y(m, n) = h(m, n) \star x(m, n) + \tau(m, n)$. The additive noise term $\tau(m, n)$ has to be ignored in order to make the homomorphic transform work. First, a Fourier transform is performed. Utilizing the convolution theorem, we have

$$Y(\omega_1, \omega_2) = H(\omega_1, \omega_2)X(\omega_1, \omega_2) \quad (4.17)$$

Then, we do the complex logarithm on both sides

$$\log(Y(\omega_1, \omega_2)) = \log(H(\omega_1, \omega_2)) + \log(X(\omega_1, \omega_2)) \quad (4.18)$$

Finally, the inverse Fourier transform is exploited to finish the homomorphic transform, see Equation 4.19. In practice, we use an inverse fast Fourier transform (IFFT).

$$IFFT(\log(Y(\omega_1, \omega_2))) = IFFT(\log(H(\omega_1, \omega_2))) + IFFT(\log(X(\omega_1, \omega_2))) \quad (4.19)$$

Now the signals are in the cepstrum domain, represented as

$$\hat{y}(m, n) = \hat{h}(m, n) + \hat{x}(m, n) \quad (4.20)$$

An assumption can be made based on the physical properties of the system response h and the reflectance x . The PSF is a smooth and continuous function in space and its components should be fairly close to the pulse center frequency, [15]. On the

other hand, the tissue reflectors cover a long range of frequencies and should appear to be little peaks in the cepstrum domain. In other words, these two signals are assumed to be located at almost separable time bands in the cepstrum domain. This idea has been applied successfully in speech signal analysis, seismology and other areas. Based on this “separability” assumption, a low-pass filtering can be employed to retrieve the approximate PSF signal \hat{h} from the transformed image signal \hat{y} . This is also named a *liftering* operation, interchanging the letters in *filtering*.

After getting the desired estimated PSF in cepstrum, \hat{h} is transformed back by Fourier transform to the logarithmic Fourier domain. Then, following the inverse steps (exactly opposite to how the original signal is transformed to the cepstrum domain), the corresponding PSF signals in the ordinary frequency domain and the time domain can be obtained. They can be used in different deconvolution approaches, such as Wiener filtering and Bayesian models. Figure 4.3 shows the detailed steps for the PSF estimation process.

There exist certain limitations on this PSF estimation approach:

- Since the additive noise is not modelled explicitly, it may actually be amplified and corrupt the recovered signal. Later papers such as [14] discuss how to solve this problem in one dimension.
- The assumption of the reflective signal and the PSF signal being separably located is not totally true in the real world. Thus, the approach can only generate an approximate result since, inevitably, the overlapping signals cannot be recovered.
- This is a parametric method. There are two parameters that determine the cutoff values along axial and lateral directions, respectively, for the *liftering* operation in cepstrum. Their best values might vary for different probes and have an impact on the reconstructed signal. The determination of parameters are nontrivial.

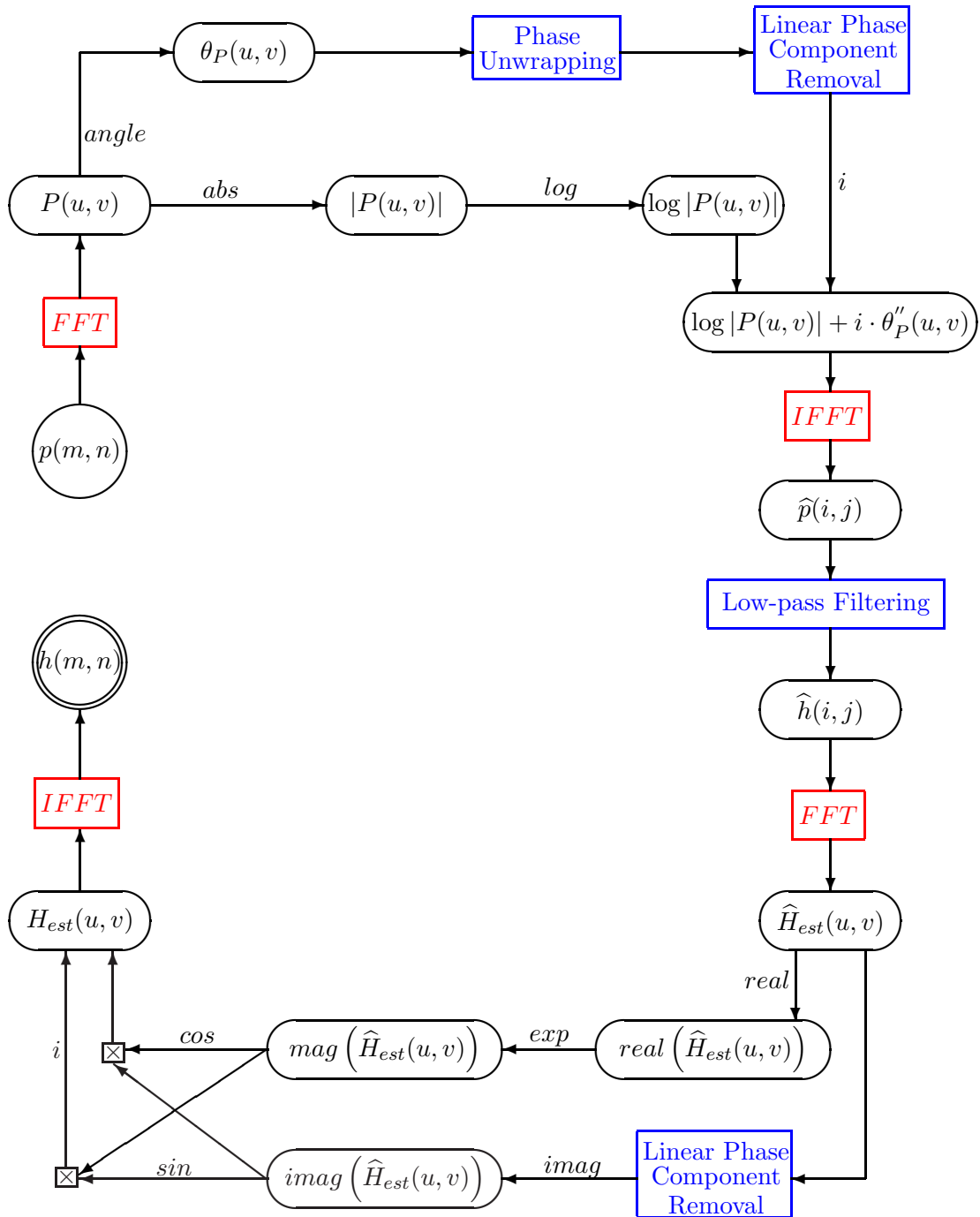


Figure 4.3: Flowchart estimating the PSF using cepstrum transforms. The input signal, which is the observed raw image, is represented with a *circle*. The output, which is the estimated PSF signal, is represented with a *double circle*. All the other intermediate variables are in *ovals*. *Boxes* represent major processing steps, each of which consists of many operations. *Arrows* represent data flow and the *operators* bound with them indicate the operations performed on the data.

Generally, there are many factors that make the separation of the PSF from the observed signals very hard. It is known that the estimation of the PSF is a field that needs more investigation. Our research does not focus on how to best recover the PSF but on how to use it as a preprocessing step. It is expected to be a good starting point for the iterative restoration model discussed later. We believe estimating PSF from the real image data instead of using parametric functions to approximate it is the best way to proceed.

Regarding the energy conservation, the L1 norm of the PSF equals one. Note that [15] deals with images in a polar coordinate system, which is the case when using sector scanners. In our implementation, we simply process images in a two-dimensional Cartesian coordinate system for linear array probes or phased array probes. Detailed descriptions of the theories of complex cepstrums of two-dimensional exponential sequences and two-dimensional phase unwrapping in the frequency domain can be found in [15].

4.4.2 Reflectance Estimation using CG Method

In this section, we show how to estimate r using a deterministic optimization method. From Equation 4.21, we know that r should be updated to maximize the conditional posterior

$$\hat{r}_{(n+1)} = \arg \max_r \left\{ p(r | \hat{\sigma}^2_{(n)}, y, h, \tau^2) \right\} \quad (4.21)$$

which is equal to maximize the *log* posterior

$$\hat{r}_{(n+1)} = \arg \max_r \log \left\{ p(r | \hat{\sigma}^2_{(n)}, y, h, \tau^2) \right\} \quad (4.22)$$

Referring to Equation 4.5, Equation 4.22 can be rewritten in a matrix-vector

form

$$\begin{aligned}\hat{r} &= \arg \min_r \left(\frac{1}{2\tau^2} (y - Hr)^T (y - Hr) + \frac{1}{2} r^T S r + Q(\widehat{\sigma^2}, \beta, \gamma) + L(r) \right) \\ &= \arg \min_r \left(\frac{1}{2\tau^2} (y - Hr)^T (y - Hr) + \frac{1}{2} r^T S r + L(r) \right)\end{aligned}\quad (4.23)$$

$$\approx \arg \min_r \left(\frac{1}{2\tau^2} (y - Hr)^T (y - Hr) + \frac{1}{2} r^T S r \right)\quad (4.24)$$

where S is a diagonal matrix

$$S = \begin{bmatrix} \frac{1}{\sigma_{11(n)}^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_{21(n)}^2} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma_{NN(n)}^2} \end{bmatrix}\quad (4.25)$$

If the two-dimensional image X consists of $N \times N$ pixels, r and y are both one-dimensional column vectors of length $N^2 \times 1$. They are constructed by unwrapping the rows of the corresponding two-dimensional variables R and Y , and concatenating them end to end in a column. More specifically, these vectors are given by

$$r = (r_{11}, \dots, r_{N1}, r_{12}, \dots, r_{N2}, \dots, r_{NN})^T\quad (4.26)$$

$$y = (y_{11}, \dots, y_{N1}, y_{12}, \dots, y_{N2}, \dots, y_{NN})^T\quad (4.27)$$

$Q(\sigma^2, \beta, \gamma)$ represents the last two terms in Equation 4.5. The second equality comes from the fact that $Q(\sigma^2, \beta, \gamma)$ is independent of r , thus, it can be ignored in the optimization process. $L(r)$ is equal to $\frac{1}{2} \sum_{i,j} \ln r_{i,j}$ and in order to use linear optimization approaches, this term is ignored. This does not cause problems since the logarithm term contributes much less to the objective function, compared to the second order exponential term, or even to the first order term. The approximate equality in Equation 4.23 is obtained by getting rid of the $L(r)$ term.

H is the convolution matrix, which is a block Toeplitz matrix with Toeplitz blocks (BTTB), assuming we use zero-padding boundary conditions. Both matrices

H and S are of dimension $N^2 \times N^2$. The entries in H are determined by the elements in the PSF h , and H is very ill-conditioned.

Since we take the negative of Equation 4.22, the objective function on r which we try to minimize is

$$\begin{aligned}
J(r) &= \frac{1}{2\tau^2} (y - Hr)^T (y - Hr) + \frac{1}{2} r^T S r \\
&= \frac{1}{2\tau^2} (y^T - r^T H^T) (y - Hr) + \frac{1}{2} r^T S r \\
&= \frac{1}{2\tau^2} (y^T y - r^T H^T y - y^T H r + r^T H^T H r) + \frac{1}{2} r^T S r \quad (4.28)
\end{aligned}$$

$$\begin{aligned}
\because r^T H^T y &= (r^T H^T y)^T = y^T H r \\
\therefore J(r) &= \frac{1}{2\tau^2} (y^T y - r^T H^T y - y^T H r + r^T H^T H r) + \frac{1}{2} r^T S r \\
&= \frac{1}{2\tau^2} y^T y - \frac{1}{\tau^2} y^T H r + \frac{1}{2} r^T \left(\frac{1}{2\tau^2} H^T H + S \right) r \\
&= \frac{1}{2\tau^2} y^T y - \frac{1}{\tau^2} r^T H^T y + \frac{1}{2} r^T \left(\frac{1}{2\tau^2} H^T H + S \right) r \quad (4.29)
\end{aligned}$$

$J(r)$ is a quadratic function and matrix $(\frac{1}{2\tau^2} H^T H + S)$ is a symmetric positive definite matrix.

The Conjugate Gradient (CG) method is an effective iterative method for minimizing a quadratic form with a symmetric positive definite matrix. Successive approximations to the solution are generated. Instead of sequentially updating each site in the image, which is how the MCMC method behaves, using vector-matrix notation makes paralleled updating possible. We show later that the CG method can be implemented efficiently on GPUs.

The classical Conjugate Gradient method is given in Figure 4.4, in which the matrix $(\frac{1}{2\tau^2} H^T H + S)$ is represented as matrix A , vector $(\frac{1}{\tau^2} H^T y)$ is notated by l , and e is the residue vector. Each iteration requires two inner products of N-vectors and one multiplication of the coefficient matrix with an N-vector.

```

Set initial guess  $r^0 = l$ .
Compute  $e^0 = l$ .
for  $i=1,2,\dots$ 
     $\rho_{i-i} = e^{(i-1)T} e^{i-1}$ 
    if  $\rho_{i-i} = 0$ 
        method fails
    if  $i = 1$ 
         $p^{(1)} = e^{(1)}$ 
    else
         $b_{i-1} = \frac{\rho_{i-1}}{\rho_{i-2}}$ 
         $p^{(1)} = e^{(i-1)} + b_{i-1}p^{(i-1)}$ 
    endif
     $q^{(i)} = Ap^{(i)}$ 
     $\alpha_i = \frac{\rho_{i-1}}{p^{(i)T} q^{(i)}}$ 
     $r^{(i)} = r^{(i-1)} + \alpha_i p^{(i)}$ 
     $e^{(i)} = e^{(i-1)} - \alpha_i q^{(i)}$ 
    check convergence; continue if necessary.
end

```

Figure 4.4: Pseudocode of the Conjugate Gradient (CG) method, from [7].

4.4.3 Variance Estimation using MCMC

The estimation equation for σ^2 is given as

$$\widehat{\sigma}_{(n+1)}^2 = \arg \max_{\sigma^2} \{p(\sigma^2 | \widehat{r}_{(n+1)}), \beta, \gamma\} \quad (4.30)$$

which is equal to

$$\begin{aligned} \widehat{\sigma}_{(n+1)}^2 &= \arg \max_{\sigma^2} \log \{p(\sigma^2 | \widehat{r}_{(n+1)}), \beta, \gamma\} \\ &= \arg \min_{\sigma^2} \left(\sum_{i,j} \left(\frac{1}{2} \ln(\sigma_{i,j}^2) + \frac{\widehat{r}_{i,j}^2 (n+1)}{2\sigma_{i,j}^2} + \beta \sum_{i,j \sim k,l} \omega_{ij,kl} \phi \left(\frac{\sigma_{i,j}^2 - \sigma_{k,l}^2}{\gamma} \right) \right) \right) \end{aligned} \quad (4.31)$$

Image restoration priors or penalties are dedicated to force pixel values to be similar to their neighbors. Geman and Geman first propose using Gibbs distributions on priors in image restoration [60]. Gibbs priors have many variances with different interaction functions that model the correlation among the spatially neighboring

pixels. Since [60], Gibbs priors have been investigated for various aspects, such as the development of the variations in [61], comparison of different variations in [62] and parameter estimation in [63] and [64]. As discussed in Section 3.2.2, prior knowledge on σ^2 requires the same belief as a common true image, and thus the well-studied Gibbs priors are applicable on σ^2 .

We investigate the exploitation of quadratic priors and non-quadratic priors in our model. If a quadratic prior is used to define the $\phi(\cdot)$ function, the objective function on σ^2 is very close to a quadratic form. Its vector expression is

$$J(\sigma^2) = \frac{1}{2}I_r \log(\sigma^2) + \frac{1}{2}r^T S r + \frac{\beta}{\gamma^2} (\sigma^2)^T W \sigma^2 \quad (4.32)$$

where both r and σ^2 are $N^2 \times 1$ column vectors. The diagonal matrix S has been defined in Equation 4.25. The other variables are defined below.

$$I_r = [1, 1, \dots, 1] \quad (4.33)$$

$$W = \begin{bmatrix} 8 & -2 & 0 & 0 & \dots & -2 & 0 & 0 & \dots & 0 \\ -2 & 8 & -2 & 0 & \dots & 0 & -2 & 0 & \dots & 0 \\ 0 & -2 & 8 & -2 & \dots & 0 & 0 & -2 & \dots & 0 \\ 0 & 0 & -2 & 8 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -2 & 0 & 0 & 0 & \dots & 8 & -2 & 0 & \dots & 0 \\ 0 & -2 & 0 & 0 & \dots & -2 & 8 & -2 & \dots & 0 \\ 0 & 0 & -2 & 0 & \dots & 0 & -2 & 8 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 8 \end{bmatrix} \quad (4.34)$$

Basically, I_r is an identity vector that performs the reduce operation on the other vector. Matrix W expresses the association between each pixel and its neighbors.

For the first order neighborhood system, W is formulated as

$$\begin{aligned}
 W &= \{w_{i,j} | 1 \leq i, j \leq N^2\}, \text{ with } (i, j) \text{ element} \\
 W_{i,j} &= \begin{cases} -2, & \text{if } (i - j) = \pm 1, \text{ or } (i - j) = \pm N, \\ 8, & \text{if } (i = j), \\ 0, & \text{otherwise} \end{cases} \quad (4.35)
 \end{aligned}$$

The Conjugate gradient method can be exploited to optimize the conditional posterior function $J(\sigma^2)$. However, as known from theory and proved by our experiments, this quadratic prior tends to make the image region smooth everywhere. This is caused by the function penalizing the differences of neighboring pixels at increasing rates, making it unable to sharpen the boundaries between different regions.

Therefore, we choose to use the edge-preserving prior given in Equations 4.4 and 4.3. However, by doing this, the objective function can no longer remain a beautiful quadratic form, and the non-convex function presents difficulties in computing the global MAP estimation. We cannot use a gradient-based optimization method to find the minimum anymore. Our solution is to embed the Monte Carlo approximation optimization method, which belongs to the stochastic optimization in our model. We follow the σ^2 estimation procedure as proposed in [6]; one major reason is that it favors GPU architecture and can be implemented efficiently on graphics hardware, which is shown in the following chapter.

The detailed steps for estimating σ^2 using the *Independent Metropolis-Hastings* algorithm can be referenced in Section 3.2.5. Two modifications are made:

1. Instead of exploiting the second order Markov Random Field neighborhood system, we use the first order neighborhood system because of its better parallel properties and can be performed in two updating passes.
2. The probability function on σ^2 is a bit different since we choose a different prior, given in Equation 4.31.

One issue that needs to be pointed out is that the results generated by MCMC analysis are actually a sequence of values sampled from the posterior distribution. It is commonly believed that after a burn-in period, the posterior mean should be used as the final estimate. These accumulations take time. As suggested in [6], in real time applications, after the burn-in period the samples for each pixel at one iteration can be used instead of the mean. Either method can be implemented efficiently on a GPU and which option to use depends on how fast the response is required by the specific applications.

4.5 JMAP Model Discussion

4.5.1 Modification Aspects

Our model is developed based on the model presented in the proceeding chapter. In summary, the differences of our model and the original model are listed as follows:

- Instead of treating σ^2 as the target variable and introducing r as the auxiliary variable, we think of r as the objective variable and σ^2 as r 's hyperparameter.
- The PSF is no longer fixed by a certain function but is estimated from the observed image directly.
- The reflectance field is no longer estimated by MCMC sampling from the conditional density functions. It is solved using a Conjugate Gradient method in a deterministic way.
- The Rayleigh scattering model is exploited as the prior on r instead of Gaussian distribution.
- The Gibbs prior on σ^2 is regarded as the prior of the hyperparameter and it is of a different form.

4.5.2 Parameter Configuration

There are two free parameters β and γ in the σ^2 prior. They are both positive and exert a big influence in the resulting images. Good estimates of these two parameters have been investigated extensively using both *Maximum Likelihood* (ML) algorithms [64] and *Expectation Maximization* (EM) algorithm [65]. Almost all available methods are dedicated to models where β and γ are treated on the hyperparameter level. However, they are regarded as hyper-hyperparameters of the hyperparameter σ^2 in our model. This carries β and γ one level further from the data y , which can be seen clearly in Figure 4.1. Thus, the parameter estimation process, already complicated, becomes harder to solve. For simplicity, β and γ are fixed by pre-computed values using trial-and-error in our experiments.

The noise variance τ^2 is another parameter. Since the additive noise has less effect, compared to the blurring operator, it is fixed as a small number as well. Throughout our experiments, we set it as 1.

4.5.3 EM Algorithms

This problem can be solved by the *Expectation Maximization* (EM) algorithm. EM learns the parameters that give the log likelihood the highest probability in the model with unknown variables. However, it would be very computationally expensive for the following reasons:

- In the E step, we need to calculate the expectation of the log likelihood within the valid independent variable range, which is any number between 0 and 255 for image processing problems.
- The objective images have high dimensions of data. The EM algorithm is known to be an expensive algorithm when applied to these problems and it is not suitable for online applications.

Chapter 5

Implementation on the GPU

5.1 Streaming Implementation Overview

General iterative image processing on CPU programs, regardless of the programming languages used, can be represented by the abstract structure in Figure 5.1.

```
for iter = 1...
  for i = 1...m
    for j = 1...n
      Operation 1;
      Operation 2;
      ...
      check convergence;
    end
  end
end
```

Figure 5.1: Abstract structure for CPU image processing programs.

Image data are stored in two-dimensional arrays whose elements can be accessed by coupled indices (i, j) within valid range. Since the CPU is designed for sequential code, the pixels have to be processed one at a time in each iteration. For some models, the pixels require the same operations and can be treated independently in one pass. In other words, there is significant parallelism that cannot be utilized efficiently in the CPU structure above.

When our JMAP model is mapped into a streaming model and implemented on GPUs, it has a totally different implementation structure. The outer two loops controlled by i and j are realized by rendering a quadrilateral, which is the only type of primitive passed to the graphics pipeline. Fragment programs perform the operations inside the loops. Textures represent the two-dimensional images and arrays.

The screen frame buffer for display only provides 8 bits for each color component. This brings trouble to multi-pass rendering. If the outputs from one pass are written into this buffer, they get clamped to $[0, 1)$. Thus, we are not able to store numbers outside this range that are necessary in computations. Our targets are, therefore, rendered to an off-screen pixel buffer (*pbuffer* for short), which supports more bits for colors. *Pbuffers* are additional rendering buffers allocated in non-visible frame buffer memory for an OpenGL renderer. Details on how to use off-screen *pbuffer* in OpenGL can be found in [66].

There are two commonly-used OpenGL options to feed outputs from one pass to following passes, which is required by the outermost loop in Figure 5.1. They are *render-to-texture* (RTT) and *copy-to-texture* (CTT). The CTT strategy involves rendering an image to a *pbuffer* and copying the contents of that *pbuffer* to a texture. RTT binds the *pbuffer* to a texture object and thus avoids data copying. However, the current RTT scheme suffers from the overhead associated with switching between rendering contexts, recognized as a bottleneck in applications. This is inherently caused by the OpenGL API, while the DirectX API does not suffer from such problems. Since our images are of relatively small resolutions, we choose to use CTT. In our experience, this does not significantly slow down the whole performance.

All computations are carried out on the GPU, whereas the CPU only provides a high level of control ensuring that the desired fragment programs are executed in proper sequence. All the operations are specified in the vertex and fragment

programs. All initial data, intermediate data and resulting data are stored in on-board video memory on the graphics card.

5.2 Solutions for Efficient Mapping on GPU

There are several issues that affect the performance of GPU programs.

- If possible, reading the data in the video memory back to the CPU should be avoided because this can still stall the pipeline and cause a critical performance bottleneck.
- The model should be of a streaming style, which is able to continuously feed data to multiple, component-parallel fragment units.
- The number of components in the target being processed should be minimized. This is the starting point for the four color packing strategy on grey-scale images.
- The number of operations specified and the number of registers used in each fragment should be minimized.

The exploitation of those general strategies, the solutions to some specific problems, and steps involved in our model when mapped onto GPUs are discussed in the following sections.

5.2.1 Fast Median Search

The candidate value for the variance field is calculated using the median of the neighbors of the current pixel being updated. This can be done efficiently in a fragment program. The *Swizzle* operator is defined by referencing the components of a 4-vector in random order. Its use is usually free, due to its direct support in the hardware.

The first order Markov Random Field neighborhood system is employed in our model. By its definition, every pixel has four neighbors. Here, the median of even numbers of elements is defined to be the average of the two middle elements. Devillard shows the fastest way to find the median of odd numbers of elements, especially for short sequences in [1]. Extending this idea, the optimal C code to compute the median of 4 elements is shown in Figure 5.2.

```

typedef float pixelvalue ;

#define PIX_SORT(a,b)
        if ((a)>(b)) PIX_SWAP((a),(b));
#define PIX_SWAP(a,b)
        pixelvalue temp=(a); (a)=(b); (b)=temp;

pixelvalue opt_med4(pixelvalue p)
{
    PIX_SORT(p[0], p[1]) ;
    PIX_SORT(p[2], p[3]) ;
    PIX_SORT(p[0], p[2]) ;
    PIX_SORT(p[1], p[3]) ;
    return ((p[1] + p[2])/2) ;
}

```

Figure 5.2: Extended fastest C code to search the median of four elements, from [1].

On GPUs, the median of four values is computed by four *comparison* operations and four *swaps*. Four neighbors are stored in \mathbf{V} , a 4-element vector, and *swap* is done by the efficient *swizzle* operator. This forms a contrast to the CPU implementation, where moving data between different channels requires additional instructions [37]. Figure 5.3 gives the corresponding fragment program code.

The *median* is a very useful statistical term in image processing and graphics applications. Based on the results in [1], Table 5.1 gives the minimum numbers of operations needed to find the median in short sequences. These numbers cannot be

```

if (V.x > V.y)
    V.xyzw = V.yxzw;
if (V.z > V.w)
    V.xyzw = V.xywz;
if (V.x > V.z)
    V.xyzw = V.zyxw;
if (V.y > V.w)
    V.xyzw = V.xwzy;
median = (V.y + V.z)/2;

```

Figure 5.3: Fragment program code to search the median of four elements.

reduced without prior knowledge of the data. Although we only experiment with

Length	Comparison	Swap	Mean
3	2	2	0
4	4	4	1
5	7	7	0
7	13	13	0
9	19	19	0

Table 5.1: Minimum numbers of operations to find the median in short sequences, from [1].

finding the median out of 4 elements on GPUs, the idea could be easily extended to more numbers. Due to the frequent use of the *median*, it could also be a potential function that becomes a hardware built-in function in the future. We believe this could possibly be utilized to improve the speedup of many applications.

5.2.2 Four Color Packing

The GPU is a streaming processor designed mainly for efficient 4-vector computations. Most of the shader instructions in 4-vector form are hardware accelerated. For grey-scale images, one value is sufficient to represent the intensity of every pixel. If four pixels can be condensed into one, whose four color channels represent those

four pixels, a speedup might be achieved. This is for two reasons: one, the reduction of the rendering image size; two, the exploitation of instructions optimized for 4-element vectors.

The idea of using color packing to maximize the utilization of vectorized fragment hardware is not new. [41], [37], and [45] have all mentioned the advantages of this special optimization strategy. This only works for grey-scale images or for grid-based scientific computations. Images with more than one color component, however, cannot be represented in this way.

There are several ways to pack the four color components:

1. The most straightforward way is to cut the whole image into four blocks: LU(left upper), LL(left lower), RU(right upper), and RL(right lower). Each of them can be used in one color channel, shown in Figure 5.4. For an image with $M \times N$ pixels, where we assume M and N are even numbers¹, we come up with a packed image of size $(M/2) \times (N/2)$. This strategy is suitable for models that do not require accessing neighboring pixels in fragment programs.

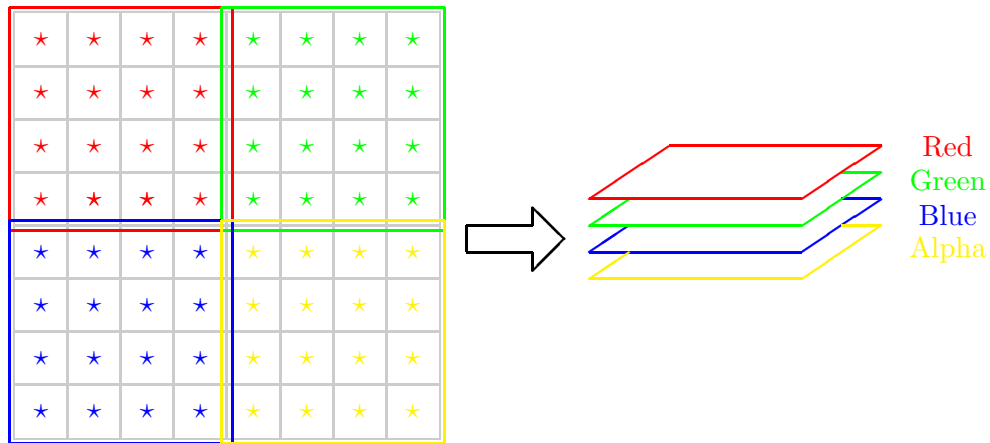


Figure 5.4: Color packing by four blocks of the same size. The resulting image has the same size as each block.

¹If these conditions are not satisfied, we can always pad zeros at two adjacent boundaries.

2. The four blocks separation strategy can be combined with domain decomposition algorithms if only a small number of neighbors need to be accessed. The basic idea is to add extra borders on each block to keep proper pixel values across other blocks. If iterative passes are required, after each iteration, the border values need to be updated in extra passes, in order to maintain consistency. This has been exploited successfully in the sparse matrix solver in [45] where a detailed explanation can be found.
3. The third way is to pack four adjacent neighbors into one pixel. This is illustrated in Figure 5.5. This was employed for the pressure field in cloud simulation in [41].

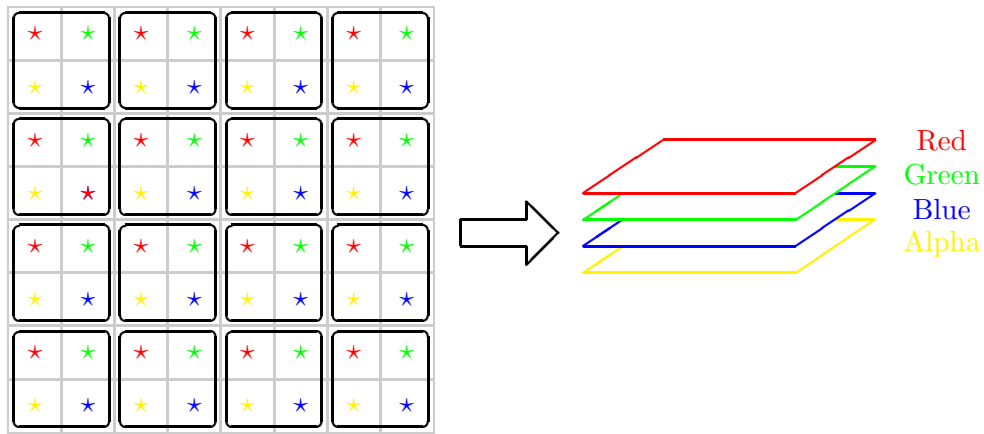


Figure 5.5: Color packing by 4 adjacent neighbors.

One of the major operations in our model is convolution. This packing strategy is the best for our problem since neighboring pixels are essential for convolution. The idea of neighbor color packing combined with four shifted PSFs leads to our fast convolution algorithm and is discussed in Section 5.2.3.

4. Under certain circumstances, packing can be done in the direction that the computation follows. Cartesian matrix-matrix multiplication and the GPU-based matrix multiplication algorithm proposed in [36] are both good examples

of this. Each pass involves the *dot* product of one row vector $A_1(i, :)$ from the first matrix A_1 with one column vector $A_2(:, j)$ from the second matrix A_2 . Every four consequential elements are packed into one pixel both on $A_1(i, :)$ and $A_2(:, j)$. After packing, the size of A_1 is altered from $M \times N$ to $M \times N/4$ and that of A_2 is changed from $N \times M$ to $N/4 \times M$. This is shown in Figure 5.6.

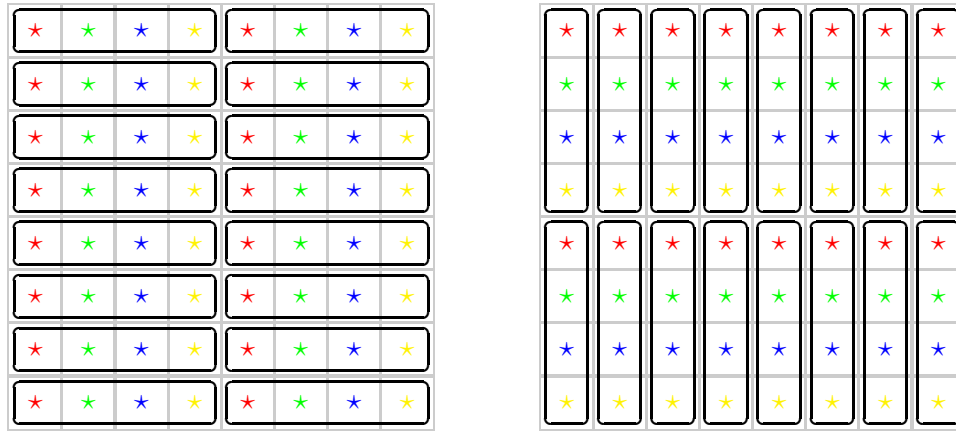


Figure 5.6: Color packing by vectors. The matrix on the left is packed along the row direction and the matrix on the right is packed along the column direction. Each packed fragment is shown in one black solid box.

From our experiment, the speedup from color packing is shader-dependent. It relies on how well the vectorization can be utilized by packing. If the four color components do not have the same operations, they cannot be processed simultaneously within one single instruction, which means the acceleration is not achieved. In this circumstance, the number of instructions inside the fragment program after packing increases. In the worst case, it could have four times more instructions than the original fragment program. Usually longer fragment programs slow down the program. In addition, the number of registers used by the program might increase as well, which is one of the key factors that worsen the GPU performance. The comparison of unpacked and packed convolutions is shown in Table 5.5 together with the OpenGL implementation comparison in the following section.

5.2.3 Fast Convolution with Large Kernels

One of the most expensive steps of using the Conjugate Gradient method is the multiplication of the Hessian matrix $(\frac{1}{2\tau^2}H^TH + S)$ with the target variable r represented as a vector. In image restoration applications when the blurring operation is assumed to be a convolution, the Hessian matrix has better structure than the common dense or sparse matrices. With the assumption that the PSF is symmetric, H is a symmetric matrix and the non-zero entries on each row in H are all the entries in PSF. Under zero boundary conditions, H is actually a block Toeplitz matrix with Toeplitz blocks (BTTB). Back to the two-dimensional representation of the column vector x , Hx can be implemented as the 2D image X convolved with the point spread function F . Because H is symmetric, H^THx is equal to $H(Hx)$ which requires two passes of convolutions. Convolution can be performed efficiently by multiplication using Fourier transforms. [2] investigates implementing Fourier transforms on GPUs using a special indexing scheme. The performance is shown in Table 5.2.

Image Size	Arithmetic (sec)	Texture Lookup (sec)
1024 ²	1.9	0.60
512 ²	0.44	0.13
256 ²	0.09	0.03
128 ²	0.01	0.007

Table 5.2: FFT performance, from [2]. The program transforms an image into the frequency domain using FFT; then a low-pass filtering is then performed there. Finally, it is transformed back to the time domain using IFFT.

[2] also reports their experiment on the same sequence of operations using the FFTW library on a 1.7 GHz Intel Zeon CPU. If the target image has 1024×1024 pixels, it takes about 0.625 second. The conclusion can be drawn that the Fourier transforms are still not fast enough compared to the FFTW library on a typical CPU. One of the benefits of using FFT to perform convolution is that it does not depend on kernel size. For the brute force implementation of convolution, the running time is

linear in the convolution kernel size if the image size is fixed. As a consequence, FFT can compete with brute force implementation when doing convolution with large kernels. We show later that provided our convolution kernel size is around 11×11 or a little larger, the FFT version convolution on a GPU is no faster than our implementation. Therefore, we do not implement it using the FFT.

Since the textures are color-packed, we need to be careful when we specify a pixel to access its neighbors. For each color component, the other three color components are all its neighbors and the outlying neighbors are located in other packed pixels. We need to control the texture coordinate offsets and fetch those neighbor texels² properly from the texture where the image data are stored. However, the four components in one pixel have different neighbors, shown in Figure 5.7.

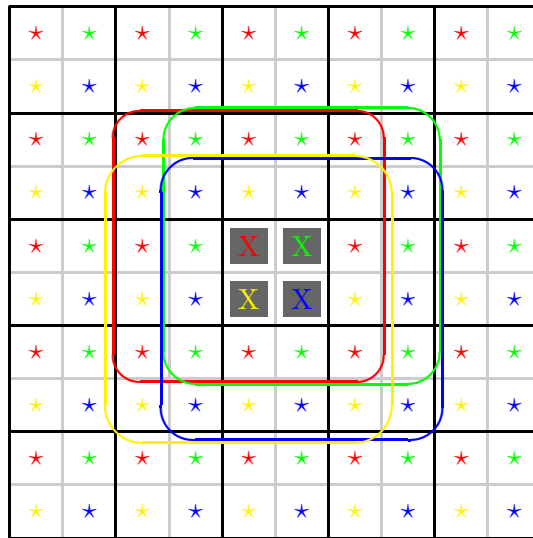


Figure 5.7: Four color components in one pixel, represented by four colorful X s, have different neighbors.

A special approach is necessary to improve the time consuming conditional check that determines if a color component is a valid neighbor. We develop a four

²A texel (texture component) represents the smallest graphical element in two-dimensional textures. A texel is similar to a pixel (picture component) because it represents an elementary unit in a graphic [67].

shifting PSF texture scheme to address this. The main advantage is that we do not need to specify different operations for different color channels R, G, B, and A respectively. Instead, we create four special PSF textures by padding zeros at different boundaries and making them of even dimensions, as shown in Figure 5.8.

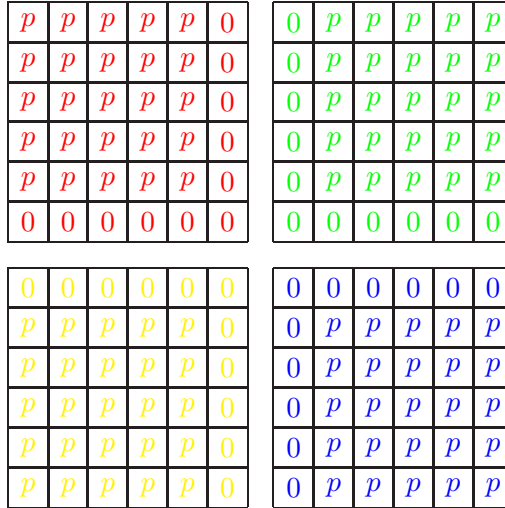


Figure 5.8: Four PSF textures with proper zero paddings. Here we use a 5×5 kernel for simple illustration.

When we fetch the neighbors, we can get the four color components in one pixel together and do the computation in vector form. There is no need to do component masking in order to determine which colors can be used as real neighbors. The programmable fragment processor provides a *dot* instruction. It calls the machine instruction *DP*, which is accelerated by the hardware. It takes two vectors as inputs and returns their dot product. Using the four PSF textures after zero padding, the entries in the convolution kernel are matched to the corresponding texels in space perfectly, and thus, the four component *dot* function can be exploited. The PSF textures as well as the original image texture are color-packed, as shown in the left plot in Figure 5.9. The right plot in Figure 5.9 presents how the original neighborhood is extended to facilitate the zero-padded PSF textures and perform fetching of the four color components. The excess time we spend with this strategy is taken

up by meaningless computations done for zero borders in PSF textures.

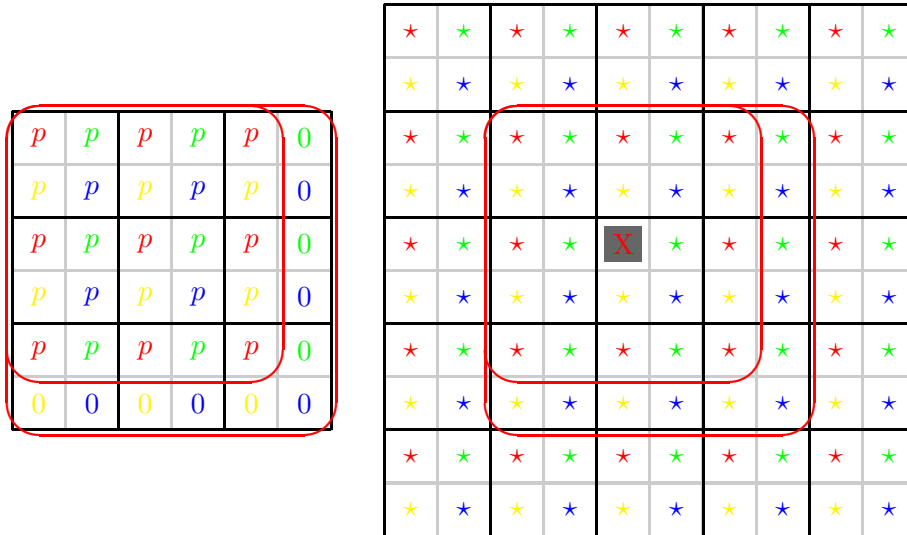


Figure 5.9: Left: red PSF texture is color-packed as well. Right: extended PSF texture helps to eliminate conditional checking inside each fragment of which ones are neighbors, and therefore, four colors can be fetched together. Notice the matching of the red PSF texture on the left with the extended supporting region on the right, represented by the solid, rounded rectangle.

With an optimized Cg code (here *optimization* is defined as using less registers and less instructions), Table 5.3 shows the number of instructions and registers required by the convolution fragment programs. Now, all the operations and data use 16-bit floating point precision. This is because exploiting 32-bit floating point precision uses up the limited temporary registers³ If that happens, the fragment program cannot be compiled at all. Notice that this brings up the insufficient precision issue, but this can be overcome by the four pass rendering scheme discussed below.

Currently, the NVIDIA programmable GPU has an upper limit of 1024 instructions per pass, per fragment program. ATI programmable GPUs are restricted to a maximum of 96 instructions at the fragment program stage. For this reason, it

³There are 32 32-bit floating point temporary registers available for the fragment program on an NVIDIA NV3X GPU.

Kernel Size	Instructions	R Regs.	H Regs.
5X5	213	1	6
7X7	316	1	6
9X9	496	1	6
11X11	716	1	6
13X13	911	1	6

Table 5.3: Number of instructions and registers for convolution with different kernels. These numbers are generated by the Cg compiler.

is impossible to perform convolution within one pass for a kernel larger than 13×13 , approaching the 1024 limit, as can be seen in Table 5.3. One possible solution is to split the fragment program into several shorter fragment programs and render them in multiple sequential passes. The precondition is for it to be separable.

In our application, the original fragment program is split based on color channels and turns into four shorter programs. Each of them performs the computation for one color channel. The intermediate values are stored in a temporary texture and passed into the next fragment program as a sample texture. The limitation on the maximum instruction number actually brings better performance. This is due to the fact that shorter programs usually perform faster than longer programs if they use comparably many temporary registers. All the computations and texture fetching are in 32-bit floating point format. This reduces potential large precision errors caused by large numbers being operated, and by the accumulation of errors after passes.

The split of long fragment programs is illustrated in Table 5.4. Here we assume that the convolution kernel is of size 21×21 . Each of the resulting fragment programs has about 600 instructions. For each fragment program, we give the fragment program complexity. This concept is taken from [68] and is defined by three elements: number of instructions (Instrs.), number of 32-bit registers (R Regs.), and number of 16-bit registers (H Regs.). *Updated* means this color component is updated by the convolution of the corresponding texel with its neighbors. *Copy*

means the output color value for this channel is copied from the intermediate texture, which stores the result from the last pass.

Pass	Red	Green	Blue	Alpha	Instrs.	R Regs.	H Regs.
1	Updated	Set 0	Set 0	Set 0	609	5	0
2	Copy	Updated	Copy	Copy	610	5	0
3	Copy	Copy	Updated	Copy	610	5	0
4	Copy	Copy	Copy	updated	610	5	0

Table 5.4: Instruction and register counts for the four pass convolution fragment programs with kernel size 21×21 .

The OpenGL extension *GL_ARB_imaging* subset provides a set of convolution functions but these functions perform slowly. In addition, it has a restriction that the convolution kernel cannot be larger than 11×11 . In Table 5.5, we compare the performance of convolution using kernels of different sizes using OpenGL and our approach. Figure 5.10 shows the corresponding two-dimensional plot. In the unpacked fragment programs, operations are only specified on the RED channel. In the packed fragment programs, the four grey-scale color values of four pixels are packed into the four color channels of one pixel and four passes are executed for each color in order to reduce the number of registers and instructions per pass.

Table 5.5 compares the performance of convolution operations of a variety of kernel sizes in different implementation approaches. After experimenting with the original unpacked fragment programs and the packed program using shifted PSFs in four passes, we observe an obvious speedup of the packed version over the unpacked one. Our approach is the only one that can actually perform convolutions with larger kernels. For kernels smaller than 21×21 , this implementation exceeds the FFT approach, which needs approximately 11 (millisecond) for a 128×128 image. Convolution is one of the most basic and important computations. Our efficient implementation makes it more feasible for use in real time applications.

Kernel Size	OpenGL	Unpacked	Packed A	Packed B	Packed C
3X3	5.6645	1.7355	1.1646	0.6107	0.4321
5X5	7.0384	3.7787	2.2880	1.0058	0.9272
7X7	10.4863	11.9086	3.9163	1.7218	1.3168
9X9	14.0530	19.7468	5.5380	2.5974	2.0223
11X11	18.3767	30.5951	10.9064	3.9335	2.3599
13X13	N/A	41.5251	14.8333	5.0574	4.2794
15X15	N/A	55.3487	19.2270	6.9209	5.7498
17X17	N/A	N/A	24.0334	8.8185	7.3428
19X19	N/A	N/A	29.4244	10.5905	9.4710
21X21	N/A	N/A	35.7147	12.6700	12.4644
23X23	N/A	N/A	42.3802	15.5078	13.9722
25X25	N/A	N/A	49.7555	17.6770	15.7803
27X27	N/A	N/A	57.7851	21.1844	18.7953

Table 5.5: Convolution running time (in milliseconds) comparison. The target image has 128×128 pixels. The numbers given in *Packed A*, *Packed B*, and *Packed C* columns are experimental results on an NVIDIA FX5600 card, an NVIDIA FX 5800 card, and an NVIDIA Quadro3000 card, respectively. Here *N/A* means these operations cannot possibly be implemented either because they are not supported (for the OpenGL column) or because of the 1024 maximum instruction limitation on fragment programs.

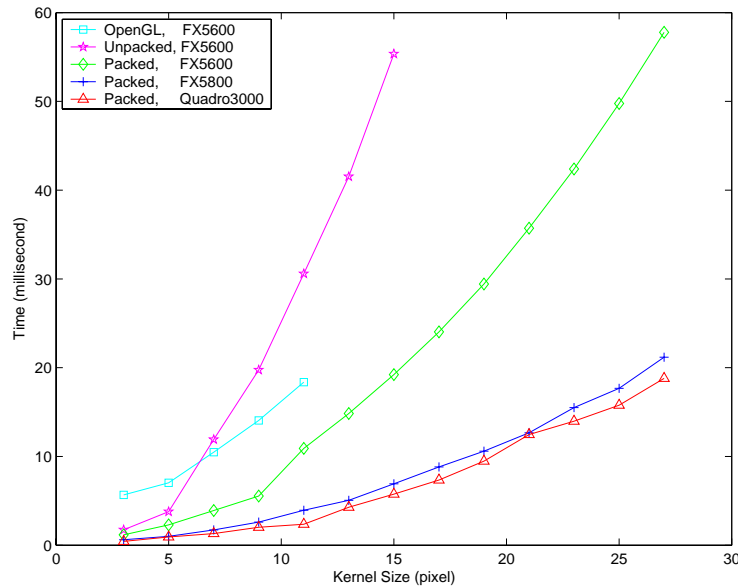


Figure 5.10: Plot on the convolution time comparison in Table 5.5.

channels, is created using the data in the array. At each iteration, two random numbers, **randOffsetX** and **randOffsetY**, are generated on the CPU and passed as parameters into the fragment programs requiring random numbers. These two random numbers act as the texture coordinate offsets of **random2Texture**. When passing through the fragment processor, each fragment reads the random number from **random2Texture** using the texture coordinates; these are the sum of the original texture coordinates and the offsets.

Another possible approach, the Perlin Noise, is widely used in graphics simulation to generate high dimensional noise using an affordable amount of data and computation [69]. One-dimensional permutation tables and a pre-generated two-dimensional noise texture are required. We have not experimented with this yet.

5.2.5 Two Passes Updating a Markov Random Field

Inherently, the MCMC sampling algorithm requires sequential site updating. When a variable is modified, all the available new variable values are supposed to be used. Here, new values are defined to be the most up-to-date for all the other sites. It seems that this scheme conflicts with the GPU's parallel architecture. We find it hard to map the MCMC algorithm with large supporting neighbors onto GPUs. This is the main reason we turn to the Conjugate Gradient method to estimate the reflectance variable, instead of using MCMC to sample from the posterior, as discussed in Section 4.1.

However, the variance field has a small neighborhood supporting area. Two commonly used two-dimensional Markov Random Fields neighborhood systems can be shown as follows.

1. *First order neighborhood system*

Each site has only four neighbors with equal influence on the center site. We may set the weights of all these four neighbors to be the same: 1.

2. *Second order neighborhood system*

	X	
X	O	X
	X	

Figure 5.12: First-order Markov neighborhood system.

Each site has eight neighbors. The sites marked with an **X** in the following table are called orthogonal neighbors, and those marked with a **W** are called diagonal neighbors. Usually, orthogonal neighbors have weights 1 and diagonal neighbors are assigned $1/\sqrt{2}$ weights.

W	X	W
X	O	X
W	X	W

Figure 5.13: Second-order Markov neighborhood system.

We use the first order MRF neighborhood system, so the whole domain can be updated efficiently in two passes, shown in Figure 5.14 . This is based on the partial parallel algorithm that performs synchronous updates using independent set partitioning. From Figure 5.13, we can observe that the **O**s only have **X**s as their neighbors; thus, **O**s are independent of each other. Therefore, the sites marked with an **O** can be updated simultaneously and this also works for all the sites marked with an **X**. In the first pass, **O**s are updated with the knowledge of **X**s. In the second pass, **X**s are computed while fixing the **O**s. This is also discussed in [41].

We have two fragment programs for the two rendering passes. Without packing the four color components, at each pass two comparisons are performed to determine whether this fragment should be updated or not. The comparisons are for the coordinates along both x and y directions. If not, its color values are simply copied from the texture to the frame buffer and get updated in the second pass. This conditional check slows down the program and makes it inefficient. This

X	O	X	O
O	X	O	X
X	O	X	O
O	X	O	X

Figure 5.14: Parallel updating within first-order Markov neighborhood system.

is because real branching is not supported in current fragment processors. What really happens with branching instructions in the hardware is that the branch code is actually executed for every fragment and conditional codes are used to suppress the unwanted results [68]. This can be overcome by color packing.

After color packing, in each pass two color components are updated, either *Red and Blue* or *Green and Alpha*. It does not matter which set gets updated first. The advantages of using color packing are significant: the total number of fragments are reduced, and the conditional checking operation per fragment is eliminated, since they are explicitly determined by the two fragment programs.

5.2.6 Display Unpacked Image

Internally, all the textures being rendered represent compact images after four color packing. Since all of them have a common type, the type remains consistent in all the rendering passes. After several iterations, we need to display the resulting image on the screen and the internal image needs to be unpacked to the original layout. In order to do this, two passes are required.

1. After the offscreen pbuffer context is disabled, the device context and window context are passed back to the current rendering target. The screen frame buffer is of the same size as the original image and two times larger than the offscreen pbuffer, both along the horizontal and vertical directions. The four color components of each texel in the packed texture actually correspond to four pixels in the unpacked image. In this pass, similar to up-sampling, the

color values of each packed texel are copied to the four square neighboring sites. The texture coordinates are recomputed in order to give the correct mapping without conditional checks. A new texture is stored after this pass.

2. In the second step, each pixel selects an appropriate color value from the associated RGBA components passed from the first pass. A simple *modulate* operation is used for the selection. In order to show a grey scale image, that color component is copied to the other three color channels as well.

5.2.7 Boundary Padding

As mentioned in Chapter 4, a zero padding boundary condition is used in the model. The original image is embedded into a larger image, whose extra entries are filled with zeros. If the PSF is of size $psfWidth \times psfHeight$, the widths of the top and bottom extra boundaries are $\lfloor psfHeight/2 \rfloor$. The widths of the left and right boundaries are $\lfloor psfWidth/2 \rfloor$. During the rendering, the zeros are never updated and only the center part is rendered, by properly controlling the texture coordinates.

This corrects the error caused by repeatedly accessing the borders, which is a texture accessing mode provided by OpenGL. Without zero padding, if the program accesses a texel whose coordinate is outside the valid range, it does not return a zero as desired. Instead, it keeps on reading the border with width 1 and uses those values in the computation. We observe serious artifacts with this rigid, off-texture accessing. With zero padding, the problem no longer exists.

If zero-padded textures and no-zero-padded textures both act as input in the same fragment program, different texture coordinates are required to fetch desired corresponding pixels from those textures. A vertex program is exploited to generate those coordinates to realize multi-textures. This fits the streaming style processing framework. With zero extensions, the quadrilateral with the original size is rendered and all the pixels can still perform the same operations without having to specify different computations on those boundary-related pixels.

The zero boundary condition is simple to implement, with satisfying results. Goodnight et al. discuss how to implement periodic boundary conditions in [68].

5.3 Summary and Discussion

5.3.1 Shader Summary

We have 12 textures to operate with. Figure 5.15 presents the fragment programs and rendering passes in the reflectance estimation initialization step. Each row corresponds to a rendering pass, except for the *reduce* operation, which contains multiple passes according to the image resolution. Texture columns of blue color are padded with zero-borders, with widths determined by the PSF. Textures in red are of dimension 1×1 . The remaining black textures are normal two-dimensional textures with no borders. Entries marked with **I** in the tables mean the corresponding texture is the *Input* of this program and its values are fetched as the operators in the fragment program. **O** means the *Outputs* of the fragment program after this pass are written into this texture. Technical details are shown in the following sections.

Shader		L	R	V	E	P	Q	T	TEMP	PSF	
psf* l → temp	R	I			O					I	
	G	I			IO					I	
	B	I			IO					I	
	A	I			I			O		I	
psf*(psf*r) → e	1	R		I			O			I	
		G		I			IO			I	
		B		I			IO			I	
		A		I			IO			I	
	2	R				O		I			I
		G				IO		I			I
		B				IO		I			I
		A				IO		I			I
-(e-temp) → e					IO		I				

Figure 5.15: Textures and fragment programs in the reflectance estimation initialization step. The textures are specified by the capital letters on the first row. The operations in the fragment programs are given in the left column.

Shader	L	R	V	E	P	Q	T	TEMP	PSF	ρ_{new}	ρ_{old}	s	B	pbuffer
r.*r → temp				I				O						
								IMO						
pbuffer(0,0) → ρ_{new}									O					I
$\rho_{new}/\rho_{old} → b$										I	I		O	
$\rho_{new} → \rho_{old}$										I	O			
e+b.p → p				I	IO								I	
1					I	IO			I					
					I	IO			I					
					I	IO			I					
					I	I	O		I					
psf*psf*p → q			I		I	O	I		I					
			I		I	IO	I		I					
			I		I	IO	I		I					
			I		I	IO	I		I					
p.*q → temp					I	I		O						
								IMO						
pbuffer(0,0) → s											O		I	
$\rho_{new}/s → s$										I	IO			
r+s.p → r		IO			I							I		
e-s.q → e				IO		I						I		

Figure 5.16: Textures and programs in the reflectance estimation step.

Shader	X	V	Random
Update σ^2 (Pass 1)	I	IO	I
Update σ^2 (Pass 2)	I	IO	I

Figure 5.17: Textures and programs in the variance σ^2 estimation step.

Figures 5.16 and 5.17 give the programs and textures in the iterative reflectance and variance estimation stage respectively. Table 5.6 summarizes the numbers of instructions, registers and passes used by each fragment program.

Shader	Passes	Instructions	R Registers	H Registers
Pixelwise Multiplication 1	1	3	2	0
Reduce	7	7	3	0
Update B	1	5	2	0
Update ρ	1	2	1	0
Update P	1	4	3	0
Convolution stage1	1	180	5	0
	1	181	5	0
	1	181	5	0
	1	181	5	0
Convolution stage2	1	180	5	0
	1	181	5	0
	1	181	5	0
	1	181	5	0
Pixelwise Multiplication 2	1	3	2	0
Update s	1	8	2	0
Update R	1	4	3	0
Update R	1	4	3	0
Update σ^2 (Pass 1)	1	388	9	1
Update σ^2 (Pass 2)	1	389	9	1

Table 5.6: Shader summary. The complexity for each fragment program is acquired from the Cg compiler and shown here, complementing the texture-shader relationship tables. All the images are rendered with 128×128 pixels.

5.3.2 Current Limitations

From our experience with programming the graphics hardware, and from the opinions of other researchers', we gather that certain aspects of GPU architecture can

be improved to make it suitable for more general computations. The corresponding limitations are these:

- Need for an added *reduction* operator.
- Extremely slow performance for long fragment programs on the first pass, due to the optimization of the hardware driver.
- Absence of scattering operation.
- Lack of texture fetching with offset.
- Deficiency of higher precision required for some applications.
- Need to reduce the penalty on read back.
- Certain resource limitations on maximum number of instructions, available registers, active textures and texture lookups in each fragment program.

5.3.3 Algorithm Mapping Discussion

The GPU architecture favors algorithms that can fit in with a streaming model of computation. Algorithms that can be vectorized often exhibit parallelism, which makes them good candidates for mapping onto GPUs. Algorithms that require sequential updating are not suitable for mapping onto GPUs, such as the MCMC estimation of the reflectance field presented in Chapter 3. This is because users have no control over the fragment processing order on current parallel fragment processor architectures.

However, in certain circumstances, some applications using MCMC do have paralleled implementation possibilities. Parallel methods for MCMC are investigated in [70] and [71]. We consider the *Geographical Splitting* in [70] and the block partition method in [71], and propose a re-structured MCMC model for the GPU. We discuss how the MCMC algorithm with a first order neighborhood system can

be implemented on GPUs in two passes, and how this model can be applied with higher order neighborhood systems.

The idea is illustrated in Figure 5.18. Without sacrificing generality, suppose each site is dependent on the 25 sites surrounding it, including itself. It is obvious that the sites marked with the same numbers can be updated simultaneously since they are independent of each other. In the first pass, all the sites marked with 1 are updated. In the second pass, all the sites marked with 2 are updated and so on. All the sites can be processed once after 25 passes without destroying the correct correlation among them. If using the original structure indicated by the upper grid in Figure 5.18, each pass in each fragment requires a conditional check to determine whether it needs to be updated. This causes inefficiency. On the other hand, the data can be re-structured into the lower grid in Figure 5.18. In this way, all the sites that can be updated in the same pass are organized together into a block. In each pass, a quadrilateral covering the corresponding block is neatly rendered.

This is practical for MCMC applications which meet the condition of a relatively small neighborhood. This means they have better parallelism and require fewer passes to update the whole domain. We have not applied this model in practice yet. But since the GPUs have a parallel fragment processor architecture, it is possible that they can provide good performance on parallel MCMC models.

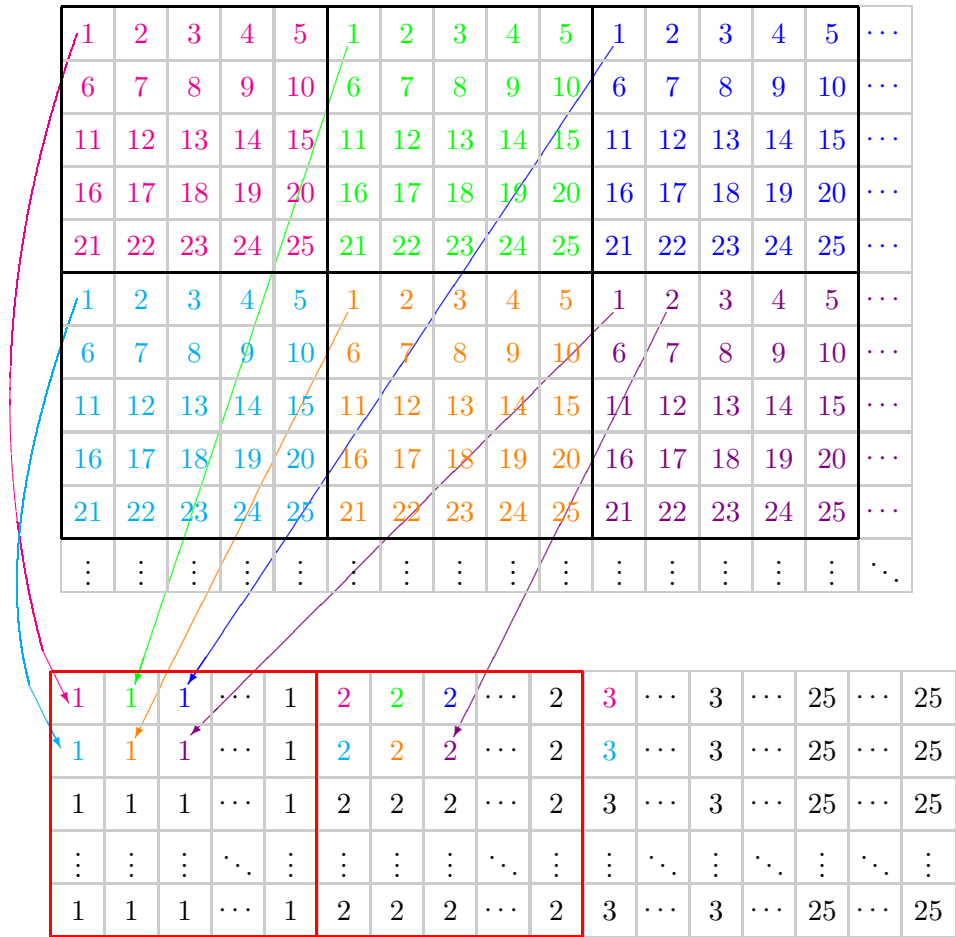


Figure 5.18: Restructured MCMC model on GPUs.

Chapter 6

Experiments

6.1 Experiments on Synthetic Images

6.1.1 Synthetic Image Generation

1. Synthetic variance field

All the images in this experiment contain 128×128 pixels. We start with an original image, Figure 6.1(a), in which the pixel intensities represent the variance field. There are two types of simulated tissue regions, with relative intensities of 38 and 50, respectively. We intend to create two types of regions that are not very different from each other, as indicated by their parameters. Since the original variance values are small, which makes the features hard to observe, rescaled images with higher contrast are shown as well (see Figures 6.1(b) and 6.1(c)). Through out this chapter, such images will accompany the original signal images for better visualization.

2. Synthetic reflectance field

The reflectance field is created following the Rayleigh distribution which models ultrasound speckle by relating reflectance to variance. The variance σ represents the mean μ of the distribution. The association between the Rayleigh

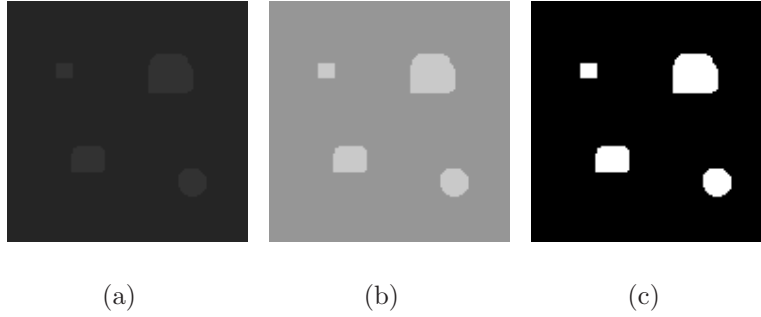


Figure 6.1: Simulated variance field. (a) Image with original intensity. (b) Image of the scaled intensity by a factor of four. (c) Image rescaled using the maximum and minimum intensity values.

distribution parameter s and its mean μ is given by

$$\mu = s\sqrt{\frac{\pi}{2}} \quad (6.1)$$

Thus, given the mean, the distribution parameter s is determined by the inverse of the above equation, leading to $s = \mu\sqrt{\frac{2}{\pi}}$. Based on the s parameter field, the reflectance field in Figure 6.2 is sampled from the Rayleigh distribution:

$$P(r) = \frac{r}{s^2} e^{-\frac{r^2}{s^2}} \quad (6.2)$$

The resulting reflectance field image is shown in Figure 6.2.

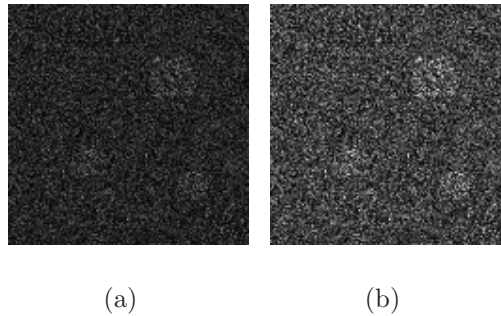


Figure 6.2: Simulated reflectance field. (a) Image with original intensity. (b) Image rescaled using the maximum and minimum intensity values.

3. Synthetic point spread function

The simulated PSF is modelled as a modulated cosine function, shown in Figure 6.3. It is of size 11×11 . Considering energy conservation, the sum of all the PSF entries equals one.

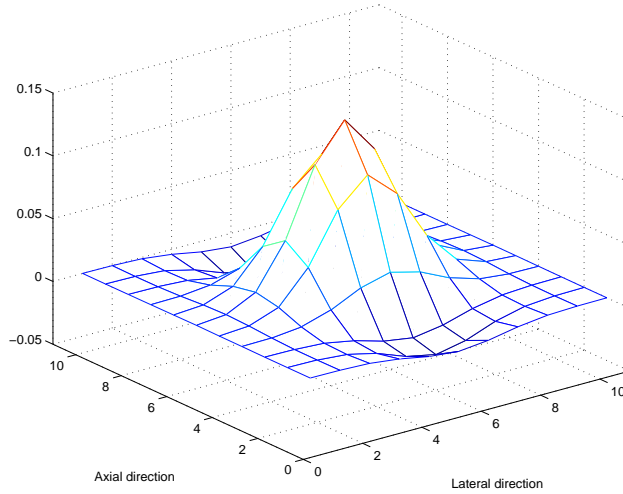


Figure 6.3: Simulated point spread function.

4. Synthetic degraded image

The reflectance field in Figure 6.2 is convolved with the simulated PSF in Figure 6.3. The zero boundary condition is enforced. Gaussian white noise with variance 1 is added to the reflectance field to form the final simulated ultrasound image, shown in Figure 6.4. This is the image from which we estimate the PSF, reflectance, and variance.

In Figure 6.2, the regions are not as observable as they are in the σ field. This is consistent with our expectation, since it is known that the blurring process may effectively conceal small structures. This adds difficulty to diagnosis based on degraded images. Here, the boundaries are not easily seen, and extra speckles are also generated in originally homogeneous regions. We show the pixel intensities on the 43rd and 26th horizontal scanlines in the image

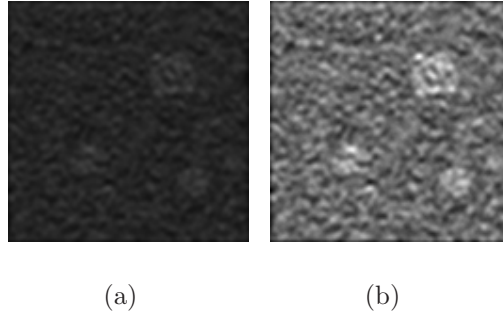


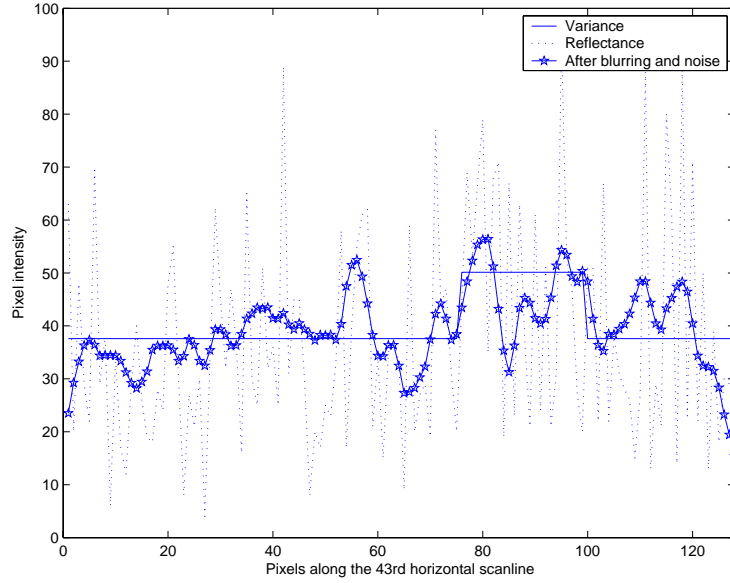
Figure 6.4: Simulated ultrasound image. (a) Image with original intensity. (b) Image rescaled using the maximum and minimum intensity values.

through different simulation stages in Figure 6.5. They illustrate the difficulty of recovering true underlying structures from blurred signals.

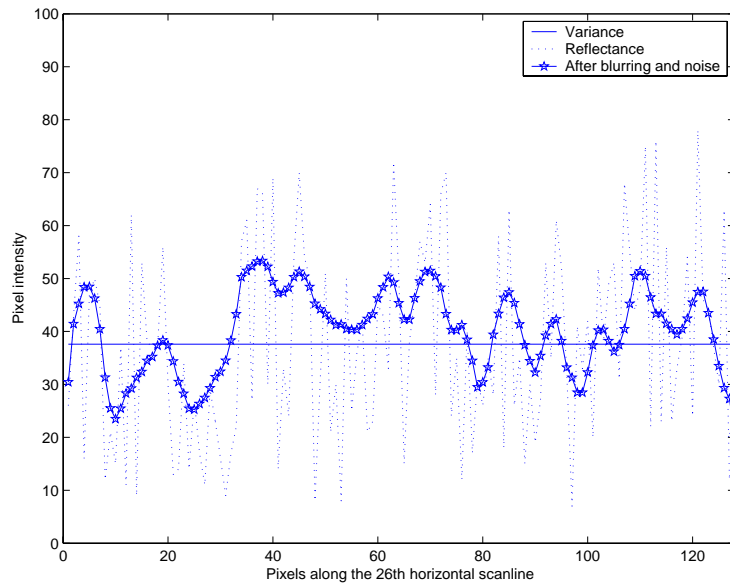
Looking at the original variance signal represented by the solid line in Figure 6.5(a), we can see it pass through two different structures. The medium is mostly homogeneous and the protuberance on the right represents a dissimilar region with higher intensity. The reflectance signal follows the Rayleigh distribution and is given by the strongly fluctuating dashed curve. The observed signal is shown in the solid curve with pentagrams. The observed signal is smoother than the reflectance, but still varies irregularly and severely. Thus it is hard to tell the actual bumped region from the observed image.

While Figure 6.5(a) shows loss of information after degradation by concealing real structures, Figure 6.5(b) illustrates the introduction of speckles by the scatterers and blurring process. This scanline passes through a homogeneous region represented by a straight, solid line. However, pixels with relatively low intensities compared to surrounding pixels, continuously appear from pixel 5 to pixel 50. This makes them seem to comprise a different region in the observed image. Actually, they do not true and these are extra speckles.

These artifacts all express the desired properties of our synthetic images, for



(a)



(b)

Figure 6.5: Comparisons of the pixels along (a) the 43rd scanline, and (b) the 26th scanline in different stages.

which we aim to include the fake information brought by the diffuse scattering, blurring and noise.

6.1.2 Estimated Point Spread Function

We implement the PSF estimation program using a package called FFTW, according to the steps presented in Figure 4.3. FFTW is a library written in C, performing discrete Fourier transforms (DFT). It performs significantly better than other DFT software [72]. Figure 6.6 shows the estimated PSF, very similar to the actual PSF used in convolution, given in Figure 6.3.

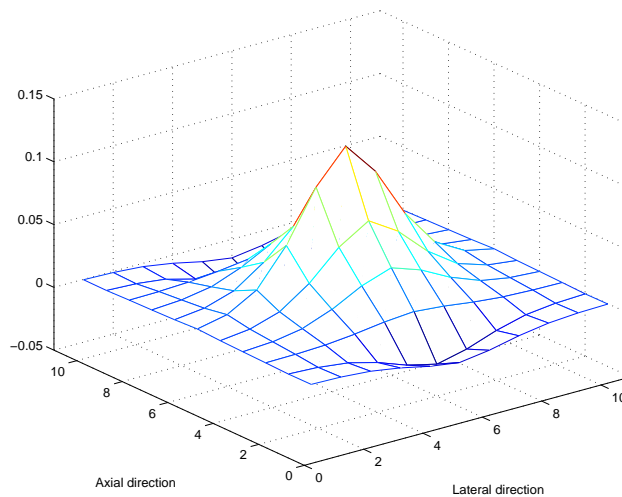


Figure 6.6: Estimated PSF from Figure 6.4.

6.1.3 Speed Performance

Our program is run on an NVIDIA Quadro 3000 GPU. It has a core frequency of 500MHz and an 850MHz memory data rate (SDRAM DDR2). It has an 8 pixel pipeline. The main program is written in C++ with the OpenGL API. All the GPU shaders are written in the NVIDIA Cg shading language. The OpenGL Extension Wrangler Library (GLEW) is exploited.

Table 6.1 gives the running time measured at each step while estimating the reflectance r . Table 6.2 shows the time for estimating the variance σ^2 . The associations of these abstract steps with actual fragment programs and textures can be found in Section 5.3.1. Table 6.3 summarizes the two tables and shows the total time per iteration.

Step	Running Time
Dot_Product_EE	0.147253
Copy_ ρ_{new}	0.013024
Update_B	0.029984
Update_ ρ	0.029395
Update_P	0.071425
Double_Convolution	5.666500
Dot_Product_PQ	0.142672
Copy_S	0.009498
Update_S	0.030663
Update_R	0.071160
Update_E	0.072314

Table 6.1: Running time (in milliseconds) for each step for estimating r .

Step	Running Time
Update_V_Pass1	2.299708
Update_V_Pass2	2.341989

Table 6.2: Running time (in milliseconds) for each step for estimating σ^2 .

Stage	Running Time
Update_R	6.625445
Update_V	4.618510
Total per Iteration	11.493057

Table 6.3: Running time (in milliseconds) for each stage and also the total time of each iteration.

The time spent on estimating the point spread function is not included in the total time since it is an one-time expense and is implemented on the CPU, not our main focus. The first iteration is not included in the final timing because

it is not the actual computation time spent on running the shaders but contains extra time in system initialization. This is caused by the hardware driver as well as the Cg compiler optimization. It has been confirmed by NVIDIA developers that “The driver will often delay some initialization such as downloading textures and compiled fragment programs until the first time something is drawn.” Fortunately, this is done only once at the beginning. It is tolerable for processing sequences of images, after the shaders are executed on the GPU.

The posterior traces of r and σ^2 are plotted in Figures 6.7(a) and 6.7(b), respectively. The updates of r and also the visual effect of r change little after 20 iterations, but σ^2 needs many more iterations to generate desirable and stable results. The updates on σ^2 take little time compared to time spent on r . Our strategy is to run all the shaders for about 20 iterations to update r and σ^2 in an alternating way. Then the r no longer changes but is used as fixed in updating σ^2 for more iterations until a satisfactory σ^2 estimate is obtained.

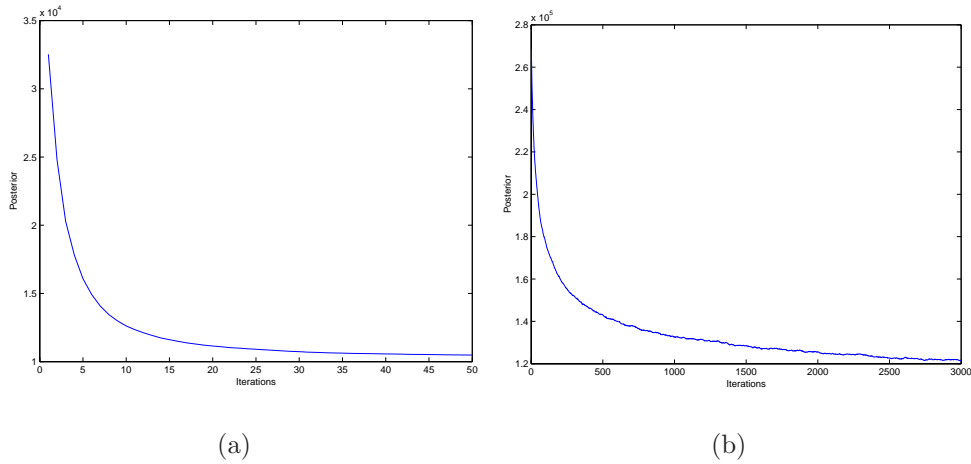


Figure 6.7: Convergence plots on (a) r and (b) σ^2 .

6.1.4 Resolution Performance of Estimated Reflectance Field

There are many judging criteria on which to evaluate whether image quality has improved. It is commonly believed that the reduction in the speckle size implies resolution increase. A two-dimensional auto-covariance function is used in [15] and [14] in order to quantify the increase in resolution; we use it here to evaluate our estimated reflectance field. Figure 6.8 shows the reflectance field after deconvolution.

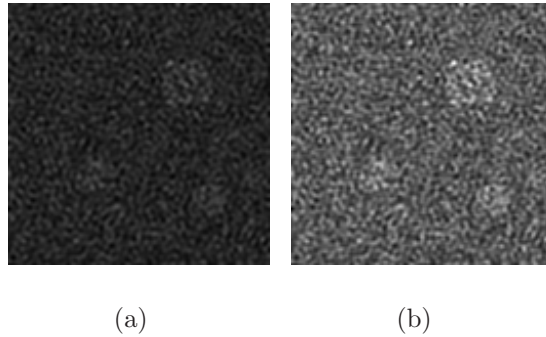


Figure 6.8: Estimated reflectance field. (a) Image with original intensity. (b) Image rescaled using the maximum and minimum intensity values.

The boundaries between different tissues are more easily observed compared to the unprocessed image in Figure 6.2. The reduction in widths of the auto-covariance plots along both axial and lateral directions is shown in Figures 6.9(a) and 6.9(b), respectively.

6.1.5 Resolution Performance of Estimated Variance Field

The estimated variance field after 3000 iterations is given in Figure 6.10. Parameters β and γ are set to be 6 and 2, respectively. These numbers are obtained by trial and error. Visually, the result is very similar to the original variance field shown in Figure 6.1, from which the reflectance and blurred images are generated. The underlying features are significantly clearer for two reasons: one, boundaries between different

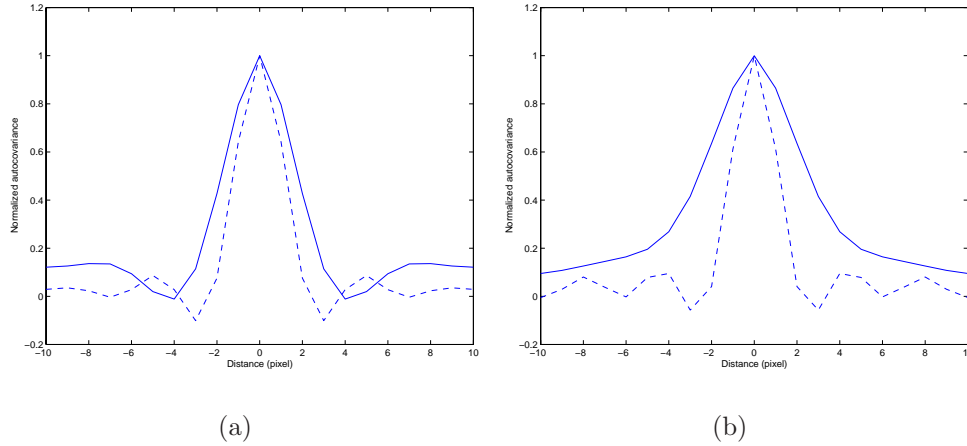


Figure 6.9: Comparisons on auto-covariance plots on unprocessed image envelope and deconvolved image along (a) the axial direction, and (b) the lateral direction. The solid curves show the auto-covariances on the original image and the dashed curves show the results after deconvolution. The peak value of auto-covariance is normalized to 1.0.

regions are much more distinguishable; two, noise and speckles are dramatically reduced at blurred regions that are originally homogeneous. The conclusion must be that the variance field is successfully reconstructed by the inverse model, which incorporates the edge-preserving prior.

However, the small square which locates in the upper left part of the hypothetical variance field is not recovered in the resulting variance fields. Its original dimension is 9×9 , which is even smaller than the dimension of the PSF. As a consequence, its feature is totally lost after the degradation, which can be seen from Figure 6.4. Thus there is no way to restore it.

Another drawback is that small artifacts might be introduced in the results, as can be seen in Figure 6.10. One reason is that the blurring process introduces some large speckles. In addition, the random variance field initialization has influence on the results. The parameters β and γ are able to control on this but there is always a tradeoff on how well the small structures can be kept and the speckles can be reduced.

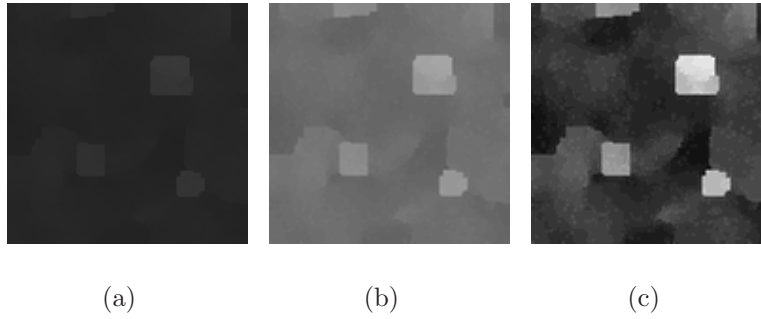


Figure 6.10: Estimated variance field after 3000 iterations. (a) Image with original intensity. (b) Image of the scaled intensity by a factor of three. (c) Image rescaled using the maximum and minimum intensity values.

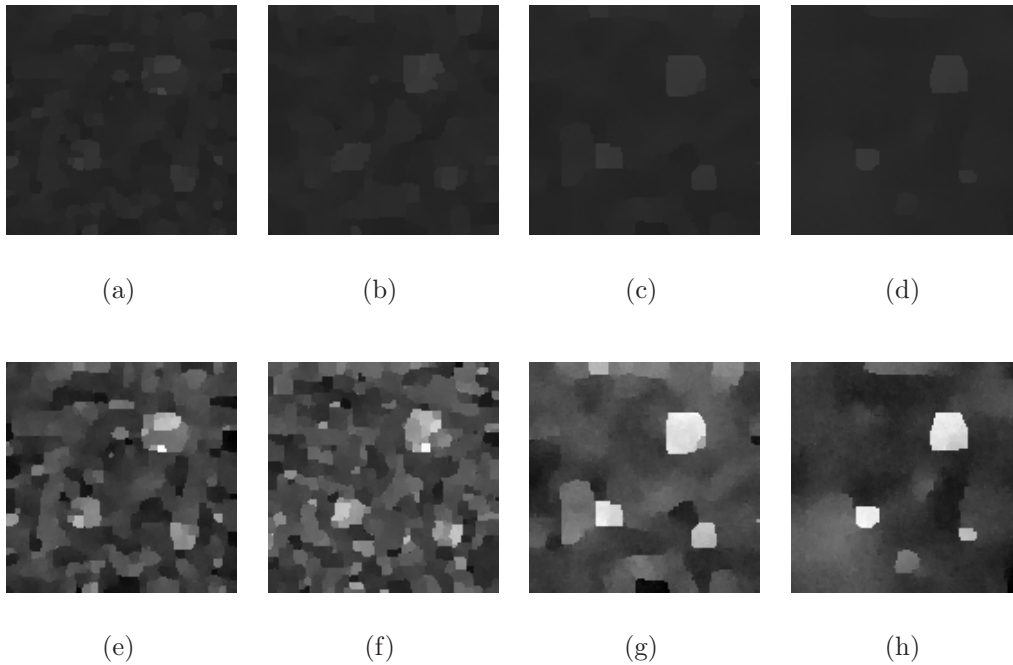


Figure 6.11: Estimated variance field after (a) 500 iterations; (b) 1000 iterations; (c) 2000 iterations and (d) 6000 iterations. (e), (f), (g), and (h) are the corresponding scaled images, by a factor of three.

We also show the resulting variance fields in Figure 6.11, obtained by stopping the Markov Chain after different numbers of iterations. We can easily see that the

estimation after 3000 iterations shown in Figure 6.10 is superior to those in Figure 6.11.

We compare the one-dimensional transects through images from different simulation stages. Figures 6.12, 6.13, and 6.14 show the pixel intensities along one particular scanline in images. The resulting variance signals, which are represented by the solid curves with pentagrams, are estimated from badly degraded signals in dashed curves. They are close to the original signals in solid curves. Homogeneous regions are relatively smooth while places where edges occur appear to be comparatively sharp.

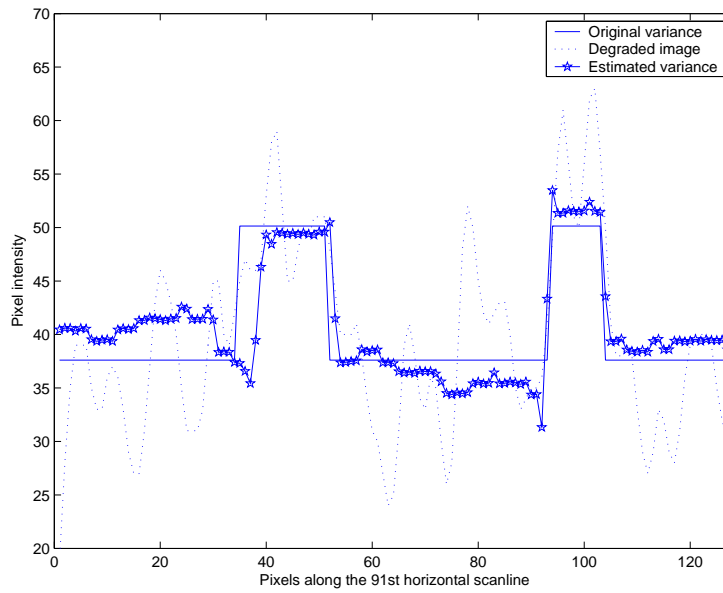


Figure 6.12: Pixels comparison on the 91st rows in the original variance field, blurred image, and estimated variance field, respectively.

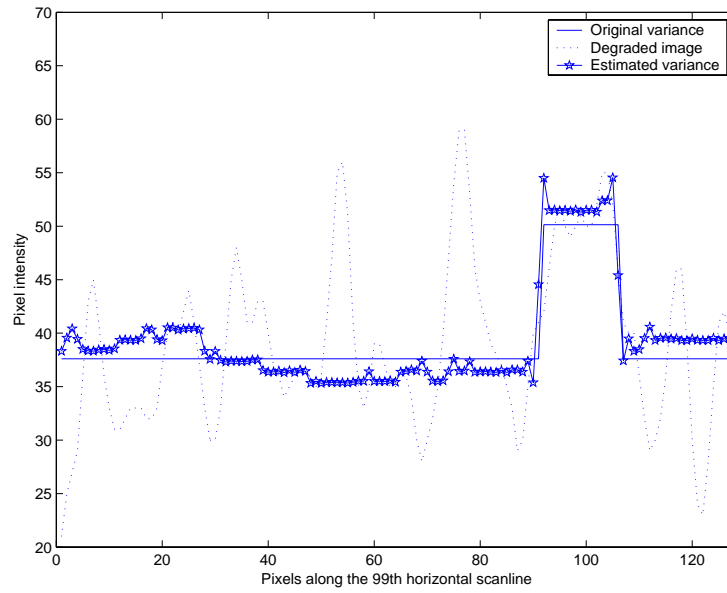


Figure 6.13: Pixels comparison on the 99th rows in the original variance field, blurred image, and estimated variance field, respectively.

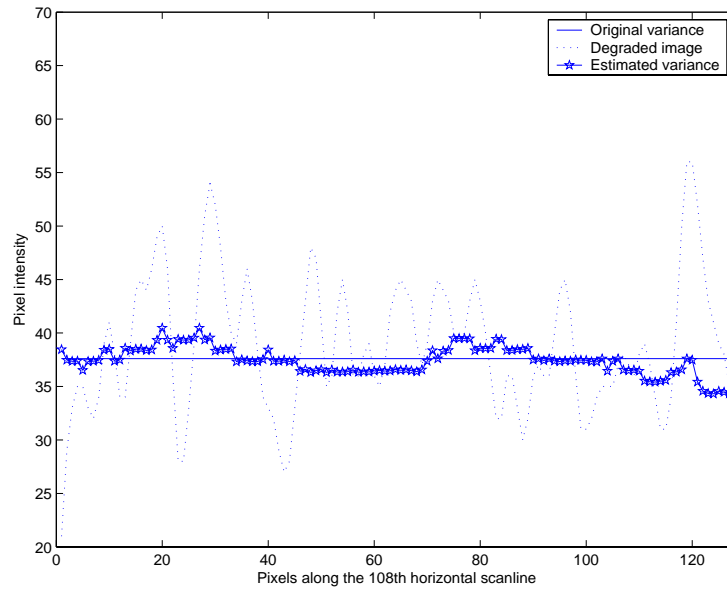


Figure 6.14: Pixels comparison on the 108th rows in the original variance field, blurred image, and estimated variance field, respectively.

6.2 Experiments on Real Image Data

6.2.1 Experiment Design

In addition to testing the speed and resolution performance of the GPU model using synthetic images, it is necessary to experiment on real ultrasound images acquired by commercial ultrasound imaging machines. Instead of experimenting on the images scanned from human tissues, we construct phantoms, which are capable of simulating tissues, and use them to evaluate the model. Compared to the unpredictable complexity of real tissues, phantoms provide more control for designing specific structures that we are interested in, and provide ground truth for comparison.

Because the speed performance of the program is data-independent, the running time of real images is the same as that presented for synthetic images. Therefore, real images are used mainly to evaluate the model's ability to extract true, small-scale structure information from the distorted, observed images. In addition, we are also interested in the influence of the prior on recovering boundaries of different shapes.

First, a cylinder is used to represent our target small structure. In reality, it may simulate a tumor. The object is embedded in a homogeneous medium. It is placed such that its central axis is perpendicular to the plane in which ultrasound pulses and echoes travel. As a result, the cylinder appears to be a circular region in the ultrasound image, provided that it has different physical properties from the surrounding medium. A phantom with a cube inside is also made to experiment boundary effects. Figures 6.15(a) and 6.15(b) illustrate these design ideas.

The cylinder and the cube have the same properties while the surrounding media in the two phantoms are made from the same materials as well. In the real ultrasound images scanned from these phantoms (shown later), the contrast between the small inclusion and the substrate is not as large as in the design pictures.

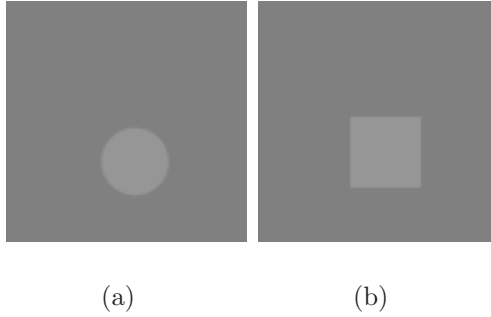


Figure 6.15: Design illustration of the two phantoms. These are the side views. The inclusions, a cylinder in (a) and a cube in (b), are of the same physical properties.

6.2.2 Phantom Construction

In this section, we introduce how tissue-mimic phantoms are actually produced. Adequate ultrasound phantoms should exhibit similar anatomic, physiological, and physical properties to the human or biological tissues that they are designed for. Stiffness, compliance, speed of sound, and acoustic impedance are all important parameters that must be cautiously considered.

First, we need to determine what tissue-equivalent materials to use. This is studied extensively. Possible solutions include a gelatin and graphite powder mixture [73], and an agar and cellulose mixture [3]. We follow the recipe described in [3] which has been proven to be capable of providing desirable properties and is also stable. We modify the densities of components to achieve our purpose.

The material mixture they propose is composed of agar, glycerol, cellulose, and distilled water. Agar is used to form the gel when mixed with water. The amount of glycerol added controls the phantom acoustic velocity. Cellulose particles provide the scattering sites. The size and density of scatterers affect the attenuation coefficient and also the acoustic velocity. We aim to test the model's effectiveness by identifying small-scale tissue structures, and estimating the variance field, which reduces the diffuse scattering speckles. Therefore, tissue types with different attenuation coefficients are needed to simulate the small inclusions and the surrounding

media. This is achieved by altering the density of cellulose.

The substrate medium solution through all the experiments is fixed. Its recipe is given in Table 6.4. The small structures are created with different cellulose densities while keeping the same amounts of other components.

Component	Density	Scale	Model
Agar	3%	by mass	A-6924 Sigma Chemical
Glycerol	3%	by volume	BDH Inc. Toronto, Ontario
Cellulose	2%	by mass	50- μ m S-5504 Sigmacell, Sigma Chemical
Distilled water	the rest	by volume	

Table 6.4: Recipe for the phantom medium, from [3].

In the process of making phantoms, the desired amount of components are well mixed and heated to approximately 78 degrees C, the melting point of agar. It is then cooled to 60 degrees C for small cylindrical inclusions, which are made first. After the cylinder solidifies, it is taken out of the mold and positioned properly into a cubic mould. Then the mixture for the surrounding medium is heated and cooled to 60 degrees C. Finally, it is poured into the same cubic mould with the small cylinder, gradually congealing to form the substrate.

6.2.3 Results on Cylinder Phantom

1. Recorded Images

Figure 6.16(a) shows an ultrasound image displayed on the ultrasound machine. It is scanned from the phantom with a cylinder inside, which has different acoustic properties from the substrate. The real world appearance of the phantom is given in Figure 6.16(b), which shows a picture captured by a digital camera.

The displayed image in Figure 6.16(a) does not present the raw echo signals acquired by the ultrasound system but the interpolated version of the signals. The interpolation aims to reduce the visual distortion caused by inefficient

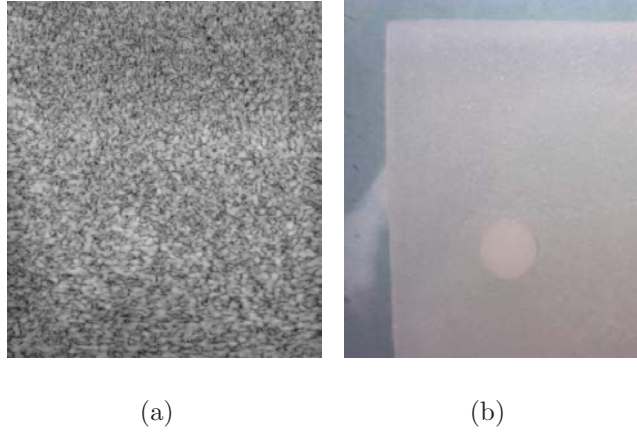


Figure 6.16: (a) Image of phantom A shown on the screen. This is the image interpolated from the raw signals shown in Figure 6.17. (b) Phantom picture captured by a digital camera.

samplings. The diameter of the cylinder, which mimics a small structure, is 6mm. It is 4% of cellulose while the substrate medium is only 2%. The property difference is purposely made small so that the contrast between the inclusion and the substrate is low. From the image, we can observe that the inclusion is almost unnoticeable. This design facilitates evaluating of the model’s capability to reveal inconspicuous tissue structures, a more meaningful and challenging design feature.

The model works on raw image data, since the interpolation process destroys the original system-tissue interaction information that is carried by the raw signals, making restoration more difficult. An ultrasound image formed from raw data is shown in Figure 6.17(a). In Figure 6.17(b), our known cylinder object, whose transect appears to be a circle, is marked by a black oval. The shape and position of the oval are determined by a feature identification procedure, introduced in the next section.

2. Feature Identification

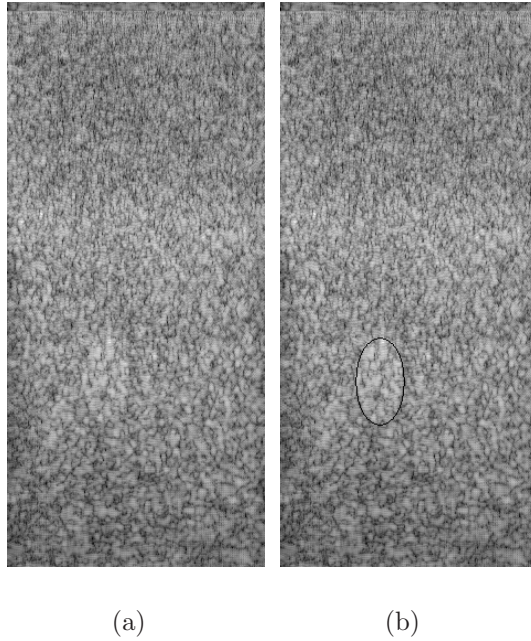


Figure 6.17: Ultrasound raw-data images scanned from the cylinder phantom. They are of resolution 234×512 . (a) Original image. (b) Image in which the known inclusion is marked by a black oval.

The location and shape of the known object in the raw-data image are found by calculating the feature correspondence from one ultrasound image in Figure 6.18(a) and one picture captured by a digital camera in Figure 6.18(b).

The sample ultrasound image is recorded on the same cylinder phantom while four fish wires are placed parallel to the cylinder object. They appear to be four bright dots in the observed ultrasound image, because the fish wire material has a very high attenuation coefficient. We use them as markers to map an image captured by a digital camera to the ultrasound raw-data image.

The shape of the circle showing a cross section of the cylinder is blurred in the ultrasound image, but can be better observed in the camera-captured picture, shown in Figure 6.18(b). A circle can be easily marked manually on the picture, see Figure 6.18(c). It is then transformed to the ultrasound raw

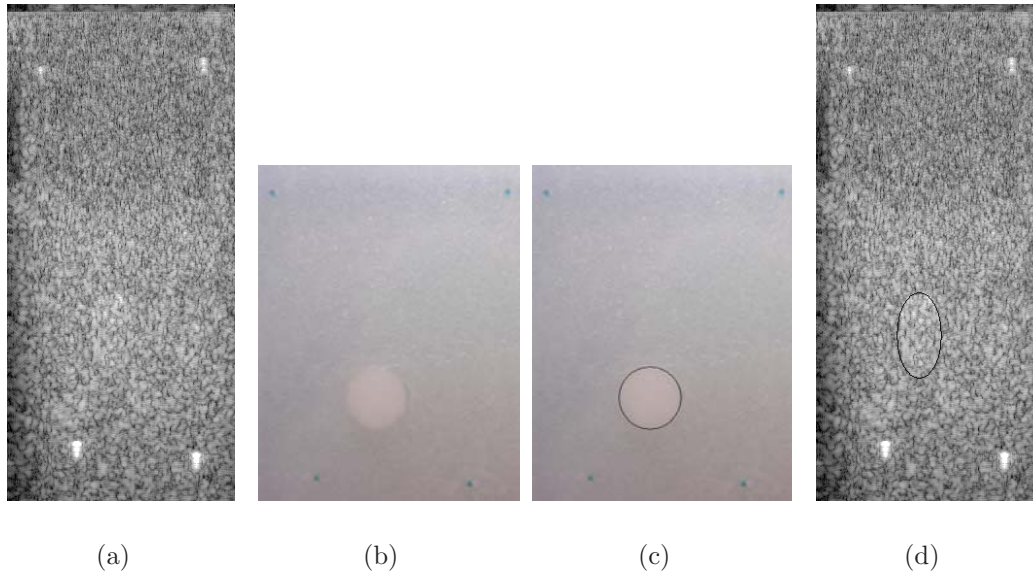


Figure 6.18: (a) Raw-data ultrasound image scanned from the cylinder phantom with four fish wires. It is of resolution 234×512 . (b) Phantom picture captured by a digital camera. (c) Camera-captured picture with the object marked by a cycle. (d) The same raw-data ultrasound image as in (a) with the marked object.

data image by rescaling the digital picture, according to the correspondence of the markers in both images. Any three of the four markers can be picked up for calculation along the horizontal and vertical directions. This is how the real feature region is identified in the same ultrasound image. The object is indicated by the the black oval in Figure 6.18(d). The shape of the object changes from a circle to an oval due to the sampling scale distortion.

The position and shape information in the sample ultrasound image can be used in any other recorded image. We assume that there is no severe pressure during the acquisition process, which might cause tissue distortion. Under this assumption, the shape of the true object is unchanged. Also, the distance between the phantom-top surface and the cylinder is assumed to be fixed. The lateral position of the object can be manually determined by the edge information found in the recorded images.

With these conditions met, the object can be marked on any image scanned from the same phantom, which is helpful for comparing the shape and position of the recovered region with the true object. This is only for qualitative visual evaluation of the results. Errors arise from several aspects:

- The marks in the camera-captured pictures are of finite size.
- The marks in the ultrasound images have finite size and low resolution due to degradation. This is clear because although the four fish wires are the same in the real world, their appearances are different in the ultrasound image.
- Lateral position identification may introduce inaccuracy because of the incomplete information coming from blurred edges.

3. Interested Region

Most of the time, we are interested in small regions in the image which might have subtle objects. It is time consuming to process the whole image. A region which contains the feature is extracted from the raw image. It has 128×128 pixels and is shown in Figure 6.19(b). Without sacrificing generality, this is the target image that we use to estimate the PSF, reflectance and variance.

4. Estimated Images

Figures 6.20 and 6.21 present the estimated PSF and reflectance field, respectively. The estimated point spread function shows a side lobe effect in the lateral direction. The deconvolved reflectance field does not show improved resolution from visual effects. The possible reasons are summarized as follows:

- (a) The deconvolution solver on the reflectance r lacks a regularization term determined by the conditional density function and derived from the joint density.

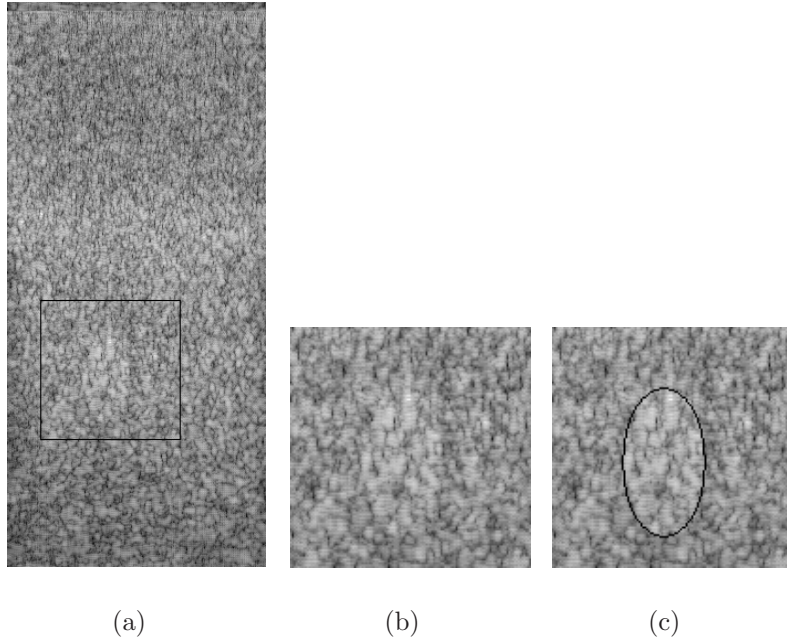


Figure 6.19: (a) A window is specified on the original raw-data image which embraces a cylindrical object. In order to fit the page, the image is resized. (b) A 128×128 small region which corresponds to the window in (a) is extracted from Figure 6.17. (c) This is the same image as in (b) but the object is marked by an oval.

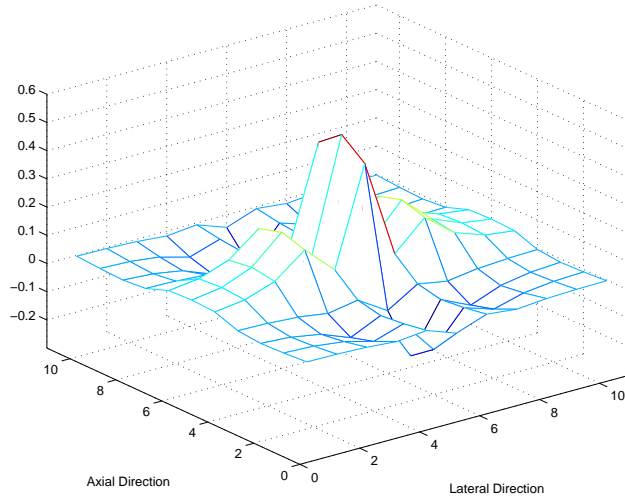


Figure 6.20: Point spread function estimated from Figure 6.19(b).

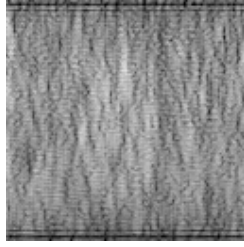


Figure 6.21: Reflectance field estimated from Figure 6.19(b).

- (b) The deconvolution solver for r has no constraints on the possibly generated pixels with negative intensity values. Those pixels deviate the desired solution, which defines the pixel intensity to be nonnegative, and are even badly accumulated through iterations. Constrained image restoration is discussed in Section 8.2.
- (c) Due to the existence of severe noise and also the spatial varying property of the PSF, it is hard to estimate the true point spread function, which influences the estimation results.

However, the variance field estimate is relatively successful. Figure 6.22 shows the estimated variance fields after different numbers of iterations. This can be stopped after arbitrary passes and still produce reasonably good solutions.

The inclusion object in the recovered variance image is more observable than in the observed ultrasound image. The variance field is smooth at originally homogeneous regions and the recovered boundary is close to the true one. The conclusion can be made that for real, acquired, ultrasound images, the deconvolution of the reflectance field is sensitive to the accuracy of the estimated point spread function and may not achieve the goal of improving image resolution. However, the estimated variance field as defined on the tissue's physical property is capable of better exhibiting underlying tissue structures, and hence, helps reduce speckles and reveal concealed true information.

Some small speckles are not reduced completely until 6000 iterations. We note

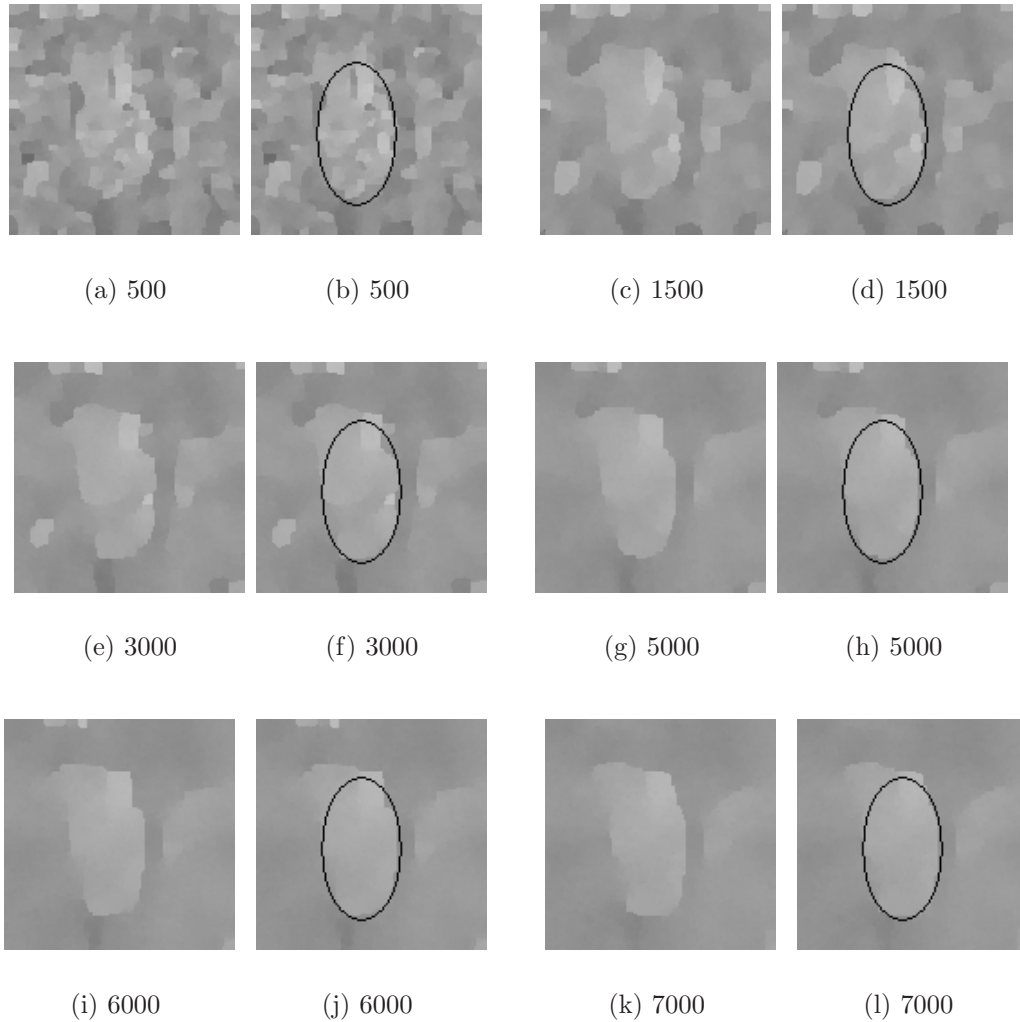


Figure 6.22: Estimated variance fields from Figure 6.19(b) after (a) 500 iterations; (c) 1500 iterations; (e) 3000 iterations; (g) 5000 iterations; (i) 5000 iterations; (k) 6000 iterations and (m) 7000 iterations. The known location and shape of the target feature are marked with a black oval in the corresponding Figures (b), (d), (f), (h), (j), (l), and (n).

that greater unpredictability and randomness in real images than synthetic images results in more iterations being needed to achieve the best outputs. There also exists a tradeoff, however, between the number of iterations and the quality of recovered boundaries. We observe from the estimation results

that the shape of the recovered object tends to become rectangular as more iterations are performed. In other words, there is a trend for the boundaries to change into straight lines. This is determined by the inherent properties of the edge-preserving prior.

The β and γ parameters are set to be 8 and 7, respectively, in this experiment.

6.2.4 Results of the Cube Phantom

An interpolated ultrasound image scanned from a phantom which has a cubic object inside is shown in Figure 6.23(a). Both the width and the length of the cubic inclusion transect in the real world are 6mm. Without prior knowledge, it is hard to tell that there is a rectangular object in the image showing a transect of the phantom. The contrast is low and the edges are blurred. Figure 6.23(b) gives the phantom picture captured by a digital camera. As described in the phantom design section, the inclusion is 4% cellulose while the surrounding medium is 2%. The real dimension of the cube is 6.5cm(height) \times 6cm(width).

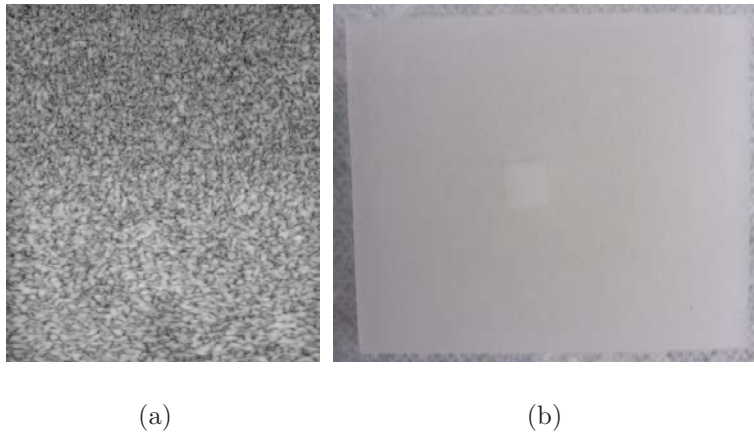


Figure 6.23: (a) An interpolated ultrasound image scanned from the cube phantom. (b) Phantom picture acquired from a digital camera.

The raw-data ultrasound image is presented in Figure 6.24(a). In Figure

6.24(b) the inclusion is marked by a box using the feature identification method which has been introduced. A region containing the target inclusion is extracted from the raw-data image and shown in Figure 6.25.

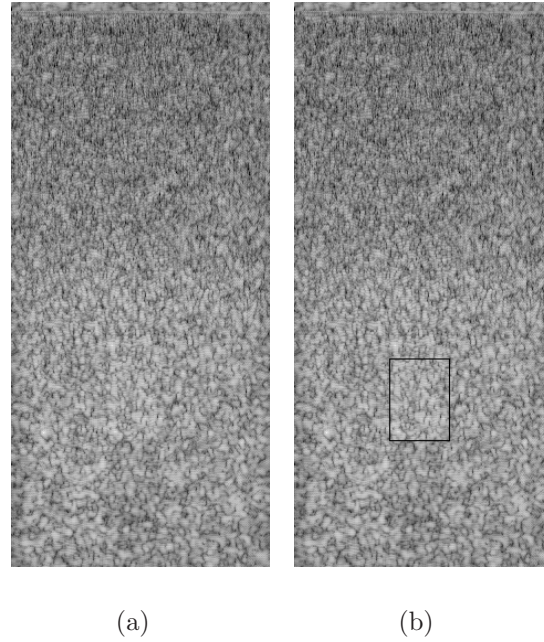


Figure 6.24: (a) Raw-data ultrasound image scanned from the phantom that has a cubic object inside. (b) The same image with object marked by a box.

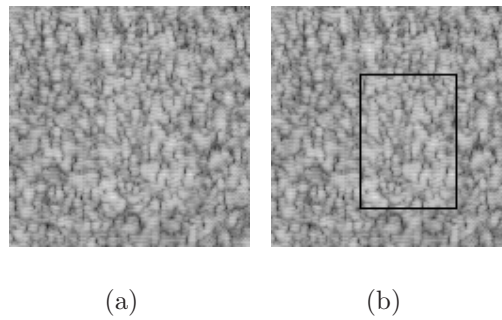


Figure 6.25: (a) An object region in the raw-data image has 128×128 pixels. (b) The same image with object marked by a box.

The estimated variance fields after different numbers of iterations are shown in Figure 6.26. Parameter β is configured as 7 and γ is 8. Some continuous edges and distinct corners are well recovered. The interior region shows pixels with low intensities, which corresponds to the relatively dark center part of the observed image. We conclude that the edge-preserving prior used in the model favors piecewise regions with line-style edges.

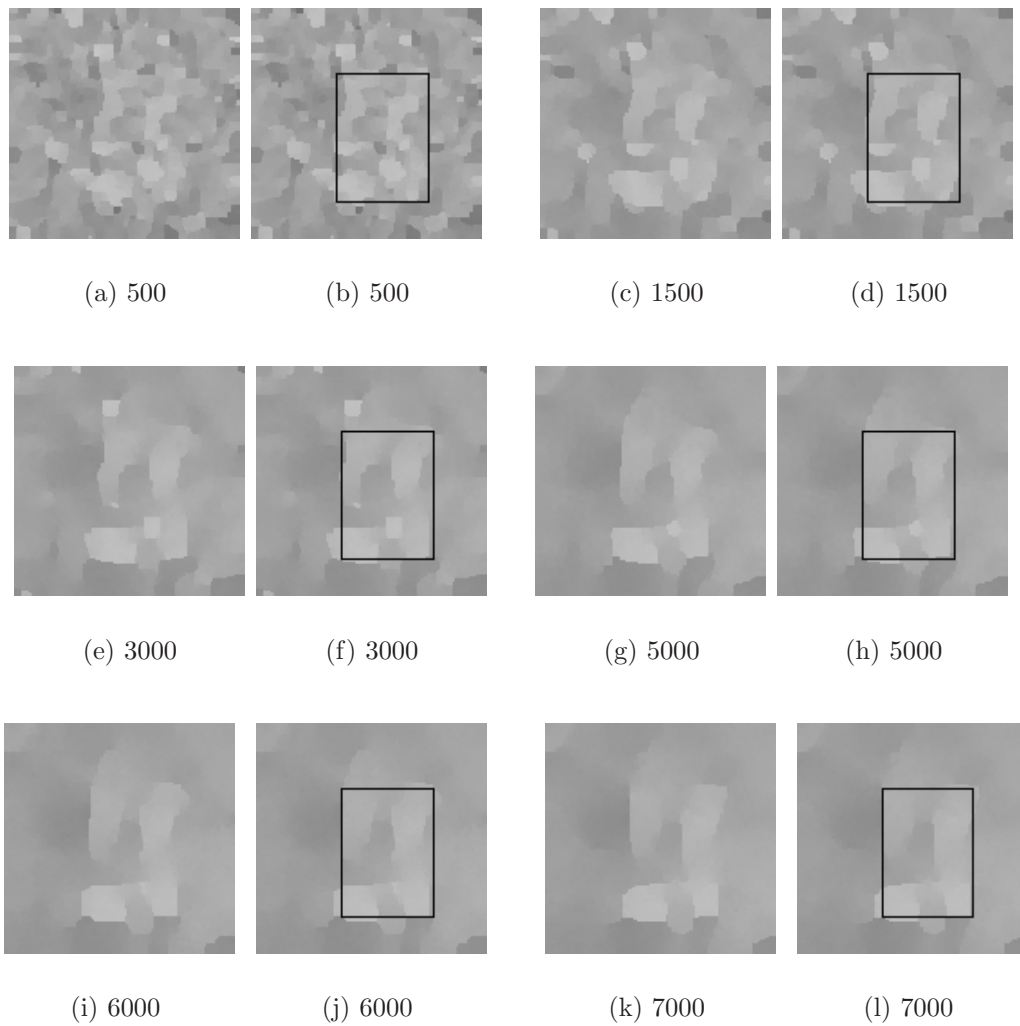


Figure 6.26: Estimated variance field from Figure 6.25 after (a) 500 iterations; (c) 1500 iterations; (e) 3000 iterations; (g) 5000 iterations; (i) 6000 iterations and (k) 7000 iterations. The known location and shape of the target feature is marked with a black box in the corresponding (b), (d), (f), (h), (j), (l), and (n).

Chapter 7

Conclusions

A modified Bayesian ultrasound image analysis model consisting of three steps has been presented here. Its development is based on the model proposed in [6] with improvement on its efficiency and parameters. In contrast with ordinary image restoration, in which reflectance reveals true scene features, the image signal at each pixel in an ultrasound image results from random interference from random cells. Grounded on this fact, the restoration model relies on a statistical diffuse speckle model. The reflectance distribution is defined by a variance parameter. The variance is capable of representing the underlying structures because of its physical property. We want to restore the variance field to better illuminate the true information concealed by the blurring and noise.

The integrated method is implemented entirely on programmable, floating point graphics processing units (GPUs). This is motivated by the fact that the streaming architecture of modern GPUs provides many features better suited for performing image processing tasks. Compared to the original model and CPU-based implementation, here, speed performance improves dramatically from 1.09 second per iteration to 11.49 millisecond per iteration. The image data can be processed at interactive rates.

From our experiments, we conclude that with cautious model design and im-

plementation, the GPU is a promising fast computation coprocessor for performing complicated image related assignments in an ultrasound imaging system, freeing the CPU for other jobs. However, the functionalities on contemporary graphics hardware add limitations on its maximum performance. More operations are expected to be developed and certain deficiencies are desired to be solved. More image processing applications on GPUs will become possible with those graphics hardware improvements in the future.

Chapter 8

Future Work

Certain limitations exist in the model and the implementation presented in this thesis. The following research directions would be fruitful for investigation in the future.

8.1 Extension to General Image Restoration

Iterative methods, such as the Conjugate Gradient (CG) method, are commonly used to restore degraded images. From our experiments on estimating the reflectance field, we conclude that GPUs are suitable for running streaming image processing models. Although we start from ultrasound images, it would be interesting to investigate other restoration models on images from other sources.

8.2 Nonnegative Constraints

The variance field doesn't suffer from the negativity problem, as long as it is initialized with positive values. The reason is that the proposal function is a positive scalar function that only generates positive candidate values on positive fields.

However, the deterministic optimization solver for the reflectance is an unconstrained one. No valid variable range is enforced when searching for optimal values.

As a result, some pixels might be assigned negative intensities after computation. They are beyond the common pixel intensity range, which is from 0 to 255, and they need to be set to zero in a final visualization. Restoration with non-negativity constraints can reduce these unreasonable results. This has been investigated in [74] and [75]. Constrained image restoration is much more complicated and it may incur losing detailed features [76]. However, it is still worth investigation since this is an inherent problem with deterministic unconstrained image restoration methods.

8.3 Parameter Configuration

The model is a parametric model, which suffers from difficulty in determining good parameters. Currently, the parameters are pre-computed by trial-and-error and considered fixed during updating. In future, we would like to investigate parameter estimation techniques and make the model more automatic.

8.4 Deconvolution of Astronomical Images at Iterative Rates

Astronomical images are always blurred versions of true scenes. Degradation sources include instrument effects such as optical lense defects and environmental effects such as atmospheric turbulence due to the long light travel distance and other stable influences. Restoration of the observed astronomical images at interactive rates would assist in many related applications.

Deconvolution is a very important approach to improving astronomical image resolution. We would like to extend our idea of using graphics hardware to accelerate ultrasound image deconvolution to these astronomical images with selectively designed deconvolution models.

Bibliography

- [1] N. Devillard. Fast median search: an ANSI C implementation. July 1998.
- [2] K. Moreland and E. Angel. The FFT on a GPU. In *Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware 03*, July 2003.
- [3] D. W. Rickey, P. A. Picot, D. A. Christopher, and A. Fenster. A wall-less vessel phantom for doppler ultrasound studies. In *Ultrasound in Med. & Biol.*, volume 21, pages 1163–1176, 1995.
- [4] L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New-York, 1986.
- [5] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer, 1999.
- [6] O. Husby, T. Lie, T. Langø, J. H. Hokland, and H. Rue. Bayesian 2-D deconvolution: A model for diffuse ultrasound scattering. Statistics 20, Norwegian University of Science and Technology (NTNU), 1999.
- [7] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.
- [8] E. Lindholm, M. J. Kilgard, and H. Moreton. A user-programmable vertex engine. In *Proceedings of ACM SIGGRAPH 01*, 2001.
- [9] T. J. Purcell, T. Buck, W. R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. In *Proceedings of ACM SIGGRAPH 02*, 2002.
- [10] U. R. Abeyratne, A. P. Petropulu, and J. M. Rei. Higher order spectra based deconvolution of ultrasound images. In *IEEE Transactions on Ultrasonics, ferroelectrics and frequency control*, volume 42, pages 1064–1075, November 1995.

- [11] J. Jensen, J. Mathorne, T. Gravesen, and B. Stage. Deconvolution of in-vivo ultrasound B-mode images. In *Ultrasonic Imaging*, volume 13, pages 122–133, 1993.
- [12] A. K. Katsaggelos. Iterative image restoration algorithms. In *Optical Engineering*, volume 28, pages 735–748, 1989.
- [13] R. N. Czerwinski, D. L. Jones, and Jr. O’Brien W.D. Ultrasound speckle reduction by directional median filtering. In *1995 International Conference on Image Processing*, volume 1, 1995.
- [14] T. Taxt and G. V. Frolova. Noise robust one-dimensional blind deconvolution of medical ultrasound images. In *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, volume 46, March 1999.
- [15] T. Taxt. Restoration of medical ultrasound images using two-dimensional homomorphic deconvolution. In *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, volume 42, pages 543–554, July 1995.
- [16] T. taxt. Three-dimensional blind deconvolution of ultrasound images. In *IEEE transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, volume 48, pages 867–871, July 2001.
- [17] M. E. Anderson and G. E. Trahey. A seminar on k-space applied to medical ultrasound. Course Notes for use in BME 265, Duke University, April 2000.
- [18] M. Berson, A. Roncin, and L. Pourcelot. Compound scanning with and electrically steered beam. In *Ultrasonic imaging*, volume 3, pages 303–308, 1981.
- [19] D. Shattuck and OT. Ramm. Compound scanning with a phased array. In *Ultrasonic Imaging*, volume 4, pages 93–107, 1982.
- [20] P. M. Shankar. Speckle reduction in ultrasound B-scans using weighted averaging in spatial companding. In *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, volume 33, pages 754–758, November 1986.
- [21] R. Rohling, A. Gee, and L. Berman. 3-D spatial compounding of ultrasound images. In *Medical Image Analysis, Oxford University Press*, volume 1, pages 177–193, Oxford, UK, 1997.
- [22] R. Entrekin, P. Jackson, J. R. Jago, and B. A. Porter. Real time spatial compound imaging in breast ultrasound: technology and early clinical experience. In *MedicaMundi*, volume 43, pages 35–42, 1999.

- [23] E. P. Simoncelli. *Bayesian Inference in Wavelet Eased Models*, volume 141, chapter Bayesian Denoising of Visual Images in the Wavelet Domain, pages 291–308. Springer-Verlag, New York, 1999. Lecture Notes in Statistics.
- [24] Y. Wan and R. Nowak. A multiscale Bayesian framework for linear inverse problems and its application to image restoration. *IEEE Transactions on Image Processing*, January 2001.
- [25] X. Zong, A. F. Laine, and E. A. Geiser. Speckle reduction and contrast enhancement of echocardiograms via multiscale nonlinear processing. In *IEEE Transactions on Medical Imaging*, volume 17, August 1998.
- [26] A. Achim, A. Bezerianos, and P. Tsakalides. Novel Bayesian multiscale method for speckle removal in medical ultrasound images. In *IEEE Transactions on Medical Imaging*, volume 20, August 2001.
- [27] J. H. Hokland and P. A. Kelly. Markov models of specular and diffuse scattering in restoration of medical ultrasound images. In *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, volume 43, July 1996.
- [28] Nvidia Corporation. *Cg Toolkit User's Manual*, release 1.1 edition, March 2003.
- [29] M. Hopf and T. Ertl. Hardware accelerated wavelet transformations. In *Proceedings of EG/IEEE TCVG Symposium on Visualization (VisSym'00)*, May 2000.
- [30] M. Hopf and T. Ertl. Accelerating 3D convolution using graphics hardware. In *Proceedings of IEEE Visualization 99*, pages 471–474, 1999.
- [31] M. Hopf and T. Ertl. Accelerating morphological analysis with graphics hardware. In *Workshop on Vision, Modelling, and Visualization VMV '00*, pages 337–345, 2000.
- [32] M. Rumpf and R. Strzodka. Nonlinear diffusion in graphics hardware. In *Proceedings of Eurographics/IEEE TCVG Symposium on Visualization*, pages 75–84, May 2001.
- [33] P. Colantoni, N. Boukala, and J. Da Rugna. Fast and accurate color image processing using 3D graphics cards. In *The 8th International Fall Workshop on Vision, Modeling and Visualization*, 2003.
- [34] A. E. Lefohn, J. E. Cates, and R. T. Whitaker. Iterative, GPU-based level sets for 3D segmentation. In *The Sixth Annual International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2003.

- [35] A. Sherbondy, M. Houston, and S. Napel. Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In *Proceedings of IEEE Visualization 03*, 2003.
- [36] E. S. Larsen and D. K. McAllister. Fast matrix multiplies using graphics hardware. In *Proceedings of IEEE Supercomputing*, November 2001.
- [37] J. D. Hall, N. A. Carr, and J. C. Hart. Cache and bandwidth aware matrix multiplication on the GPU. Technical report, University of Illinois, 2003.
- [38] C. J. Thompson, S. Hahn, and m. Oskin. Using moden graphics architectures for general-purpose computing: A framework and analysis. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture*, pages 306–317, November 2002.
- [39] Á. Moravánszky. *ShaderX 2: Shader Programming Tips and Tricks with DirectX 9*, chapter Dense Matrix Algebra on the GPU. Wordware Publishing, 2003.
- [40] M. J. Harris, G. Coombe, T. Scheuermann, and A. Lastra. Physically-based visual simulation on graphics hardware. In *Proceeding of ACM SIGGRAPH 02*, 2002.
- [41] M. J. Harris, W. V. Baxter III, T. Scheuermann, and A. Lastra. Simulation of cloud dynamics on graphics hardware. In *Proceedings of Graphics Hardware 02*, 2002.
- [42] D. Knott and D. K. Pai. CInDeR: Collision and interference detection in real-time using graphics hardware. In *Proceedings of Graphics Interface 03*, pages 73–80, 2003.
- [43] S. Redon N. Govindaraju, M. C. Lin, and D. Manocha. Interactive collision detection between complex models using graphics hardware. In *Proceedings of Graphics Hardware 03*, 2003.
- [44] J. Kruger and R. Westermann. Linear algebra operators for GPU implementation of numerical algorithms. In *Proceedings of ACM SIGGRAPH 03*, 2003.
- [45] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. In *Proceedings of ACM SIGGRAPH 03*, 2003.

- [46] R. F. Wagner, M. F. Insana, and D. G. Brown. Statistical properties of radio-frequency and envelope-detected signals with application to medical ultrasound. In *J. Opt. Soc. Amer.*, volume 4, 1987.
- [47] V. Dutt. *Statistical analysis of ultrasound echo envelope*. PhD thesis, Mayo Clinic College of Medicine, August 1995.
- [48] J. W. Goodman. *Laser Speckle and Related Phenomena*, chapter Statistical Properties of laser speckle patterns. New York:Springer-Verlag, 1975.
- [49] R. F. Wagner, M. F. Insana, and D. G. Brown. Unified approach to the detection and classification of speckle texture in diagnostic ultrasound. In *Optical Engineering*, volume 25, 1986.
- [50] M. F. Insana, R. F. Wagner, B. S. Garra, D. G. Brown, and T. H. Shawker. Analysis of ultrasound image texture via generalized Rician statistics. In *Optical Engineering*, volume 25, 1986.
- [51] T. Langø, T. Lie, O. Husby, and J. Hokland. Effect of spatially invariant ultrasound point spread function. In *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, volume 48, pages 131–141, 2001.
- [52] <http://computer.howstuffworks.com/computer-memory2.htm>.
- [53] M. J. T. Smith and A. Docef. *A study Guide for Digital Image Processing*. Scientific Publishers, 1999.
- [54] J. T. Kent K. M. A. de Souza and K. V. Mardia. Estimation of objects in highly variable images using Markov Chain Monte Carlo. In *Proceedings of the 8th British Machine Vision Conferences*, 1997.
- [55] S. Gemen and D. McClure. Statistical methods for tomographic image reconstruction. In *Proc. 46th Sess. Int. Stat. Inst. Bulletin ISI*, volume 52, 1987.
- [56] M. A. Hurn, O. Husby, and H. Rue. *Spatial Statistics and Computational Methods, Lecture Notes in Statistics 173*, chapter A Tutorial on Image Analysis, pages 87–141. Berlin: Springer Verlag, 2003.
- [57] A. Mohammad-Djafari. Joint estimation of parameters and hyperparameters in a Bayesian approach of solving inverse problems. In *Proceedings of the International Conference on Image Processing*, volume II, pages 473–477, 1996.
- [58] L. R. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals*. Prentice Hall, Englewood Cliffs, NJ, 1978.

- [59] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes, The art of Scientific Computing*. Cambridge University Press, 1986.
- [60] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. In *IEEE Transactions on Pattern Analysis and machine intelligence*, volume PAMI-6, pages 721–741, 1984.
- [61] D. Geman and G. Reynolds. Constrained restoration and the recovery of discontinuities. In *IEEE Transactions on Pattern Analysis and machine intelligence*, volume 14, pages 367–383, 1992.
- [62] D. F. Yu and J. A. Fessler. Edge-preserving tomographic reconstruction with nonlocal regularization. In *IEEE transactions on medical imaging*, volume 21, pages 159–173, 2002.
- [63] V. E. Johnson, W. H. Wong, X. Hu, and C. T. Chen. Image restoration using Gibbs priors: boundary modelling, treatment of blurring, and selection of hyperparameter. In *IEEE transactions on pattern analysis and machine intelligence*, volume 13, pages 413–425, May 1991.
- [64] Z. Zhou, R. M. Leahy, and J. Qi. Approximate maximum likelihood hyperparameter estimation for Gibbs priors. In *IEEE transactions on image processing*, volume 6, June 1997.
- [65] A. Mohammad-Djafari. On the estimation of hyperparameters in Bayesian approach of solving inverse problems. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 495–498, 1993.
- [66] C. Wynn. Using p-buffers for off-screen rendering in OpenGL. NVIDIA Corporation, 2001.
- [67] <http://whatis.techtarget.com>.
- [68] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys. A multi-grid solver for boundary value problems using programmable graphics hardware. In *Proceedings of Graphics Hardware 03*, pages 1–11, 2003.
- [69] K. Perlin. An image synthesizer. In *IEEE Computer Graphics: Image Synthesis*, volume 19, 1988.
- [70] M. Whiley and S. Wilson. Parallel methods for MCMC with spatial Gaussian models. Presentation at Workshop on Parallel Algorithms in Statistical Methods, January 2002.

- [71] D. Wilkinson. Parallel block-MCMC for large highly structured latent process models, using MPI on a distributed memory cluster. Presentation at Workshop on Parallel Algorithms in Statistical Methods, January 2002.
- [72] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *1998 ICASSP Conference Proceedings*, volume 3, pages 1381–1384, 1998.
- [73] M. M. Burlew, E. L. Madsen, J. A. Zagzebski, and S. W. Sum. A new ultrasound tissue-equivalent material. In *Radiology*, volume 134, pages 517–520, 1980.
- [74] J. G. Nagy and Z. Strakos. Enforcing nonnegativity in image reconstruction algorithms. In *Mathematical Modeling, Estimation, and Imaging*, volume 4121, pages 182–190, 2000.
- [75] M. Hanke, J. G. Nagy, and C. Vogel. Quasi-newton approach to nonnegative image restorations. In *Linear Alg. and Appl.*, volume 316, pages 223–236, 2000.
- [76] M. Hanke. *Surveys on Solution Methods for Inverse Problems*, chapter Iterative Regularization Techniques in Image Restoration, pages 35–52. Springer-Verlag, 2000.