Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science
Mathematical Image Analysis Group

Master's Thesis

Efficient Computation and Representation of the Diffusion Echo

Submitted by

Özgün Çiçek

on
December 17, 2014

Supervisor:
Prof. Dr. Joachim Weickert

Advisor:
Prof. Dr. Joachim Weickert

Reviewers:
Prof. Dr. Joachim Weickert
Dr. Björn Andres

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, December 17, 2014 Özgün Çiçek

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass die vorliegende Arbeit mit der elektronischen Version übereinstimmt.

Statement in Lieu of an Oath

I hereby confirm the congruence of the contents of the printed data and the electronic version of the thesis.

Saarbrücken, December 17, 2014 Özgün Çiçek

Abstract

Diffusion gained an important role in image processing and computer vision problems. Each mathematically well-founded diffusion model proved its advantages especially in image smoothing. However, unlike the linear diffusion, nonlinear models require a strong mathematical intuition to understand the process. A visual description would yield a better interpretation within and in between various models. *The diffusion echo* is a good way to visualise diffusion processes. However, it is computationally expensive. The diffusion echo of a pixel in an image has the size of the image itself. Additionally, in nonlinear settings, the diffusion echo is also different for each pixel. Moreover, large stopping times are useful to analyse the image structures, which increases the computational burden further.

To this end, this thesis exploits parallel programming on GPU with CUDA and Fast Explicit Diffusion (FED) schemes to compute diffusion echoes in an efficient way. With the implementation in this thesis, it is possible to compute 1024 segment-like diffusion echoes of an image of size 256×256 in 160 sec.

Furthermore, having thousands of segment-like diffusion echoes led us to analyse the image structures in a compact way. PCA is used to find significant *eigenechoes* to represent all diffusion echoes of an image. In our evaluation, the first 30 eigenechoes of 4096 diffusion echoes resulting from edge-enhancing diffusion of a 256×256 image can represent all 65536 diffusion echoes of the image with the average MSE of 10.83.

Inspired by the scale-space representation, we introduce a diffusion echo-driven segment scale-space. Each segment-like echo can be used as a coarse segment and traced back in time to improve localization.

Besides the wide range of possible future work, the work in this thesis can help improving the existing diffusion models as well as designing new ones with more intuitive and better understanding.

Acknowledgements

I would like to thank several people for their great support and understanding during my master's thesis.

First of all, I would like to thank Prof. Joachim Weickert for giving me the opportunity to write my thesis in the Mathematical Image Analysis (MIA) Group under his supervision and offering me many interesting topics from which I could choose the one that fits best to my interests and work with motivation. Without his elegant ideas, fruitful discussions and without being an HiWi student in his group, this work would have not been possible. I also would like to thank him for providing me with the sequential implementations of the diffusion models and the FED scheme used in this thesis.

I would like to thank all MIA members for the inspirational atmosphere and discussions during my thesis and my work as a HiWi student. Especially, I would like to thank Christopher Schroers for his help and suggestions in parallel programming. Without him I would have not been able to learn it in such a short time. I would also like to thank Pascal Peter for the fruitful discussions on possible future work.

I would also like to thank all my friends in Saarbruecken for making these two years so unforgettable for me. I appreciate your understanding during the writing part of the thesis. I also thank my friends in Ankara a lot for the great understanding and support during my master studies. Especially, I thank Bennet for making the life easier for me, Ebru and Ezgi for their friendly support whenever I need, Hüseyin for the great discussions we had, and Sophie, Lucas, and Antoanetta for all the fun.

I cannot thank my family enough for their never ending encouragement and love. Thank you mom for your patience with me.

To the memory of my uncle...

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Motivation. Diffusion processes become more and more involved in image processing and computer vision. They are good examples of the physical concepts which are used fruitfully in this field. They are used in a wide range of image processing and computer vision tasks such as image compression, multi-scale representations and image smoothing. Their power is two fold. First, they are mathematically well-founded schemes offering a transparent continuous modeling of the diffusion phenomena [22]. This allows to design stable models for specific applications with a physical intuition. The linear diffusion model can be used to obtain a Gaussian smoothing. The nonlinear isotropic diffusion model would be more suitable for a discontinuity preserving smoothing. Also anisotropic diffusion models can be chosen in order to steer the diffusion more according to the underlying local image structure. For example, the edge-enhancing diffusion model would be better when the edges are the center of attention. Similarly, the coherence-enhancing diffusion model would be more useful when the coherence is the most desired structure. A very comprehensive analysis on each model can be found in [22]. Secondly, they still favor the development of new models as well as the reinterpretation of the classical models. Anisotropic diffusion can be modified to create a new model called corner enhancing diffusion by introducing a corner-ness measure [6]. It is also possible to see Gaussian smoothing from a different point of view by reformulating it into linear diffusion.

Despite their promising results and popularity, how they exactly behave on a single pixel is still not easily interpretable. This is exactly the motivation for the work by Dam and Nielsen [6]. Their starting point is the existence of a Green's function for the Partial Differential Equation (PDE) of the linear diffusion as a Gaussian [9, pp. 43-56]. The Green's function tells exactly how the convolution filter looks like. This means that the convolution of the image with the corresponding Green's function is equivalent to iterating the process defined by the PDE. Although nonlinear diffusion models yield more desirable results, they lack of such an intuitive and visual description. Because their dependence on the local structure around each pixel turns into a disadvantage: the diffusion filter changes from pixel to pixel and has the size of the original image itself. While the research on a closed form solution for the PDEs

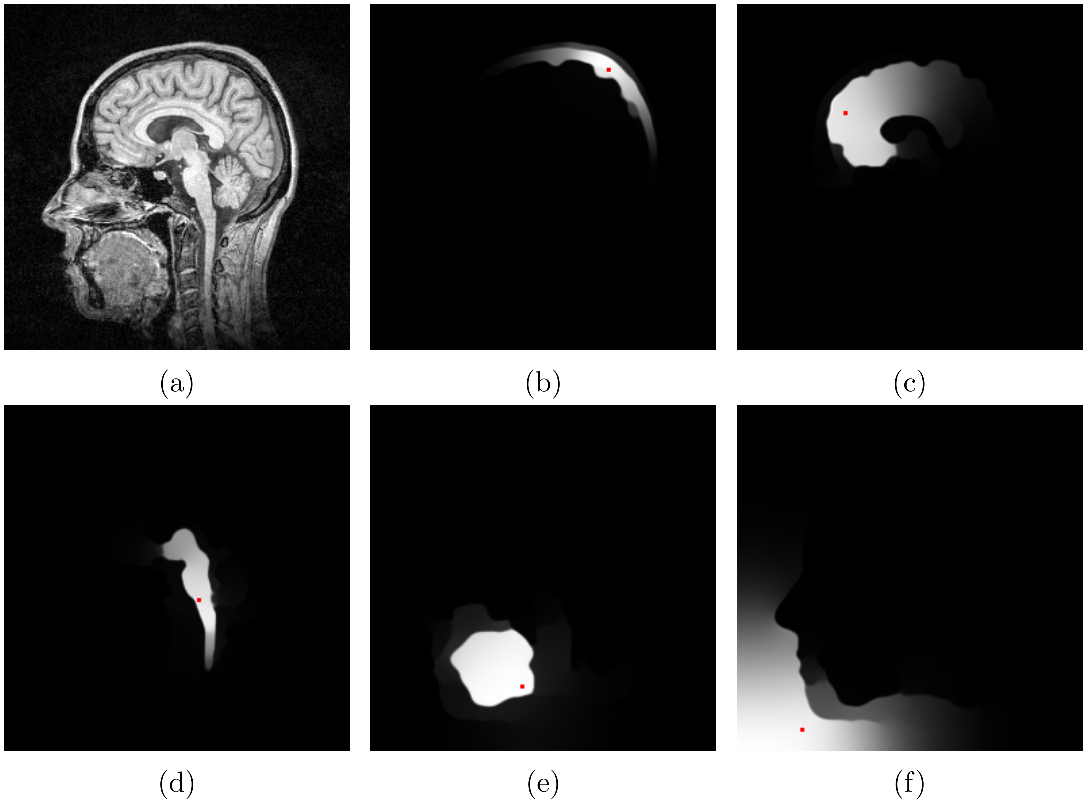


Figure 1.1: Diffusion echoes (b)-(f) of different pixels of 256×256 head image (a) for isotropic nonlinear diffusion with Perona-Malik diffusivity with $\sigma = 0.5$, $\lambda = 1$, number of FED cycles=10, and diffusion time T=2000.

of nonlinear diffusion models is still ongoing, Dam and Nielsen [6] offered an explicit computation of the impulse response of a nonlinear diffusion process which they call *the diffusion echo*. Figure 1.1 shows various diffusion echoes of different pixels resulting from the isotropic nonlinear diffusion with the Perona-Malik diffusivity. Each diffusion echo belongs to the pixel marked with a red dot. It can be observed that each pixel spreads differently depending on the local structure and the diffusion process.

Related work and contribution. According to our knowledge, Dam and Nielsen's work [6] is the only work on the concept of the diffusion echo until quite recently. The diffusion echoes they propose are very preliminary. They are limited to several pixels and early stopping times due to the computational expenses in those times. Therefore, they lack of further analysis on large stopping times and better representation of all diffusion echoes. A very recent and related work on the diffusion echo is Jennewein's master's thesis [11]. It focuses on theoretical properties of the diffusion echo without advanced numeric and representative concerns. This is exactly the scope of this thesis. In the first part, my work addresses the efficient computation of the diffusion echoes of the sampled pixels of an image. In recent years, the General-Purpose Computing on Graphics Processing Unit (GPGPU) offers tremendous speed ups for parallelizable tasks. The idea is to exploit the parallel architecture of the Graphics Processing Unit

(GPU) in order to compute thousands of independent computations simultaneously. The Compute Unified Device Architecture (CUDA) introduced by NVIDIA provides a high-level platform for the GPU programming. Also the Fast Explicit Diffusion (FED) scheme [8] [7] is used to speed up diffusion processes along with the parallel computations. In the second part, my work focuses on the compact representations of the diffusion echoes by the important image structures. A well-known machine learning algorithm, the Principal Component Analysis (PCA), is used to find the representative *eigenechoes*. Therefore, storing only few eigenechoes will be enough to represent and reconstruct all the diffusion echoes. Another very recent work, which is close in spirit, is [21] by Milanfar and Talebi. They compute the filter entries only for the sampled points in an image. The Nystroem extension helps them to approximate the eigenvectors and the eigenvalues of the whole filter by these sampled filters. Then they shrink the eigenvalues to denoise the image. Their work is very close to ours except the method they use to approximate eigenvectors and the filtering methods they focus on. They improve patch-based denoising methods such as Non-Local Means (NLM) and Block Matching and 3-D (BM3D) filtering. In order to compare both methods, we adapted their eigenvector approximation to the diffusion echoes.

Organisation. The rest of the thesis is organized as follows. In Chapter 2, we will establish a necessary background knowledge for the thesis by introducing the diffusion models, their discretizations, the fast explicit diffusion scheme, the scale-space concept and the mean squared error. In Chapter 3, we will introduce the diffusion echo in detail and analyse the diffusion echoes of various diffusion models. In Chapter 4, we will present GPGPU and CUDA briefly. Then we will explain how to compute diffusion echoes efficiently with parallel algorithms. We will provide experiments with different diffusion models and parameters with corresponding computational times. The chapter will conclude with a brief discussion. In Chapter 5, we will introduce the PCA. Afterwards, we will evaluate the potential of the eigenechoes to represent the individual image segments. Following the PCA, we will discuss how to adapt Nystroem approximation to our framework. We will provide with experiments including both methods with different parameters, diffusion models and images. The chapter will end with a brief discussion. In Chapter 6, we will propose a diffusion echo-driven segment scale-space. In Chapter 7, we will conclude the work in the thesis and discuss possible future work.

Chapter 2

Preliminaries

In this chapter, we will explain various diffusion models, their discretizations and how to speed them up using the Fast Explicit Diffusion (FED) scheme already in the sequential programmes. Afterwards, we will explain the scale-space concept and the Mean Squared Error (MSE) which we use as the error measure.

2.1 Diffusion Models

Diffusion process is a physical concept that tries to equilibrate concentration differences in a system by conserving the overall mass in it. The concentration differences create a flux j which tries to equilibrate these differences [22]. *Fick's law* states this equilibrium as

$$j = -D \cdot \nabla u, \tag{2.1}$$

where D is the *diffusion tensor* which is a symmetric positive definite matrix and the concentration gradient is defined as $\nabla u := (\partial_x u, \partial_y u)^T$ in 2-D.

The preservation of mass in a diffusion process can be expressed as

$$\partial_t u = -div(j), \tag{2.2}$$

where t denotes time and the divergence operator is defined for a 2-D vector $\boldsymbol{v} = (v_1, v_2)^T$ as $div(\boldsymbol{v}) := (\partial_x v_1 + \partial_y v_2)$.

Combining these 2 equations, one obtains the *diffusion equation*, also known as *heat equation*, as

$$\partial_t u = div(D \cdot \nabla u). \tag{2.3}$$

When this concentration differences are associated with the gradients of grey values in an image, the diffusion process can be interpreted as image smoothing and detail removing [22].

2.1.1 Linear Diffusion

Linear diffusion is the simplest and the most intuitive diffusion model. The diffusion tensor is replaced by the identity matrix I to impose constant diffusivity. This means the diffusion process is independent of the evolving image structures; therefore, the same for all pixels in the image [22].

For a bounded image $f \in L^\infty(\Omega)$, where Ω represents the rectangular 2-D image grid, the PDE for linear diffusion is expressed as

$$\partial_t u = \Delta u, \tag{2.4}$$

$$u(x,0) = f(x), \tag{2.5}$$

where the Laplacian operator is defined as $\Delta u := (\partial_{xx} u + \partial_{yy} u)$ in 2-D. This PDE has a unique solution [9, pp. 43-56] that is given by

$$u(x,t) = \begin{cases} f(x) & (t = 0) \\ (K_{\sqrt{2t}} * f)(x) & (t > 0) \end{cases}, \tag{2.6}$$

where $*$ is the convolution operator defined as

$$(K_\sigma * f)(x) := \int_{\mathbb{R}^2} K_\sigma(x-y) f(y) \mathrm{d}y. \tag{2.7}$$

Here K_σ is a 2-D Gaussian with standard deviation $\sigma > 0$:

$$K_\sigma(x) := \frac{1}{2\pi\sigma^2} \cdot \exp\left(-\frac{|x|^2}{2\sigma^2}\right). \tag{2.8}$$

We see that reaching a stopping time T in diffusion process is equivalent to convolving the image with a Gaussian of standard deviation σ where the relation is $T = \frac{1}{2}\sigma^2$.

Gaussian as a Green's function helps interpret the evolution of the linear diffusion much better. It provides a visual description of the convolution filter. It is obvious from the Gaussian filter that the new value of a pixel after linear diffusion process is nothing but a Gaussian weighted average of the neighbouring pixels. Another advantage is that the Gaussian scale-space is one of the well-studied scale-spaces in the literature [26] [10].

Although it is simple, intuitive and well-studied, it has several disadvantages. The process is completely independent of the underlying image structure. Therefore, as it removes the noise and blurs the image, it also blurs and delocalizes important structures in the image, such as edges and corners [22]. This is often undesired. However, these problems can be addressed by nonlinear diffusion models.

2.1.2 Isotropic Nonlinear Diffusion

The isotropic nonlinear diffusion is remedy for the disadvantages of the linear diffusion. It adapts the diffusion process according to the evolving image structures to preserve discontinuities [22]. Therefore, in the scale-space representation edges are well-localized and survive long. The PDE of this model proposed by Perona-Malik [16] is

$$\partial_t u = div(g(|\nabla u_\sigma|^2)\nabla u). \tag{2.9}$$

σ is not in the original work of Perona and Malik [16] and added later by Catté et al. [5]. Here $u_\sigma := K_\sigma * u$ smoothes the image with a Gaussian prior to differentiation, thus allows to establish well-posedness properties for this process.

The basic idea here is that with the help of diffusivity function g, the diffusion process is slowed down at possible edges to favor diffusion within the segments other than at segment boundaries. As a result, edges are preserved. In order to reach this goal, a proper diffusivity function $g(s^2)$ must follow these properties [23]:

- $g > 0$ and $g \in C^\infty$
- $g(0) = 1$, decreasing in s^2 on $[0, \infty)$ and $\lim_{s^2 \to \infty} g(s^2) = 0$.

There are several diffusivity functions available in the literature. However we will only present three of them here which are mentioned often in the thesis.

- Diffusivity function proposed by Perona and Malik [16]:

  $$g(s^2) = \frac{1}{1 + s^2/\lambda^2} \quad (\lambda > 0). \tag{2.10}$$

  λ controls the importance of an edge and gradient magnitude s^2 serves as a fuzzy edge detector. The edges where the gradient magnitude is less than λ are smoothed by forward diffusion while the edges having gradient magnitude greater than λ are even enhanced by backward diffusion.

- Weickert's diffusivity [22]:

  $$g(s^2) = \begin{cases} 1 & (s^2 = 0) \\ 1 - \exp \frac{-3.31488}{(s/\lambda)^8} & (s^2 > 0) \end{cases}. \tag{2.11}$$

  It decays more rapidly comparing to (2.10). Therefore, discontinuities survive for a longer time and more segment-like results can be obtained.

- Learned diffusivity family:

  $$g(s^2) = \left(1 + \frac{s^2}{\lambda^2}\right)^{-\gamma} \quad (\lambda > 0). \tag{2.12}$$

It is proposed in a very recent research on natural image statistics [17]. The diffusivity and its parameters are learned from the natural images.

2.1.3 Anisotropic Nonlinear Diffusion

The isotropic nonlinear diffusion still has a scalar diffusivity that depends on only the magnitude of the edges. This can be extended anisotropically to the direction of the local structure by exchanging the scalar diffusivity function by a diffusion tensor [22]. The diffusion tensor is a symmetric, positive definite and 2×2 matrix. It can be formulated as

$$D(\nabla u_\sigma) := (v_1|v_2) diag(\lambda_1, \lambda_2) \begin{pmatrix} v_1^T \\ v_2^T \end{pmatrix} = \lambda_1 v_1 v_1^T + \lambda_2 v_2 v_2^T. \tag{2.13}$$

Here v_1, v_2 are the eigenvectors and λ_1, λ_2 are the corresponding eigenvalues. The diffusion tensor needs to reveal local image structure. Therefore, the eigenvectors and the eigenvalues can be steered according to the structure tensor which is an important representative of the local image structures [22]. The structure tensor is defined as

$$J_\rho(\nabla u_\sigma) := K_\rho * (\nabla u_\sigma \nabla u_\sigma^T). \tag{2.14}$$

This matrix notation also prevents the cancellation of the adjacent gradients having the same direction, but opposite orientation that is a problem present in isotropic nonlinear diffusion for a large σ. The reason that the matrix is convolved with a Gaussian is to integrate the gradients in a neighbourhood of ρ [23]. One eigenvector of the structure tensor is parallel and the other one is perpendicular to the local structure with corresponding eigenvalues μ_1 and μ_2.

The diffusion tensor can use the eigenvectors of the structure tensor to adapt itself to the direction of the local structure. Moreover, the eigenvalues can be adapted according to the goal of the model. Here, we present two anisotropic diffusion models used in the experiments of this thesis.

Edge-Enhancing Diffusion

As the name suggests, this diffusion is specifically designed to favor diffusion along edges and slow it down across edges [22]. Therefore, the edges are preserved and even enhanced up to a scale as the image evolves. This can be achieved by setting ρ to 0 so that one eigenvector of the structure tensor is parallel and the other one is perpendicular to the local image gradient. This leads to the corresponding eigenvalues $\mu_1 = |\nabla u_\sigma|^2$ and $\mu_2 = 0$. Now each direction can be treated separately by tuning the eigenvalues of the diffusion tensor as

$$\lambda_1(\mu_1) := g(\mu_1), \tag{2.15}$$

$$\lambda_2 := 1 \tag{2.16}$$

Coherence-Enhancing Diffusion

Coherence enhancing diffusion is specifically designed to steer the process according to flow-like coherent structures [22]. The goal is to favor diffusion along the flow-like lines and slow it down across them. This model can be achieved by setting $\rho > 0$ and tuning the eigenvalues of the diffusion tensor separately to the eigenvalues of the structure tensor as

$$\lambda_1 := \alpha, \tag{2.17}$$

$$\lambda_2 := \begin{cases} \alpha & (\mu_1 = \mu_2) \\ \alpha + (1-\alpha)\exp\left(\frac{-C}{(\mu_1-\mu_2)^{2m}}\right) & (else) \end{cases} \tag{2.18}$$

for $C > 0$ and $m \in \mathbb{N}$. Here $(\mu_1 - \mu_2)$ is the coherence measure and $0 < \alpha < 1$ ensures that D is uniformly positive definite.

2.2 Discretizations

All the diffusion models mentioned in Section 2.1 are continuous models and need to be discretized to be applied to digital images. A digital image can be thought as a sampling of a continuous image on to a finite pixel grid. With grid sizes h_1, h_2, in x and y directions, respectively, and time step size τ, $u(x_i, y_i, t_k)$ approximates $u^k_{i,j}$ where $x_i = (i - \frac{1}{2})h_1$, $y_j = (j - \frac{1}{2})h_2$ and $t_k = k\tau$ [23].

2.2.1 Linear Diffusion

Using forward differences, one can find the derivative with respect to time as

$$\partial_t u = \frac{u^{k+1}_{i,j} - u^k_{i,j}}{\tau} \tag{2.19}$$

and second derivative with respect to x-axis as

$$\partial_{xx} u = \frac{u^k_{i+1,j} - 2u^k_{i,j} + u^k_{i-1,j}}{h_1^2}. \tag{2.20}$$

Extending this derivatives in y direction, Equation 2.4 becomes [23]

$$\frac{u^{k+1}_{i,j} - u^k_{i,j}}{\tau} = \frac{u^k_{i+1,j} - 2u^k_{i,j} + u^k_{i-1,j}}{h_1^2} + \frac{u^k_{i,j+1} - 2u^k_{i,j} + u^k_{i,j-1}}{h_2^2}.$$

This equation can be explicitly solved for $u^{k+1}_{i,j}$ as

$$u^{k+1}_{i,j} = (1 - 2\frac{\tau}{h_1^2} - 2\frac{\tau}{h_2^2})u^k_{i,j} + \frac{\tau}{h_1^2}u^k_{i+1,j} + \frac{\tau}{h_1^2}u^k_{i-1,j} + \frac{\tau}{h_2^2}u^k_{i,j+1} + \frac{\tau}{h_2^2}u_{i,j-1}.$$

This explicit scheme can be represented with the stencil

0	$\frac{\tau}{h_2^2}$	0
$\frac{\tau}{h_1^2}$	$1-2\frac{\tau}{h_1^2}-2\frac{\tau}{h_2^2}$	$\frac{\tau}{h_1^2}$
0	$\frac{\tau}{h_2^2}$	0

where each weight refers to the corresponding locations that increase from bottom left to top right

(i-1,j+1)	(i,j+1)	(i+1,j+1)
(i-1,j)	(i,j)	(i+1,j)
(i-1,j-1)	(i,j-1)	(i+1,j-1)

Stability. A discrete model is stable when the sum of all weights in its stencil is 1 and the weights are nonnegative [23]. Stencil weights for the linear diffusion sum up to 1 and all non-central weights are nonnegative. Therefore, the condition on stability depends on the nonnegativity of the central weight imposing a limit for the time step:

$$\tau \leq \frac{1}{\frac{2}{h_1^2}+\frac{2}{h_2^2}}. \tag{2.21}$$

Boundaries. Since pixel grid is finite, boundaries require to be treated specially. Neumann boundary conditions can be used to induce 0 gradient at boundaries as:

$$\boldsymbol{n}^T \nabla u = 0 \quad on \quad \partial\Omega \times [0,\infty) \tag{2.22}$$

where $\boldsymbol{n}$ is the outer normal vector at the image boundaries $\partial\Omega$. This can be realized for discrete settings by mirroring the pixels at boundaries.

2.2.2 Isotropic Nonlinear Diffusion

Discretization of the isotropic nonlinear diffusion equation (2.9) with finite differences leads to [23]

$$\begin{aligned}\frac{u_{i,j}^{k+1}-u_{i,j}^{k}}{\tau} &= \frac{1}{h_1}\left(\frac{g_{i+1,j}^k+g_{i,j}^k}{2}\cdot\frac{u_{i+1,j}^k-u_{i,j}^k}{h_1}-\frac{g_{i,j}^k+g_{i-1,j}^k}{2}\cdot\frac{u_{i,j}^k-u_{i-1,j}^k}{h_1}\right)\\ &+\frac{1}{h_2}\left(\frac{g_{i,j+1}^k+g_{i,j}^k}{2}\cdot\frac{u_{i,j+1}^k-u_{i,j}^k}{h_2}-\frac{g_{i,j}^k+g_{i,j-1}^k}{2}\cdot\frac{u_{i,j}^k-u_{i,j-1}^k}{h_2}\right).\end{aligned}$$

When it is solved for $u_{i,j}^{k+1}$, it yields the stencil

0	$\frac{\tau}{2h_2^2}(g_{i,j+1}+g_{i,j})$	0
$\frac{\tau}{2h_1^2}(g_{i,j}+g_{i-1,j})$	$1-\frac{\tau}{2h_1^2}(g_{i+1,j}+2g_{i,j}+g_{i-1,j})$ $-\frac{\tau}{2h_2^2}(g_{i,j+1}+2g_{i,j}+g_{i,j-1})$	$\frac{\tau}{2h_1^2}(g_{i+1,j}+g_{i,j})$
0	$\frac{\tau}{2h_2^2}(g_{i,j}+g_{i,j-1})$	0

Stability. The stencil weights sum up to 1 and non-central weights are nonnegative. Similar to the linear diffusion, stability depends on the nonnegativity of the central weight as

$$\tau \max_{i,j} \left| \frac{(g_{i+1,j} + 2g_{i,j} + g_{i-1,j})}{2h_1^2} + \frac{(g_{i,j+1} + 2g_{i,j} + g_{i,j-1})}{2h_2^2} \right| < 1 \tag{2.23}$$

Since $0 < g_{i,j} \leq 1$, this yields the following limit on time step size:

$$\tau < \frac{1}{\frac{2}{h_1^2} + \frac{2}{h_2^2}}. \tag{2.24}$$

Boundaries. Similar to the linear diffusion, Neumann boundary condition (Equation 2.22) can be used to impose 0 gradient at boundaries by mirroring boundary pixels.

This explicit scheme can be rewritten in a compact way [23]. When we represent the 2-D image as a 1-D vector $\boldsymbol{u} = (u_1, u_2, ..., u_N)^T$, the explicit scheme can be rewritten as a system of equations:

$$\frac{\boldsymbol{u}^{k+1} - \boldsymbol{u}^k}{\tau} = \boldsymbol{A}(\boldsymbol{u}^k)\boldsymbol{u}^k. \tag{2.25}$$

When we let a single index $k(i,j)$ represent the index (i,j), then this $N \times N$ matrix $\boldsymbol{A}$ has entries $a_{k,l}$ satisfying

$$a_{k,l} = \begin{cases} \frac{g_k + g_l}{2h_n^2} & (l \in \mathcal{N}_n(k)), \\ -\sum_{n=1}^{2} \sum_{l \in \mathcal{N}_n(k)} \frac{g_k + g_l}{2h_n^2} & (l = k), \\ 0 & (else), \end{cases} \tag{2.26}$$

where $\mathcal{N}_n(k)$ is representing the neighboring pixels of k in n-directions.

Solving for u^{k+1} gives

$$\boldsymbol{u}^{k+1} = (\boldsymbol{I} + \tau \boldsymbol{A}(\boldsymbol{u}^k))\boldsymbol{u}^k, \tag{2.27}$$

which can be simplified as

$$\boldsymbol{u}^{k+1} = \boldsymbol{Q}(\boldsymbol{u}^k)\boldsymbol{u}^k. \tag{2.28}$$

2.2.3 Anisotropic Nonlinear Diffusion

For $D = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$ equation (2.3) can be rewritten as

$$\partial_t u = div(D \cdot \nabla u) = \partial_x(a\partial_x u) + \partial_x(b\partial_y u) + \partial_y(b\partial_x u) + \partial_y(c\partial_y u). \tag{2.29}$$

Using finite differences for the homogeneous derivative terms as in the isotropic nonlinear diffusion case and central differences for the mixed derivative terms as

$$\partial_x u = \frac{u^k_{i+1,j} - u^k_{i-1,j}}{2h_1}, \tag{2.30}$$

equation (2.29) becomes [23]

$$\begin{aligned}\frac{u^{k+1}_{i,j} - u^k_{i,j}}{\tau} = \frac{1}{h_1}&\left(\frac{a^k_{i+1,j}+a^k_{i,j}}{2}\cdot\frac{u^k_{i+1,j}-u^k_{i,j}}{h_1} - \frac{a^k_{i,j}+a^k_{i-1,j}}{2}\cdot\frac{u^k_{i,j}-u^k_{i-1,j}}{h_1}\right)\\ +\frac{1}{2h_1}&\left(b^k_{i+1,j}\cdot\frac{u^k_{i+1,j+1}-u^k_{i+1,j-1}}{2h_2} - b^k_{i-1,j}\cdot\frac{u^k_{i-1,j+1}-u^k_{i-1,j-1}}{2h_2}\right)\\ +\frac{1}{2h_2}&\left(b^k_{i,j+1}\cdot\frac{u^k_{i+1,j+1}-u^k_{i-1,j+1}}{2h_1} - b^k_{i,j-1}\cdot\frac{u^k_{i+1,j-1}-u^k_{i-1,j-1}}{2h_1}\right)\\ +\frac{1}{h_2}&\left(\frac{c^k_{i,j+1}+c^k_{i,j}}{2}\cdot\frac{u^k_{i,j+1}-u^k_{i,j}}{h_2} - \frac{c^k_{i,j}+c^k_{i,j-1}}{2}\cdot\frac{u^k_{i,j}-u^k_{i,j-1}}{h_2}\right)\end{aligned}$$

When it is solved for $u^{k+1}_{i,j}$, yields the stencil

$\frac{\tau}{4h_1h_2}(-b_{i-1,j} - b_{i,j+1})$	$\frac{\tau}{2h_2^2}(c_{i,j+1} + c_{i,j})$	$\frac{\tau}{4h_1h_2}(-b_{i+1,j} - b_{i,j+1})$
$\frac{\tau}{2h_1^2}(a_{i-1,j} + a_{i,j})$	$1 - \frac{\tau}{2h_1^2}(a_{i-1,j} + 2a_{i,j} + a_{i+1,j})$ $- \frac{\tau}{2h_2^2}(c_{i,j-1} + 2c_{i,j} + c_{i,j+1})$	$\frac{\tau}{2h_1^2}(a_{i+1,j} + a_{i,j})$
$\frac{\tau}{4h_1h_2}(-b_{i-1,j} - b_{i,j-1})$	$\frac{\tau}{2h_2^2}(c_{i,j-1} + c_{i,j})$	$\frac{\tau}{4h_1h_2}(-b_{i+1,j} - b_{i,j-1})$

Stability. When D is positive semidefinite, a and c are both nonnegative. However, b might have any sign that causes the weights marked as red have arbitrary signs. Therefore, this standard discretization does not guarantee stability. However, there exists nonnegative discretizations for a positive definite D, if $h_1 = h_2$ and the condition below holds [23].

$$cond(D) = \frac{\lambda_{max}}{\lambda_{min}} \leq \frac{1 + \frac{1}{2}\sqrt{2}}{1 - \frac{1}{2}\sqrt{2}} = 3 + 2\sqrt{2} \tag{2.31}$$

Although this condition limits anisotropy, it guarantees stability. This nonnegative discretization gives the stencil

$\frac{\tau(\lvert b_{i-1,j+1}\rvert-b_{i-1,j+1})}{4h_1h_2} + \frac{\tau(\lvert b_{i,j}\rvert-b_{i,j})}{4h_1h_2}$	$\frac{\tau(c_{i,j+1}+c_{i,j})}{2h_2^2} - \frac{\tau(\lvert b_{i,j+1}\rvert+\lvert b_{i,j}\rvert)}{2h_1h_2}$	$\frac{\tau(\lvert b_{i+1,j+1}\rvert+b_{i+1,j+1})}{4h_1h_2} + \frac{\tau(\lvert b_{i,j}\rvert+b_{i,j})}{4h_1h_2}$
$\frac{\tau(a_{i-1,j}+a_{i,j})}{2h_1^2} - \frac{\tau(\lvert b_{i-1,j}\rvert+\lvert b_{i,j}\rvert)}{2h_1h_2}$	$1 - \frac{\tau(a_{i-1,j}+2a_{i,j}+a_{i+1,j})}{2h_1^2} - \frac{\tau(\lvert b_{i-1,j+1}\rvert-b_{i-1,j+1}+\lvert b_{i+1,j+1}\rvert+b_{i+1,j+1})}{4h_1h_2} - \frac{\tau(\lvert b_{i-1,j-1}\rvert+b_{i-1,j-1}+\lvert b_{i+1,j-1}\rvert-b_{i+1,j-1})}{4h_1h_2} + \frac{\tau(\lvert b_{i-1,j}\rvert+\lvert b_{i+1,j}\rvert+\lvert b_{i,j-1}\rvert+\lvert b_{i,j+1}\rvert+2\lvert b_{i,j}\rvert)}{2h_1h_2} - \frac{\tau(c_{i,j-1}+2c_{i,j}+c_{i,j+1})}{2h_2^2}$	$\frac{\tau(a_{i+1,j}+a_{i,j})}{2h_1^2} - \frac{\tau(\lvert b_{i+1,j}\rvert+\lvert b_{i,j}\rvert)}{2h_1h_2}$
$\frac{\tau(\lvert b_{i-1,j-1}\rvert+b_{i-1,j-1})}{4h_1h_2} + \frac{\tau(\lvert b_{i,j}\rvert+b_{i,j})}{4h_1h_2}$	$\frac{\tau(c_{i,j-1}+c_{i,j})}{2h_2^2} - \frac{\tau(\lvert b_{i,j-1}\rvert+\lvert b_{i,j}\rvert)}{2h_1h_2}$	$\frac{\tau(\lvert b_{i+1,j-1}\rvert-b_{i+1,j-1})}{4h_1h_2} + \frac{\tau(\lvert b_{i,j}\rvert-b_{i,j})}{4h_1h_2}$

Boundaries. Neumann boundary condition can also be realized for this model by mirroring pixels at the boundaries.

2.3 Fast Explicit Diffusion Scheme

When the diffusion models are discretized with the explicit scheme as in Section 2.2, each model has a limit for time step size to be stable over time [23]. The explicit scheme is simple to implement; however, it is really slow with these time step restrictions. With such small time steps, the time to reach a certain stopping time is long. However, one advantage is that it is suitable for parallel schemes. As can be seen from the discretizations, each new grey value computation at time step $k+1$ only depends on the grey values in time step k. Therefore, each computation is completely independent of each other. However, this is not the end of the story. This scheme has been shown to be sped up in sequential programmes by Fast Explicit Scheme (FED) which can also be applied to parallel programmes.

The idea is as follows. Grewenig et al. [8] [7] showed that a box filter of length $2n+1$ can be obtained by convolving n explicit linear diffusion filters having different time step sizes. These step sizes can be computed from the stable step size τ_{stable} as

$$\tau_i = \tau_{stable}\frac{1}{2cos^2(\pi\frac{2i+1}{4n+2})} \quad i = 0, 1, ..., n-1. \tag{2.32}$$

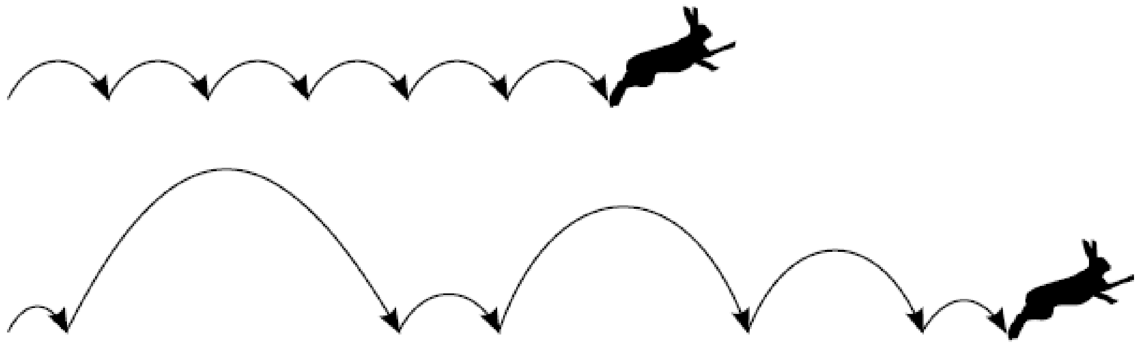


Figure 2.1: Schematic comparison of FED to a standard explicit scheme. Illustration: P. Gwosdek [13]

After one cycle of diffusion schemes with varying τ_is, the box filter is approximated. This one cycle takes time

$$\theta = \tau_0 + \tau_1 + ... + \tau_{n-1} = \tau_{stable} \cdot \frac{n^2 + n}{3}. \tag{2.33}$$

From the central limit theorem, it is also known that a Gaussian can be well-approximated by convolving box filters iteratively [23]. So can the linear diffusion. This means that repetition of this FED cycle can approximate linear diffusion.

In this framework up to half of the τ_is violate time step size τ_{stable} for stability as in Figure 2.1; however, one cycle reaches the box filter at the end which is already stable. Hence, further iterations of it for the diffusion process do not violate the stability conditions.

As a result, Equation 2.33 shows that with n steps a stopping time of order $O(n^2)$ can be reached using FED scheme instead of $O(n)$. It offers a remarkable speed up and even outperforms semi-implicit and AOS schemes where they are applicable [23]. It can also be generalized to all diffusion models as long as the iteration matrix $\boldsymbol{A}(\boldsymbol{u}^k)$ is symmetric and constant during one FED cycle. Since a box filter is already efficient, using FED scheme for the linear diffusion is not necessary.

2.4 Scale-Space Concept

Images include features which only appear at a certain range of scales. This requires to analyse images in different scales. Scale-spaces create a multiscale image representation by embeding image $f : \mathbb{R}^2 \to \mathbb{R}$ into a family

$$\{T_t f | t > 0\} \tag{2.34}$$

with some requirements [23]. These requirements can be grouped into three classes:

- **architectural properties:** such as semi-group property

$$T_0 f = f$$
$$T_{t+s} f = T_t(T_s f) \quad \forall s, t \geq 0$$

- **simplification:** such as causality, maximum-minimum principle. These requirements are to make sure that images gets simpler as the scale increases i.e. no new structures are introduced.

- **invariances:** such as translation and rotation invariances

One of the oldest scale-spaces is Gaussian scale-space that was invented by Iijima [10] and reinvented by Witkin [26] in the western world. PDEs of diffusion processes also constitutes scale-space with the original image as a initial value. Scale-space properties of the diffusion models, which we discuss in this thesis, are analysed comprehensively by Weickert in [22].

2.5 Mean Squared Error

The Mean Squared Error (MSE) measures the mean of the squared differences between actual and estimated values. It can be formulated for actual data $\boldsymbol{X}$ and its estimation $\bar{\boldsymbol{X}}$ as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{X}_i - \bar{\boldsymbol{X}}_i)^2 \tag{2.35}$$

Chapter 3

Diffusion Echo

3.1 What is the Diffusion Echo?

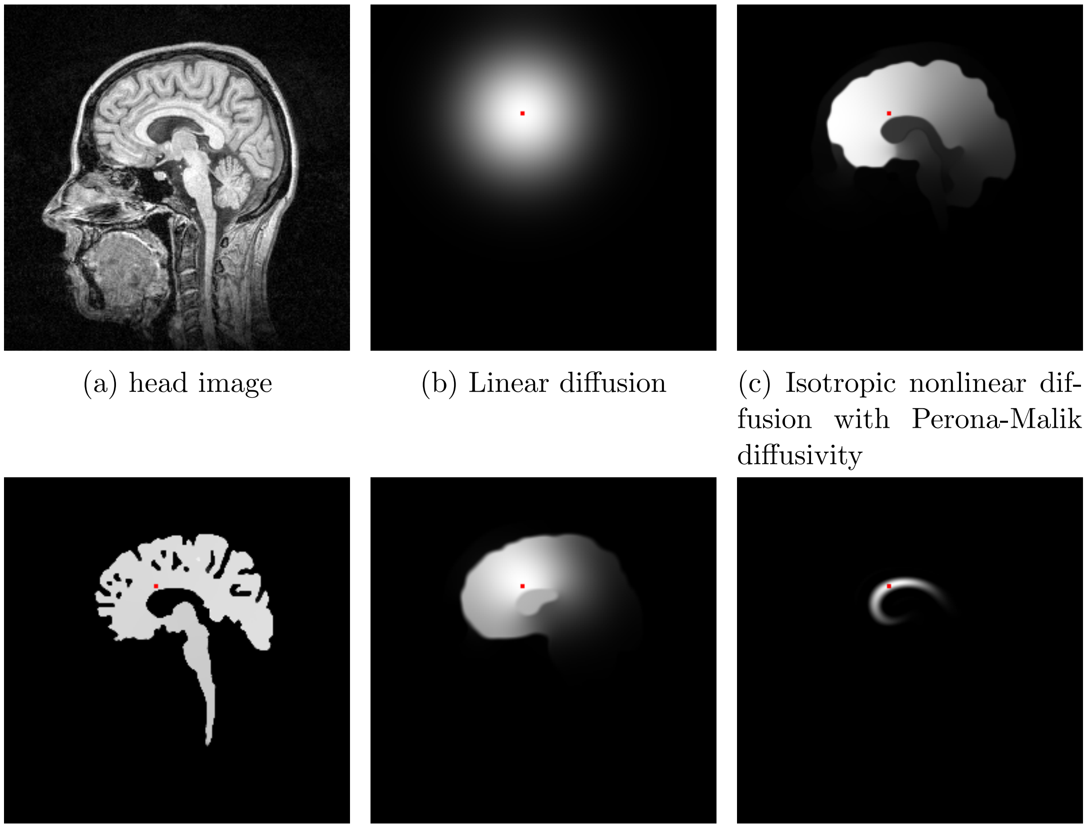
As diffusion processes offer more and more transparent mathematical modelling and well suited models for specific applications in image processing and computer vision, it is more and more important to understand the underlying true process of any existing or newly designed model. For the linear diffusion this is easy. There exists a closed form solution to its PDE as a Gaussian [9, pp. 43-56]. This makes the model easy to interpret and offers a intuitive description to it. It tells us how exactly the diffusion filter looks like. Although nonlinear models may provide more plausible results, they lack of such an interpretation.

Finding closed form solutions for PDEs of nonlinear diffusion is still an ongoing research. The difficulty comes from the fact that the nonlinear processes are highly dependent on the local image structure. Meaning that the diffusion filter for each pixel is different and has the size of the image. However, Dam and Nielsen [6] showed that there is a way to explicitly compute the shift variant diffusion filter. At each iteration of the diffusion process, the filter weights are already computed. They use these weights to form the filter. They call each filter formed for a single pixel its *Diffusion Echo*. The name speaks for itself: it displays explicitly the behaviour of one pixel on the diffusion model. In Figure 3.1, how the diffusion echoes of the same pixel can change with the different diffusion models can be seen.

In their work [6] there are two types of diffusion echoes: the diffusion echo source distribution and the diffusion echo drain distribution.

Diffusion Echo: Source

The diffusion echo source distribution defines how a specific pixel p contributes to the each pixel q in the rest of the image [6]. It can be easily computed by creating an auxiliary image as in Figure 3.2. This image has 1 at the pixel location of pixel p and 0 elsewhere, (discrete impulse function located at p). This image can be iterated as the original image itself with the exact same weights computed for the original image.



(a) head image

(b) Linear diffusion

(c) Isotropic nonlinear diffusion with Perona-Malik diffusivity

(d) Isotropic nonlinear diffusion with Weickert's diffusivity

(e) Edge-enhancing diffusion

(f) Coherence-enhancing diffusion

Figure 3.1: Diffusion echoes (b)-(f) of the same reference point of 256×256 head image (a) for different diffusion models.

Diffusion Echo: Drain

Diffusion echo drain distribution is the opposite of the source distribution. It defines how a pixel p is contributed by the rest of the image pixels [6]. This defines exactly the underlying diffusion filter for pixel p.

These two distributions are the same for the linear diffusion but different for the nonlinear models [6]. The diffusion echo distribution used in this thesis is the source distribution. For the simplicity, we will refer to it as diffusion echo in the rest of this thesis.

Since diffusion echo shows how a pixel contributes to other pixels over time, it can be regarded as the summary of the diffusion at that pixel up to a scale. Therefore, it offers a well visualized description of the process used. We can obtain true information and deductions related to the image structures by this description. Besides the different diffusion schemes, it reveals the differences between various discretiza-

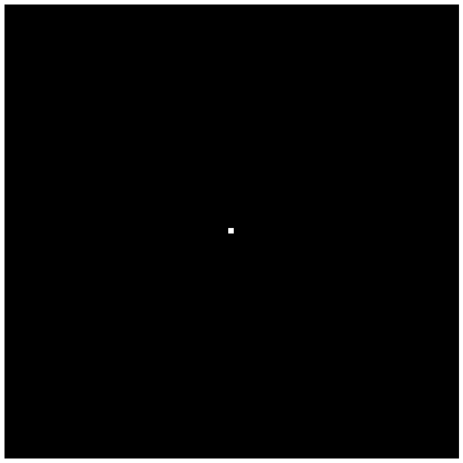


Figure 3.2: Auxiliary image of size 256×256 for the pixel at location (128,128)

tions and parameters. This steers the diffusion from being information simplifying to information gathering technique. As Dam and Nielson states: ”Diffusion knows” [6].

With the implementation in this thesis, it is possible to compute many diffusion echoes within seconds. Therefore, we can analyse more diffusion echoes and larger stopping times. Detailed analysis on each diffusion model is explained in the following section.

3.2 Diffusion Echoes of Different Diffusion Models

3.2.1 Linear Diffusion

Figure 3.3 shows 12 of 64 diffusion echoes computed for the linear diffusion of the 256×256 head image together with the resulting image. Red dots are added after the computations to mark the pixels that the diffusion echoes belong to for better visualization. These pixels are selected uniformly i.e. 1 pixel is chosen in each 32×32 pixel block of an image. We see that each diffusion echo is a Gaussian. At the boundaries, contribution of the mirrored Gaussians are also present. This perfectly presents the modeling of the diffusion model: it is independent of the image itself and uses mirrored pixels at the boundaries. In Figure 3.4, we see the same experiment with different stopping times for the same reference pixel. We can also realize the relation between stopping time T and the standard deviation of Gaussian σ as $T = \frac{1}{2}\sigma^2$. As the stopping time T is multiplied by 4 from Figure 3.4a to Figure 3.4c, we observe that the σ of the Gaussian gets doubled.

3.2.2 Isotropic Nonlinear Diffusion

Figure 3.5 shows 12 of 64 uniformly sampled diffusion echoes for the isotropic nonlinear diffusion with Perona-Malik diffusivity of the 256×256 head image together with the smoothed image. This sample is more interesting than its linear counterpart. Each echo is a very coarse representation of the segment it belongs to. This

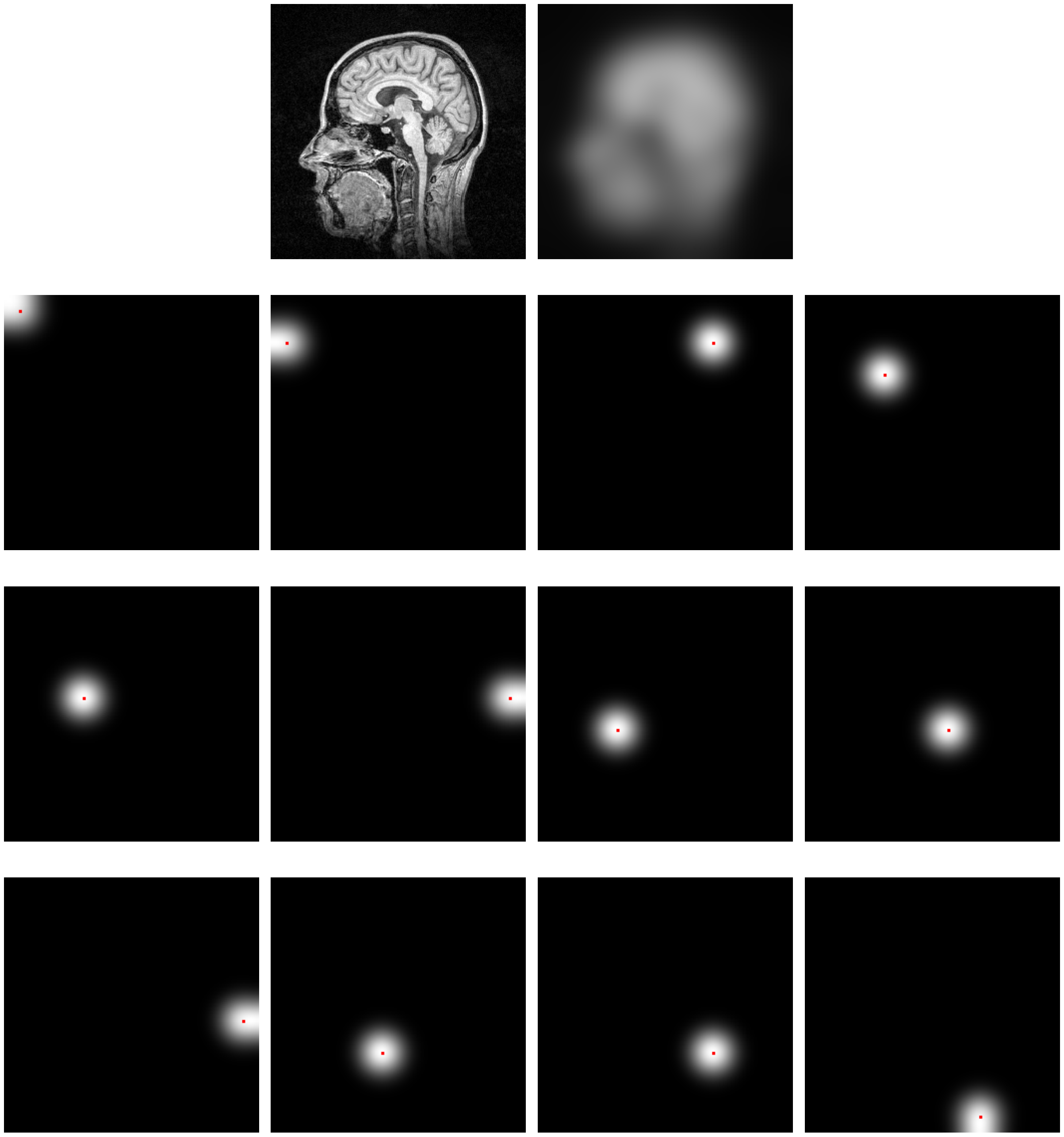
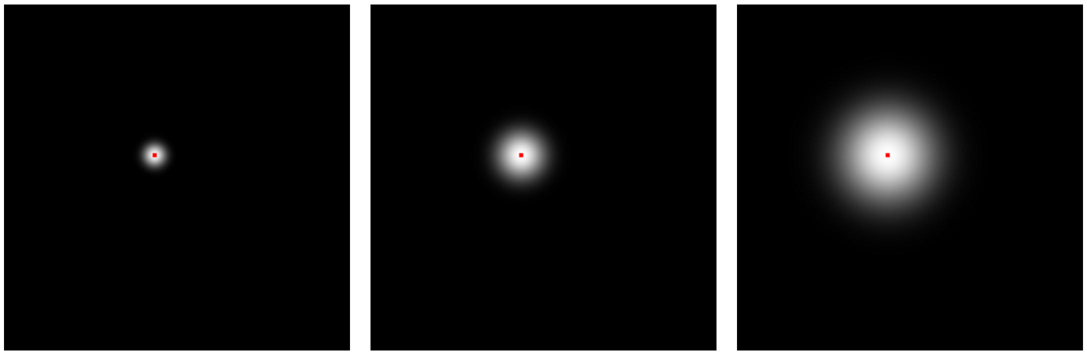


Figure 3.3: Original image, smoothed image and 12 of 64 diffusion echoes after linear diffusion with T=100, $\tau_{stable} = 0.25$, FED cycles=5, and running time=88ms. Echoes are rescaled from their original floating point range [0, 1] to integer range [0, 255].

comes from the nature of how this model is designed. It slows the diffusion down at edges. However, this diffusivity does not decay sufficiently fast to give sharp segment boundaries. We see in Figure 3.6 the same experiment with different stopping times. The edges do not survive long enough for segment-like echoes. It is clear that when the stopping time is small, a segment is visible but not complete. However, when the stopping time is increased to let the pixel diffuse through the whole segment, then the



(a) T=16 (b) T=64 (c) T=256

Figure 3.4: Diffusion echoes for linear diffusion of the same reference point of the head image with different stopping times T.

edges are blurred. This time vs. sharpness problem can be addressed by Weickert's diffusivity.

In Figure 3.7, we present 12 of 64 echoes of the isotropic nonlinear diffusion with Weickert's diffusivity on the same image. Since it decreases more rapidly than Perona-Malik diffusivity, it requires longer diffusion times. Due to the same reason it also gives sharper edges. All the echoes are relatively well-located segments that the sampled pixels belong to. In Figure 3.8, we see how long the edges survive around a single pixel as the diffusion time increases. The other parameters are the same as in Figure 3.7. With this model it is more likely to find segment-like echoes with appropriate parameter selection. Nevertheless, there are no new parameters introduced. The same parameters as in smoothing applications can be chosen with larger stopping times. The diffusion echoes acquired with this model can be useful for segmentation algorithms.

3.2.3 Edge-Enhancing Diffusion

Intuitively, edge-enhancing diffusion seems a good choice for segmentation-like echoes. It is designed to enhance edges so that they survive longer. Pruebab image (128×128) can be an appropriate test case for this diffusion model. In Figure 3.9 we see 12 of 64 diffusion echoes acquired with this model. We see that the edges survive longer, however, the corners are rounded. Figure 3.10 shows the same experiment with different stopping times. We can see that the edges do not survive as long as they do in isotropic nonlinear diffusion with Weickert's diffusivity. In order to use this model for segmentation, post processing would be required. One example would be tracing the corners back to their original locations in a scale-space manner together with an adaptive sampling that avoids pixel selection near edges.

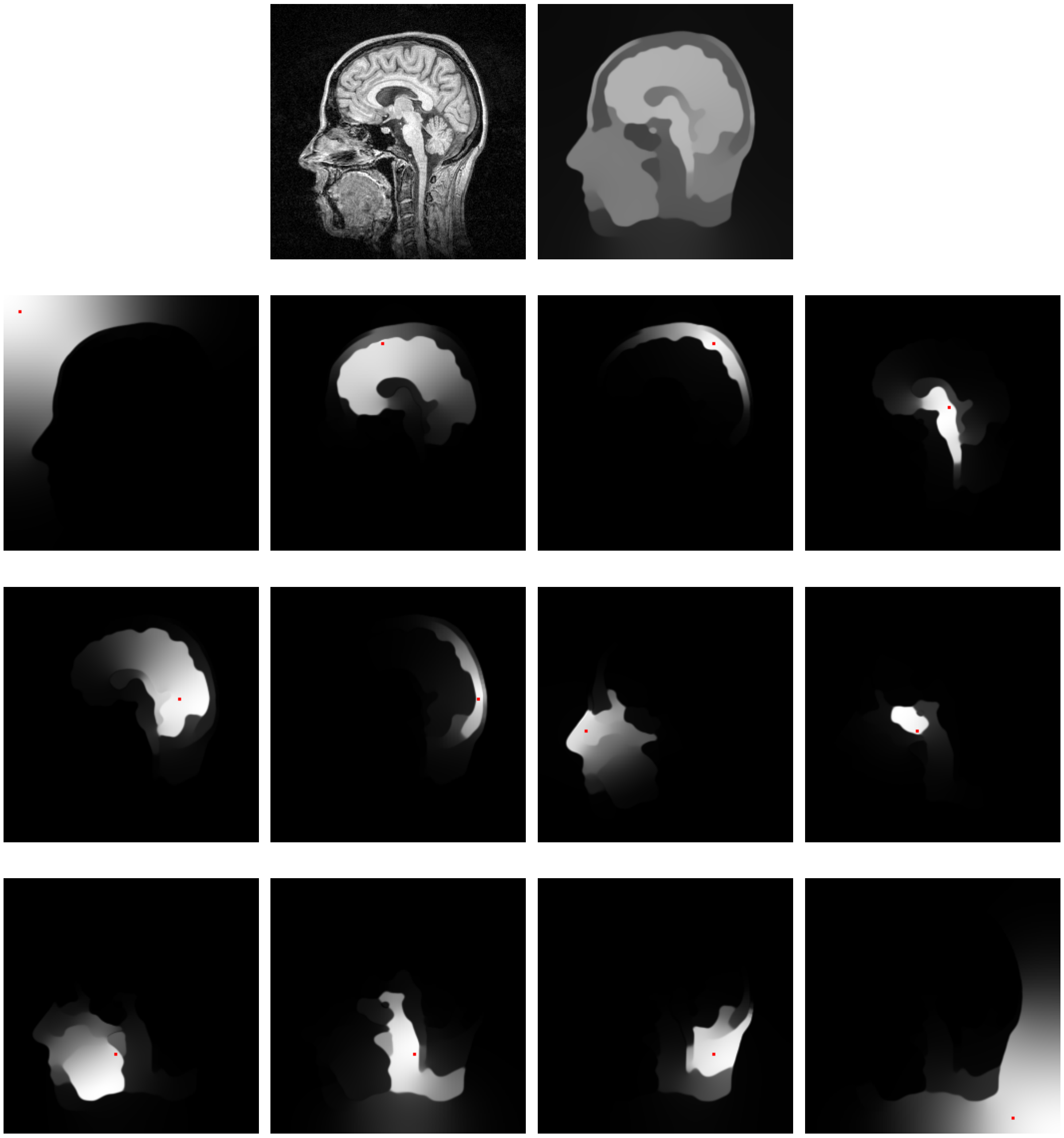
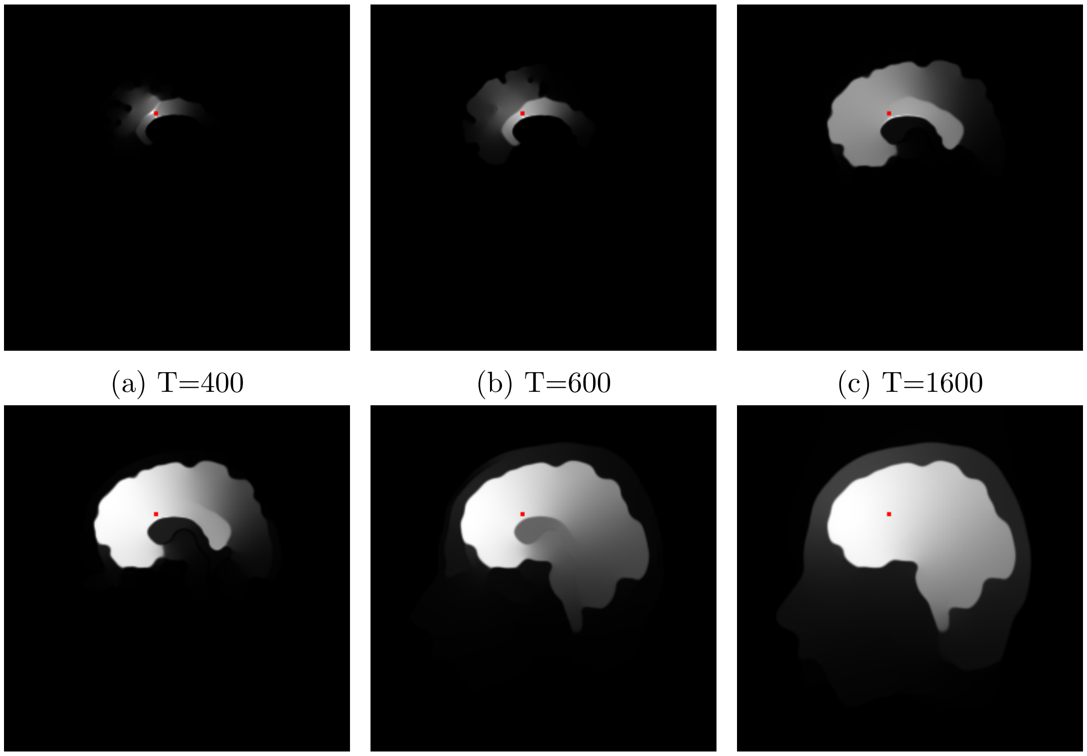


Figure 3.5: Original image, smoothed image and 12 of 64 diffusion echoes after isotropic nonlinear diffusion with T=3000, FED cycles=10, $\lambda = 1$, $\tau_{stable} = 0.24$, $\sigma = 0.5$, and running time=1138ms. Echoes are rescaled from their original floating point range [0, 1] to integer range [0, 255].

3.2.4 Coherence-Enhancing Diffusion

This diffusion model is intuitively expected to fail for segment-like echoes. Still it raises curiosity about how the echoes would look like. Figure 3.11 shows 12 of 64 echoes computed for coherence-enhancing diffusion on 300×300 finger image. It is a good test image for this model. The resulting echoes barely give any idea about



(a) T=400 (b) T=600 (c) T=1600

(d) T=2000 (e) T=4000 (f) T=6000

Figure 3.6: Diffusion echoes for isotropic nonlinear diffusion with Perona-Malik diffusivity of the same reference point of the head image with different stopping times T.

the segments, however, describe the model to the fullest. Each pixel spreads its value in the direction of the coherent structures. In Figure 3.12, we present the same experiment with varying stopping times. We can see the spread of the same reference pixel over time in the direction of coherence in a better way.

As can be seen from these figures, the samples consisting 64 diffusion echoes are computed in a matter of seconds. With the efficient computation in this thesis, it is now possible to see the effect of any parameter on any diffusion model. The advantage here is two fold: Firstly, it may lead to finding the most appropriate model for any application. Secondly, it may lead to modifying the existing models or discovering new ones with a visual support. As a result, diffusion can be used to gather information on images rather than simplifying the image [6].

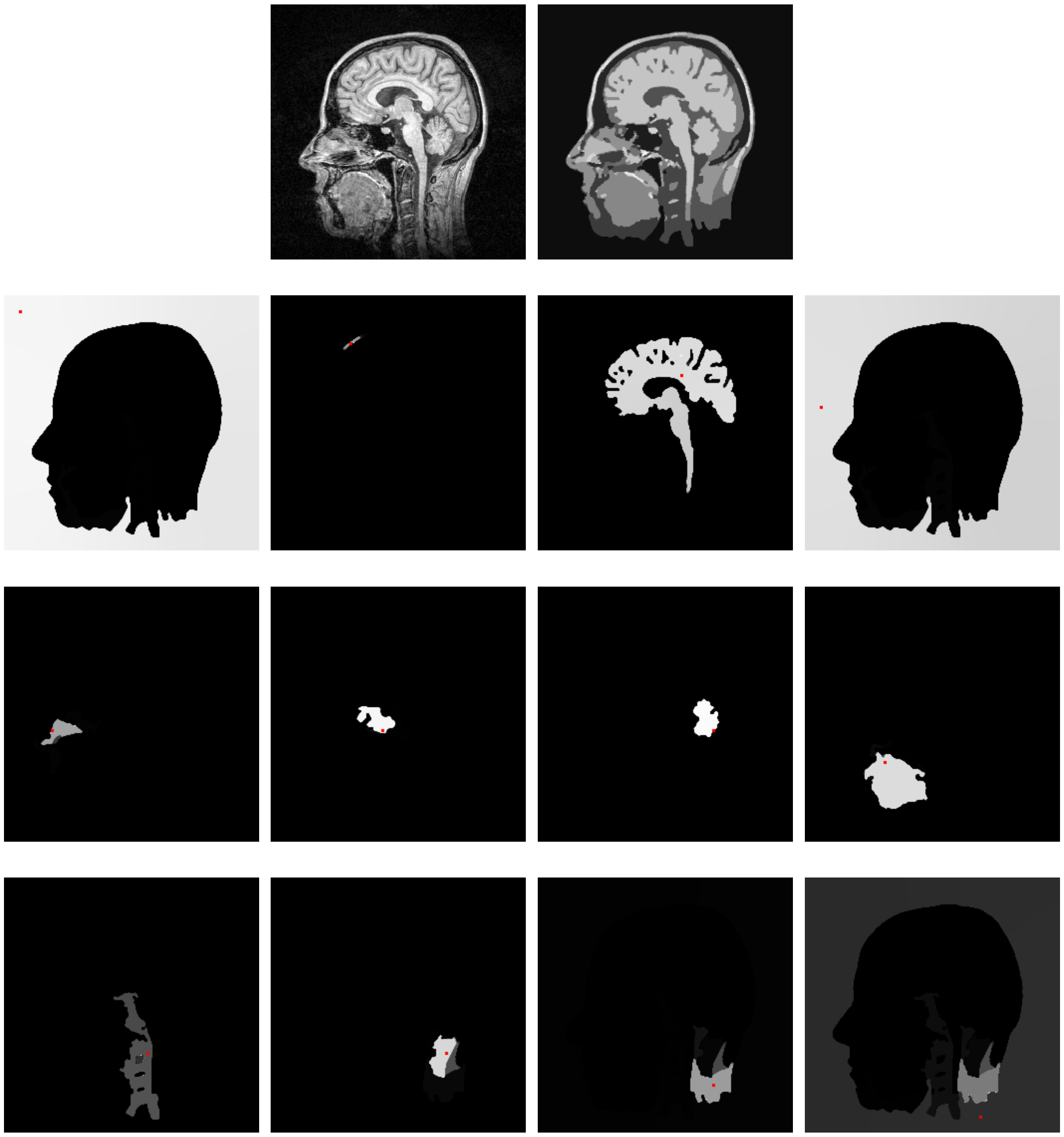
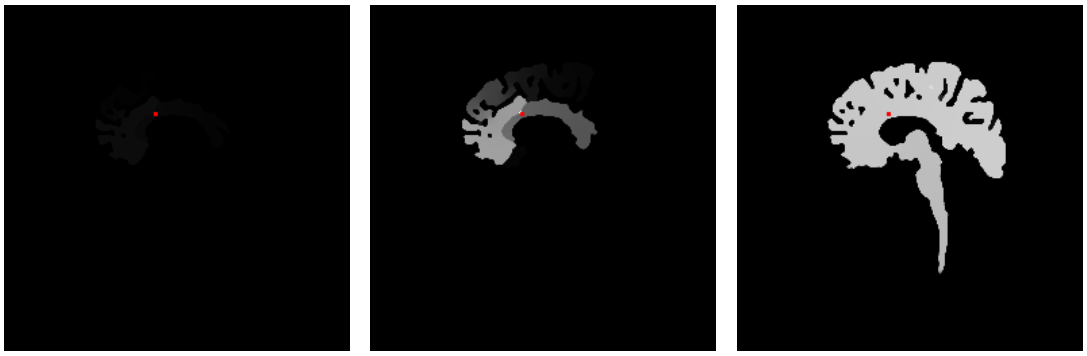


Figure 3.7: Original image, smoothed image and 12 of 64 diffusion echoes after isotropic linear diffusion with Weickert's diffusivity with T=100000, FED cycles=25, λ=3, τ_{stable} = 0.24, σ = 0.5, and running time=10410ms. Echoes are rescaled from their original floating point range [0, 1] to integer range [0, 255].



(a) T=8000 (b) T=10000 (c) T=100000

Figure 3.8: Diffusion echoes for isotropic nonlinear diffusion with Weickert's diffusivity of the same reference point of the head image with different stopping times T.

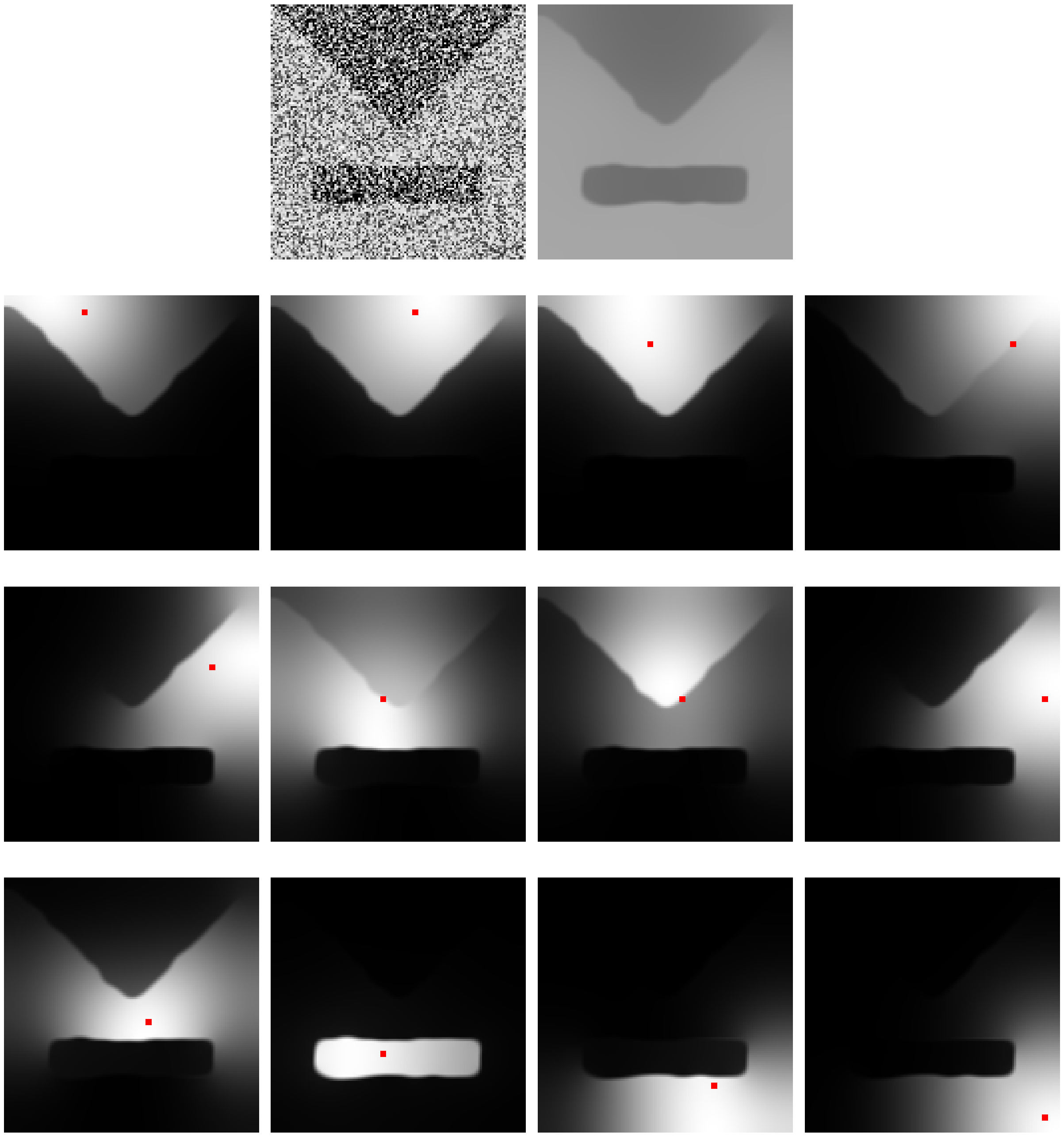
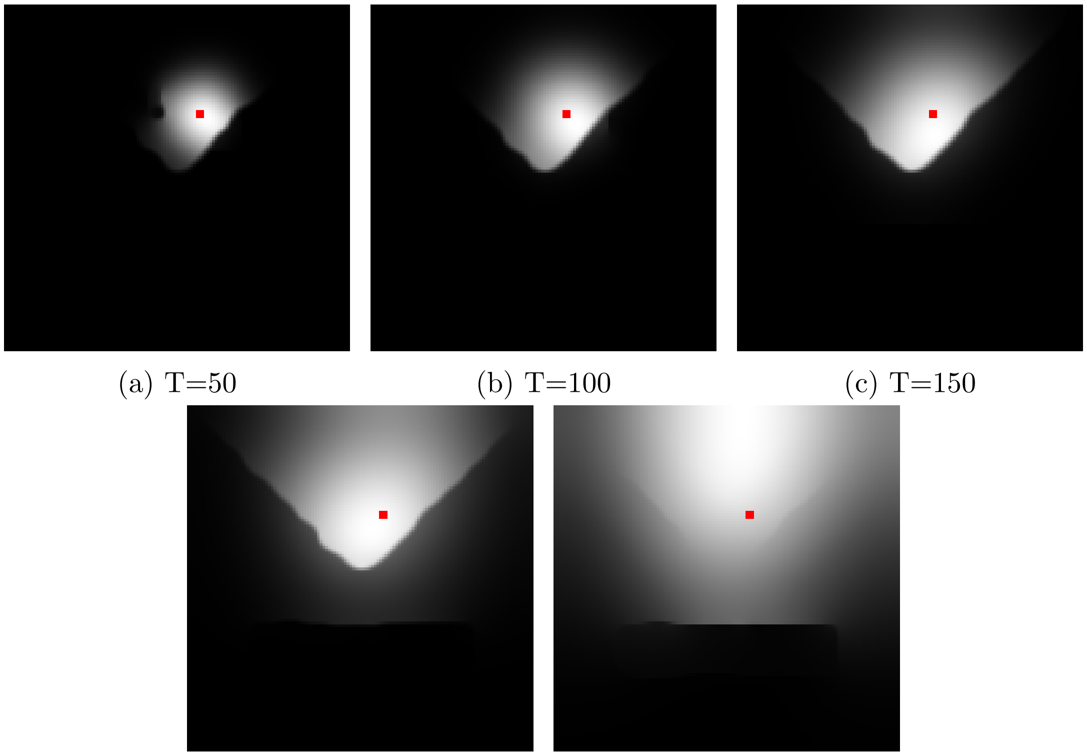


Figure 3.9: Original image, smoothed image and 12 of 64 diffusion echoes after edge-enhancing diffusion with T=700, FED cycles=10, $\lambda = 3$, $\tau_{stable} = 0.24$, $\sigma = 2.5$, and running time=217ms. Echoes are rescaled from their original floating point range [0, 1] to integer range [0, 255].


(a) T=50 (b) T=100 (c) T=150

(d) T=180 (e) T=250

Figure 3.10: Diffusion echoes for edge-enhancing diffusion of the same reference point of the pruebab image with different stopping times T.

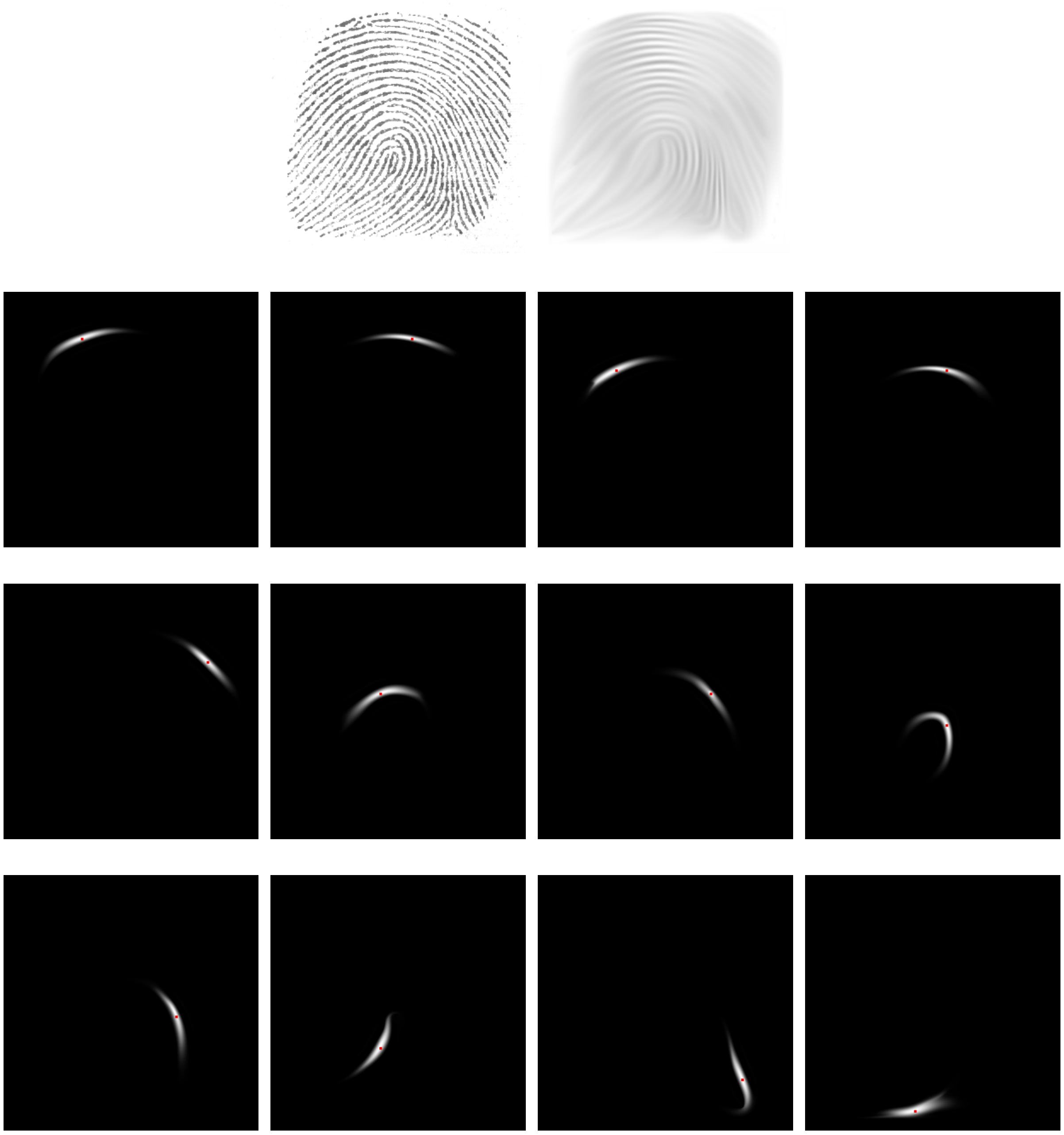
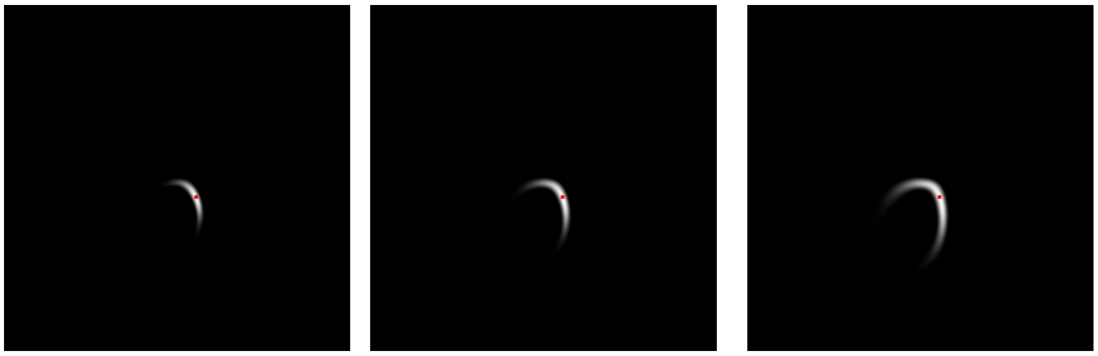


Figure 3.11: Original image, smoothed image and 12 of 64 diffusion echoes after coherence-enhancing diffusion with T=300, FED cycles=10, $C = 1$, $\alpha = 0.001$, $\tau_{stable} = 0.24$, $\sigma = 0.5$, and running time=887ms. Echoes are rescaled from their original floating point range [0, 1] to integer range [0, 255].



(a) T=10 (b) T=50 (c) T=100

Figure 3.12: Diffusion echoes for coherence-enhancing diffusion of the same reference point of the finger image with different stopping times T.

Chapter 4

Efficient Computation of the Diffusion Echo

In this chapter, we will explain the efficient computation of the diffusion echoes in detail. First, we will give a general overview of GPU computing and CUDA. Subsequently, we will explain how to efficiently compute diffusion echoes in a parallel way on GPU using CUDA. Then we will present experimental results with a wide range of parameters. The chapter will end with a brief discussion.

4.1 GPU Computing

4.1.1 Parallel Programming on GPU

The need for fast computations has increased dramatically over the last decades with increasing data sizes in scientific computations, complex problems and real-time applications. However, speeding up CPUs cannot compensate for this need anymore. That is why researchers exploited the idea of parallelism. As Owes et al. states in their comprehensive overview on GPU computing [15]: "Parallelism is the future of computing".

Even though, parallelism is very-well studied on CPUs, it offers tremendous speed ups when exploited on GPUs due to their architecture. The GPUs today are not only very powerful graphics engines as the game industry increasingly demands, but also highly parallel programmable processors that outpace their CPU counterparts [15]. CPU architecture is designed in such a way that the pipeline is divided in time. Each unit in the pipeline processes one stage of the whole process of a group of elements. Output of one unit is input of the next one. Hence, all units can be used at the same time in an efficient way. The GPU pipeline is, however, very different. It divides the resources in space. Output of one unit in pipeline is input of another. Each of these units is a highly specialized hardware.

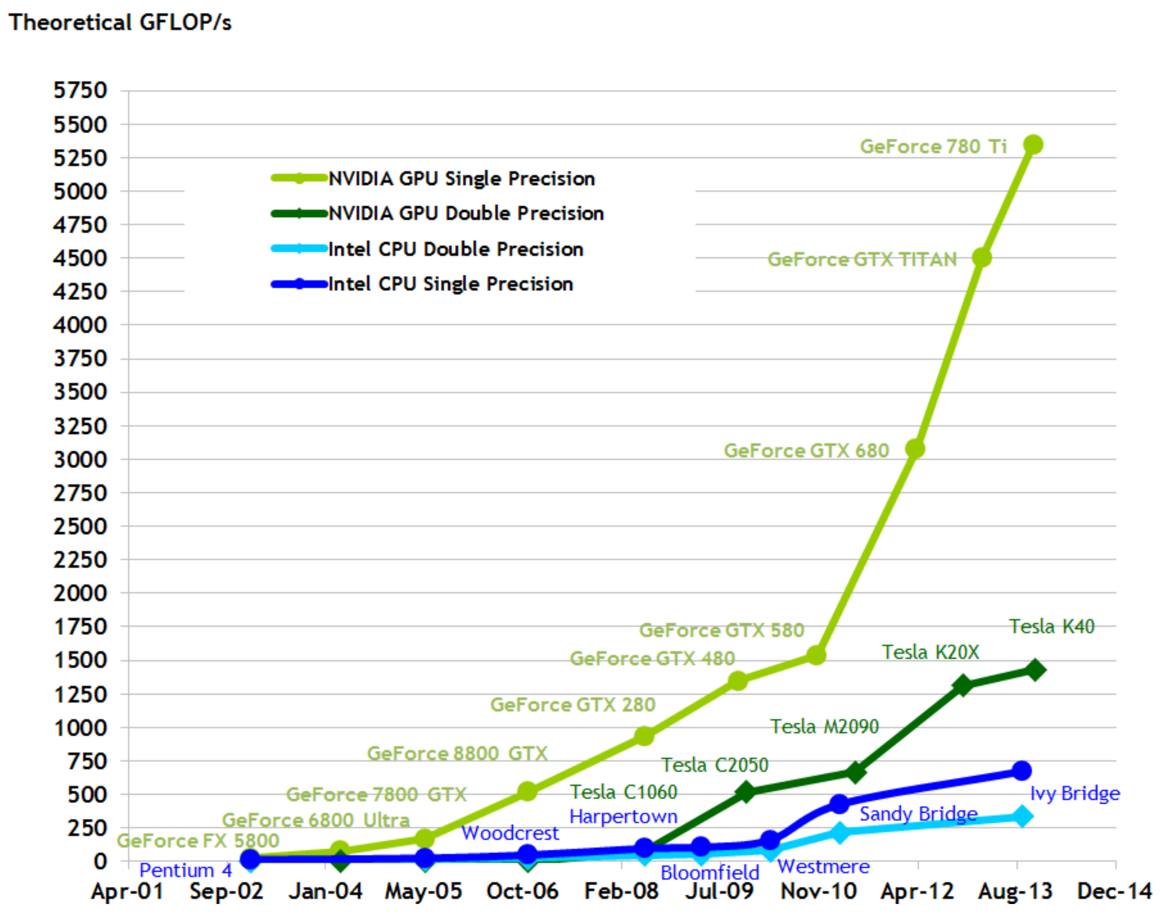


Figure 4.1: Floating-point operations per second for CPU and GPU [2]

4.1.2 History of Parallel Programming on GPU

In the past, the GPUs were designed as special-purpose fixed function processors for graphics computations with several parallel programmable units. They required hacky and very low-level programming with graphics terminology in order to use these programmable parts. However, today the GPUs are well designed as a single parallel programmable unit with several special-purpose fixed function units [15]. Figure 4.1 shows how GPUs are improving and getting faster over years. Today, ten thousands of threads can run in parallel on cores of GPU multiprocessors while CPUs can only afford tens of them. thousands of threads are even expected by the architecture for full utilization [20]. Figure 4.2 shows the organization of these multiprocessors in the modern GPUs. A closer look of a streaming multiprocessor is also seen in Figure 4.3.

4.1.3 General-Purpose Computation on GPU

Even though the graphics pipeline is still used in GPUs, parallel programs are not necessarily graphics related [15]. This leads to the idea of General-Purpose computation on GPU (GPGPU). This idea makes it possible to use the huge performance potential of the GPUs for scientific computations and applications on large data. The increas-

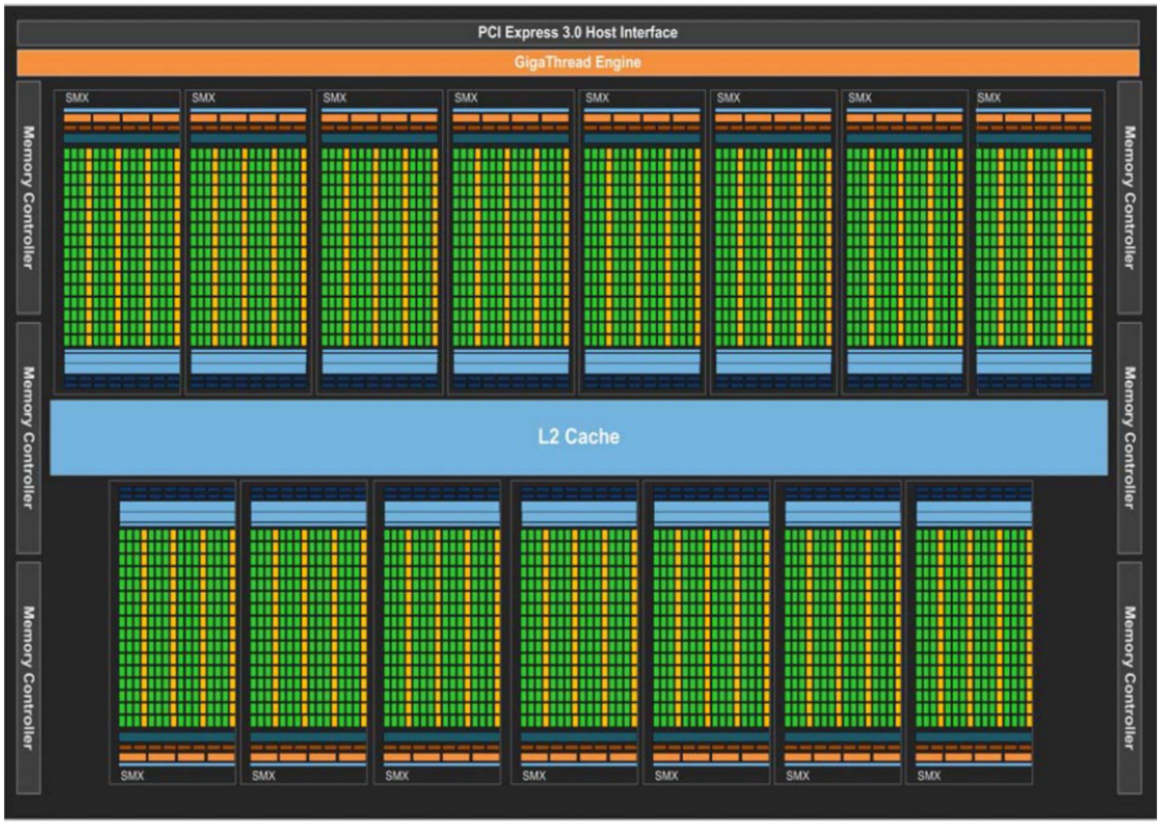


Figure 4.2: Organization of multiprocessors in a modern GPU [20]

ing image sizes make parallel programming very appealing to image processing and computer vision tasks. The basic model in GPGPU programming is single-program multiple-data (SPMD) [15]. This means that the same program is run by multiple threads independently of each other. Therefore, one program is highly parallelizible when the computation of each data point is independent of each other. In Section 2.2, stencils of discrete diffusion filters show that each pixel at a time step is computed independent of each other. Therefore, explicit schemes for diffusion processes are appropriate for GPGPU.

4.2 Parallel Programming with CUDA

As the GPGPUs got more and more popular, many program development platforms were offered. Convergence of several parallel programmable GPU parts into a single programmable part led to convergence of separate instruction sets into a single instruction set. Having a high-level programming language for one single programmable unit made the programmers more focused on the programming task at hand rather than the hardware specifications [15].

One of such high-level parallel programming languages is CUDA, which was introduced by NVIDIA in 2006. CUDA is a scalable parallel programming model and

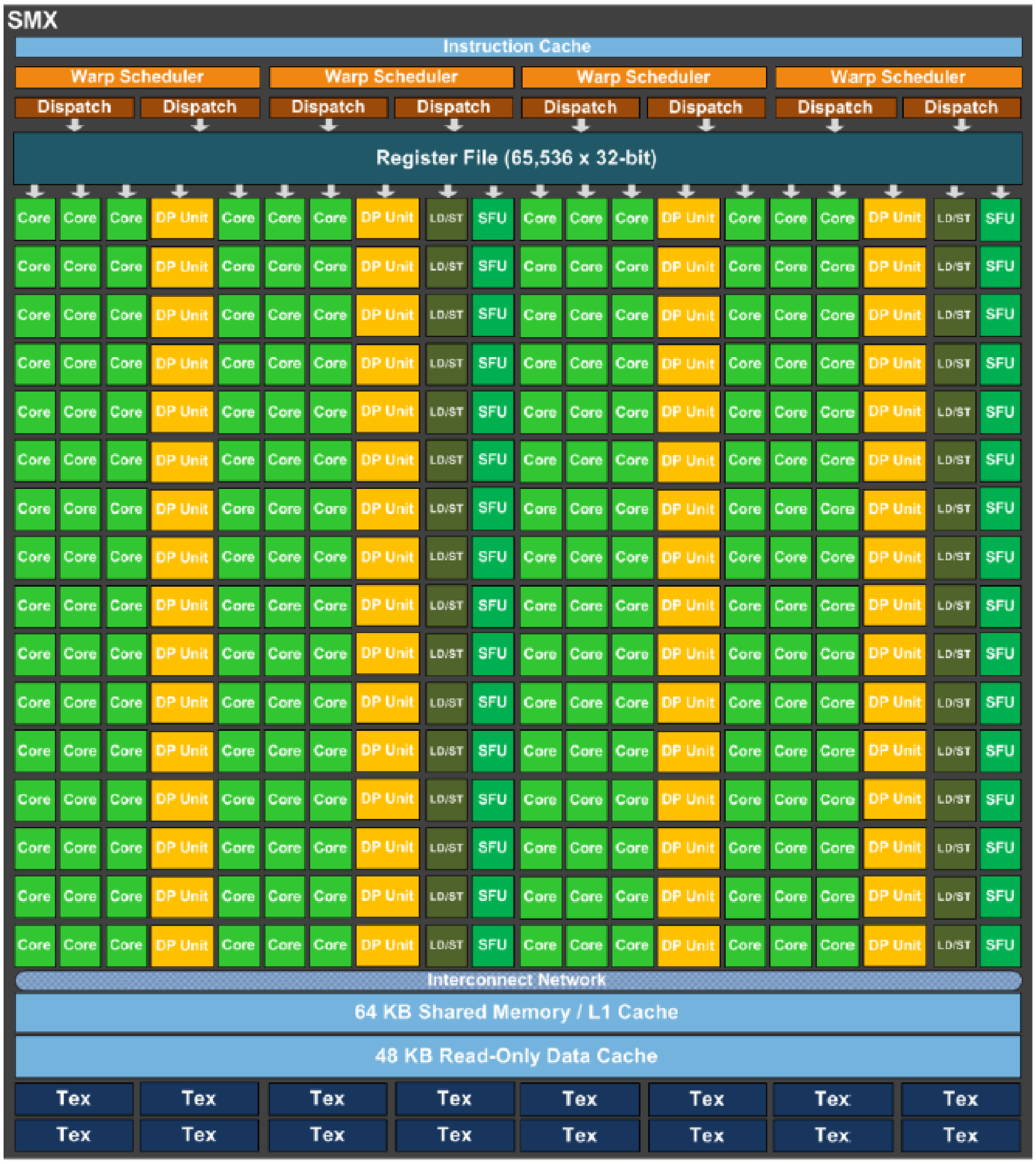


Figure 4.3: Organization of a streaming multiprocessor in a modern GPU [20]

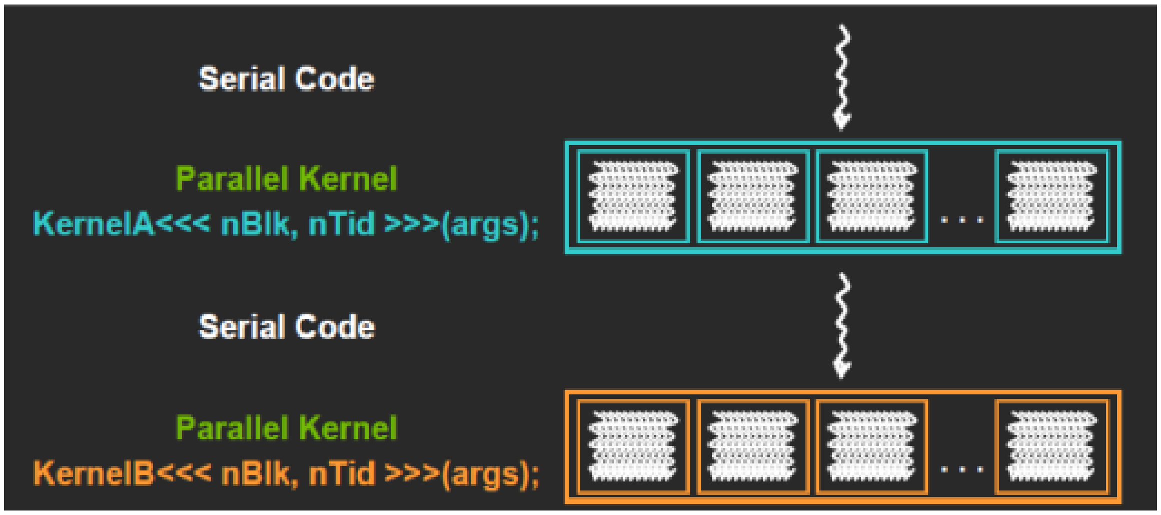

Serial Code
Parallel Kernel
KernelA<<< nBlk, nTid >>>(args);
Serial Code
Parallel Kernel
KernelB<<< nBlk, nTid >>>(args);

Figure 4.4: Heterogeneous serial-parallel programming model of CUDA [20]

software environment which offers both data parallelism and multi-threading. In this thesis we programmed the parallel implementations of the diffusion algorithms in CUDA version 6.0. CUDA offers parallel programming by extending familiar programming languages to parallel programming such as: CUDA C, CUDA Fortran, Python. The language we chose in this thesis is CUDA C. It is a version of the C language with extensions to make it more expressive in parallel programming. It is a heterogeneous serial-parallel programming model [20]. The serial code runs on a CPU thread which is called *host* and the parallel kernel code runs in thread blocks across multiple processing elements of the GPU which is called *device*. Both host and device code have their own separate memory space in DRAM. This programming model can be better visualized as in Figure 4.4. This kernel function is defined with a *_global_* declaration specifier and uses <<< >>> to specify the execution configurations when it is called.

4.2.1 Parallel Kernel Execution Model

The structure of the kernel execution model is summarized as follows [20].

- Data should be copied from the host to the device memory to make it accessible by each thread.
- The programmer has to launch the kernel by specifying the number of grids and thread blocks explicitly.
- The same kernel program is run to compute the value of each thread concurrently.
- The values computed are written to the device memory and they can be read in later kernel functions.

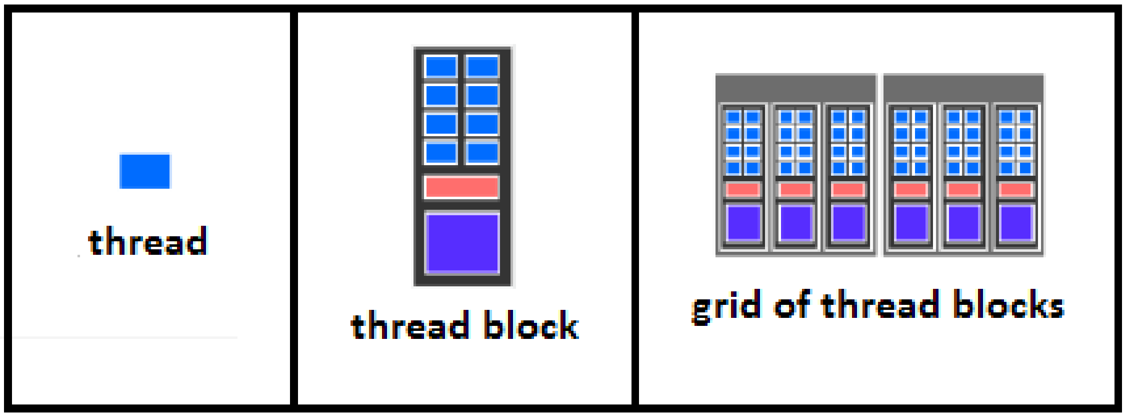


Figure 4.5: Thread, Block and Grid Organization in CUDA [20]

- Data should be copied back to the host memory, if it will be used later in a sequential code.

4.2.2 Thread-Block-Grid Hierarchy

Grids and thread blocks have to be arranged explicitly by the programmer, optimally in such a way that the performance gain is maximal. Figure 4.5 shows the thread, block and grid hierarchy in CUDA. One thread block can contain up to 1024 threads. Hence, if more than 1024 threads are required, an according number of grids have to be chosen such that the number of data points equals the number of grids times the number of threads chosen in a block [2].

Each thread in a block has a thread id, denoted as *threadIdx*, and each block has a block id, denoted as *blockIdx*. They can be accessed inside the kernel function [20]. This allows us to process different indices of arrays in each thread concurrently. Figure 4.6 shows how this addressing is done in 1-D. *BlockIdx.x* enables access to the corresponding block with dimension *BlockDim.x* in x direction. Hence any thread in this block can be accessed by the index $blockIdx.x * blockDim.x + threadIdx.x$. In our context it is used as a pixel index to compute each pixel value in parallel.

4.2.3 Thread Execution

Running threads in kernel functions in CUDA is asynchronous. If synchronization among the threads is required, $\_\_syncthreads()$ can be called in the kernel function to keep all threads in a block waiting for the last thread to finish [20].

4.2.4 Memory Hierarchy

In CUDA, each thread has its own private local memory and each block of threads has a fast shared memory. This shared memory can be thought of as a cache with low-latency, which is visible to all threads in that block [20]. Also there is a global

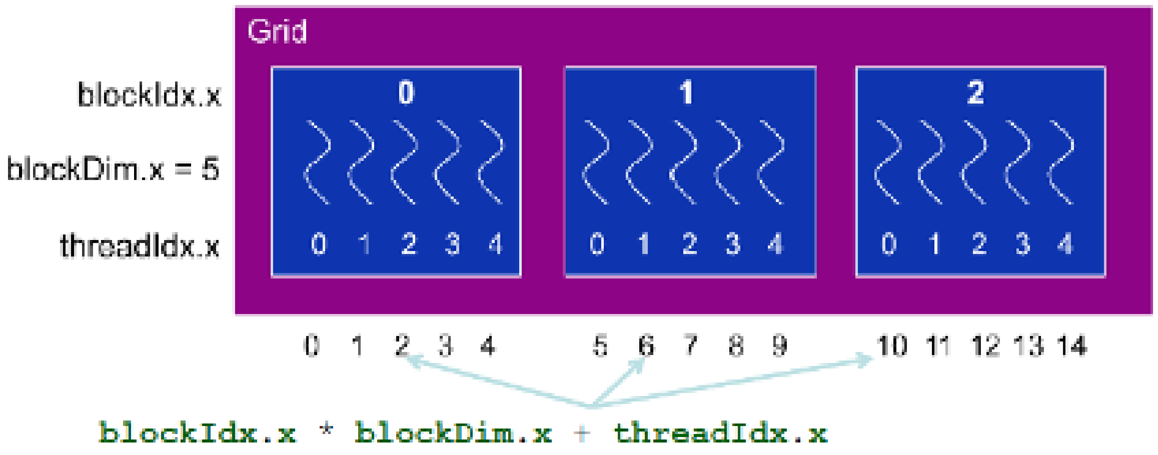


Figure 4.6: Thread addressing in CUDA [20]

memory to which all threads can reach. Figure 4.7 gives a detailed visualization of the memory hierarchy in CUDA. The arrows in the figure show which memory units are accessible to the host and/or the device code. CUDA also supports texture and constant memory for special data formats [2]. Below only the texture memory will be explained since it is a specialized memory for image-like formats.

4.2.5 Texture and Surface Binding

Data dependent applications can always benefit from clever memory types and alignments. Considering big data sizes, parallel programming can also benefit quite well. As on the CPU, reading chunks of the memory into the caches in advance can accelerate applications dramatically. By doing so, the *spatial locality* is exploited, i.e. if an element of an array is read, it is likely that an element in the neighbourhood will be read. Hence, the neighbouring elements are placed in the cache together with the requested element. However, for image-like data, 2-D locality is more appropriate. In order to exploit *dimensional locality* CUDA uses texture caches to improve the performance [25].

CUDA supports two data types that use the memory alignment mentioned above: textures and surfaces. While Textures are read-only, Surfaces provide both read and write operations [2]. There are 1-D, 2-D 3-D and 2-D layered textures and surfaces. 2-D layered texture or surface is a sequence of 2-D textures or surfaces. We experimented with 3-D and 2-D layered textures and surfaces in the Section 4.4.4, since they are appropriate for the 3-D data in our context.

Textures in CUDA use floating point representation for addressing. For example, for a 1-D array of size n, indices should be in the range $[0, n-1]$. It is obvious that in this range there can be subpixel locations. CUDA handles this with interpolation. It supports nearest neighbor and linear interpolation [2]. 2-D layered textures are

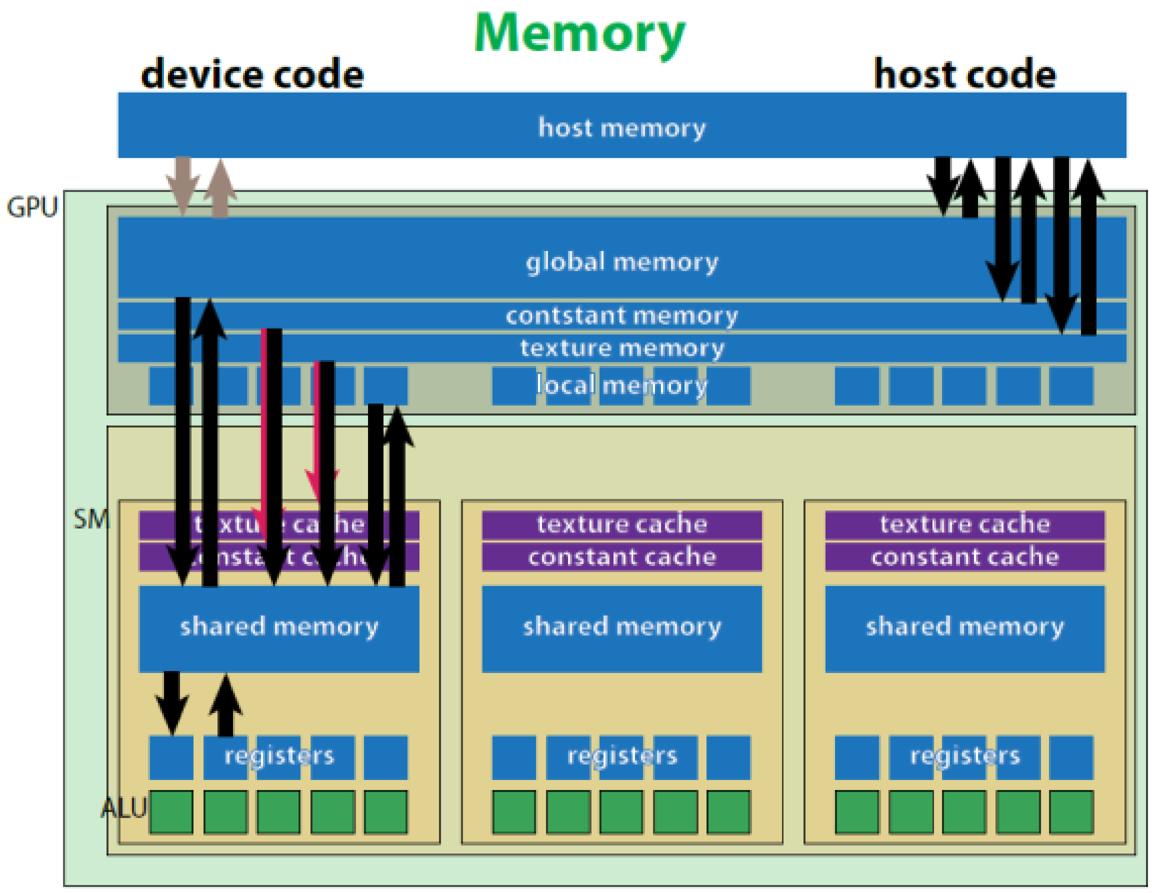


Figure 4.7: Memory hierarchy in CUDA [20]

slightly different. They use the same addressing within each layer but use integer indices for layer addressing. In contrast to textures, surfaces use byte addressing. For example, in order to access x^{th} row and y^{th} column in an integer valued image, one should use $(x * (sizeof(int)), y)$ as the index. 2-D layered surface adressing is similar to 2-D layered textures. It uses surface addressing within each layer; however, it uses integer indices for layer addressing [2]. CUDA also has a limitation for the dimensions of textures and surfaces. For 2-D layered textures and surfaces, the size in x, y, z-dimensions cannot exceed $16384, 16384, 2048$ respectively. For 3-D textures and surfaces these limits are $4096, 4096, 4096$ [1].

4.3 Parallel Computation of the Diffusion Echo on GPU

In the work of Dam and Nielsen [6], diffusion echoes are computed only for several pixels. It is easy to interpret the behaviour of the diffusion process from the echoes but only up to small stopping times. However, each pixel has a different echo and long diffusion times might be more interesting to see the general structures around the pixels. In this case, for an image with n pixels, there exist n echoes, which have to be computed until a desired stopping time. Additionally, considering that each echo has exactly the size of the original image, n images of size n need to be stored. This raises the computational and memory complexity to $O(n^2)$. However, parallel programming on the GPU mentioned above can reduce the computational time significantly.

Considering once more that the echoes show underlying image structure, we came up with an intuitive conjecture.

Conjecture 1: Pixels within one segment of an image have similar diffusion echoes as the diffusion time increases, i.e. they are correlated.

Therefore, storing all echoes would be redundant. The very first idea is to sample the image grid *uniformly* and to compute the echoes of sampled pixels simultaneously.

As described in Chapter 3, this requires to create an auxiliary image for each sampled pixel with the size of the original image as in Figure 3.2. After this initialization, the diffusion is implemented iteratively. At each iteration the weights $\boldsymbol{Q}$ to diffuse the original image are used to diffuse all images created. At the end of the iterations, the diffusion echoes of all sampled pixels have been computed up to a stopping time. This problem can be formulated as follows for $m \leq n$ sampled pixels:

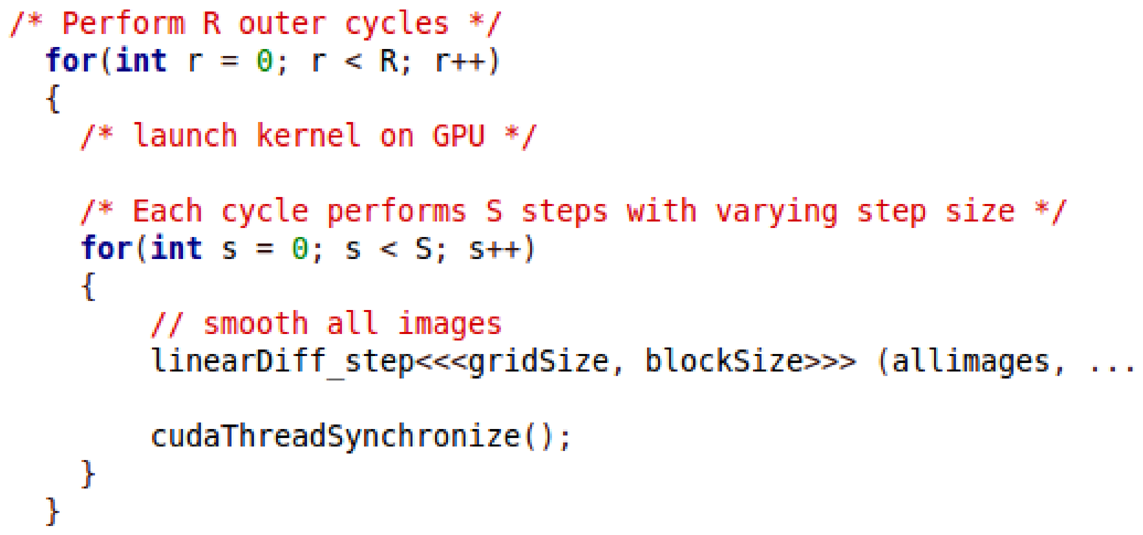
$$(\boldsymbol{u}^{k+1}, \boldsymbol{e}_1^{k+1}, \boldsymbol{e}_2^{k+1}, ..., \boldsymbol{e}_m^{k+1}) = \boldsymbol{Q}(\boldsymbol{u}^k) \cdot (\boldsymbol{u}^k, \boldsymbol{e}_1^k, \boldsymbol{e}_2^k, ..., \boldsymbol{e}_m^k), \tag{4.1}$$

$$\boldsymbol{u}^0 = \boldsymbol{f}, \tag{4.2}$$

where $\boldsymbol{f}$ is the original bounded image, $\boldsymbol{u}$ is the evolving image and $\boldsymbol{e}_i$s are diffusion echoes corresponding to the sampled pixels. This enables us to design parallel algorithms for the diffusion models as in the following subsections.

4.3.1 Parallel Linear Diffusion Algorithm

The parallel linear diffusion algorithm is the same as the corresponding sequential algorithm except that the function computing new values of the images at step $k+1$ is not a standard C subroutine. Instead, a kernel function is called to compute the values in parallel. In Figure 4.8 we can see the new algorithm using the FED scheme. In the inner cycle, S iterations of linear diffusion with varying time steps are iterated. The time steps are computed according to the Equation (2.32). After S iterations of the inner cycles, a box filter is approximated. Outer cycle iterates this box filter R times in order to smooth image with the linear diffusion. The kernel function is called with $<<< blocksize, gridsize >>>$ configuration arranged according to the 3-D data in hand. As explained in Section 4.2, for a sequence of 65 images of size 256×256, $gridsize$ in x and y directions is computed as $gridsize.x = \frac{256}{blocksize.x}$ and $gridsize.y = \frac{256}{blocksize.y}$, respectively, for any chosen $blocksize$. Similarly, in z direction, it is computed as $gridsize.z = \frac{65}{blocksize.z}$. If the result of the division is fractional,



```
/* Perform R outer cycles */
  for(int r = 0; r < R; r++)
  {
    /* launch kernel on GPU */

    /* Each cycle performs S steps with varying step size */
    for(int s = 0; s < S; s++)
    {
        // smooth all images
        linearDiff_step<<<gridSize, blockSize>>> (allimages, ...

        cudaThreadSynchronize();
    }
  }
```

Figure 4.8: Parallel linear diffusion algorithm

then it should be rounded up in order to comprise all the data points. How this kernel function looks like will be explained in the next section.

4.3.2 Parallel Isotropic Nonlinear Diffusion Algorithm

Similar to the linear diffusion case, the sequential algorithm to compute isotropic nonlinear diffusion can be used in the parallel algorithm. As in Figure 4.9, the FED scheme is adapted to the isotropic nonlinear scheme as follows: In the outer FED cycle, only the original image is presmoothed with linear diffusion. Then the diffusivities are computed depending on the smoothed image. As required in the FED scheme, explained in Section 2.3, the iteration matrix $\boldsymbol{A}(\boldsymbol{u}^k)$ must be kept constant during diffusion with varying time steps. That is why the diffusivity calculation is in the outer cycle. In the inner cycle, the diffusivities are used to diffuse all the images in the sequence (diffusion echo $\boldsymbol{e}$ or original image $\boldsymbol{u}$). While the presmoothing and diffusivity computation are on the evolving original 2-D image, the diffusion process is on the whole 3-D data. That is why a different block and grid configuration is required for kernels computing linear diffusion and diffusivity.

4.3.3 Parallel Edge-Enhancing Diffusion Algorithm

The parallel algorithm of edge-enhancing diffusion also follows from the sequential model. The algorithm is similar to the isotropic nonlinear diffusion model except that diffusivities are turned into diffusion tensor entries. This is to allow tuning the diffusion separately in the direction along the edge and the direction across the edge according to Equation (2.15). The algorithm can be seen in Figure 4.10.

```
/* Perform R outer FED cycles */
for(int r = 0; r < R; r++)
{
  /* launch kernel on GPU */

  // presmooth the original image
  linearDiff_step<<<gridSize2, blockSize2>>> (image1, ...

  cudaThreadSynchronize();

  // find diffusivities based on original image
  diffusivity<<<gridSize2, blockSize2>>> (image1, ...

  cudaThreadSynchronize();

  /* Each cycle performs S steps with varying step size */
  for(int s = 0; s < S; s++)
  {
      // diffuse all images with the diffusivities computed
      isononlinearDiff_step<<<gridSize, blockSize>>> (allimages, ...

      cudaThreadSynchronize();
  }
}
```

Figure 4.9: Parallel isotropic nonlinear diffusion algorithm

4.3.4 Parallel Coherence-Enhancing Diffusion Algorithm

This scheme is a modified version of the parallel edge-enhancing diffusion. As can be seen from Figure 4.11, firstly, the structure tensor entries are computed on the original image in the outer FED cycle. Then the structure tensor entries are smoothed by linear diffusion until a desired integration scale is reached. Depending on the structure tensor entries, the diffusion tensor entries are computed according to Equation (2.17). This allows tuning the diffusion separately in the direction along and across the coherent structures.

Before we move on to the design of the kernel functions, it is important to understand the problem at hand. One can see from Equation (4.1) that the computation of the time step $k + 1$ of each evolving image in the sequence is independent of each other. This, firstly, led us to an intuitive idea of computing each echo simultaneously. But when we see the big picture in the equation, we see once again that the diffusion of each image is computed with an explicit scheme. The explicit scheme is highly parallelizable as explained in Section 2.2. Therefore, this problem can be parallelized in different directions. We discuss these parallelizations in the following subsection.

```
/* Perform R outer cycles */
  for(int r = 0; r < R; r++)
  {
    /* launch kernel on GPU */

    // presmooth the original image
    linearDiff_step<<<gridSize2, blockSize2>>> (image1, ...

    cudaThreadSynchronize();

    // find diffusivities based on original image
    diffusion_tensor<<<gridSize2, blockSize2>>> (image1, ...

    cudaThreadSynchronize();

    /* Each cycle performs S steps with varying step size */
    for(int s = 0; s < S; s++)
    {
        // diffuse all images with the diffusivities computed
        edgeenhancingDiff_step<<<gridSize, blockSize>>> (allimages, ...

        cudaThreadSynchronize();
    }
  }
```

Figure 4.10: Parallel edge-enhancing diffusion algorithm

4.3.5 Types of Parallelizations

Parallel Computation of Diffusion Echoes

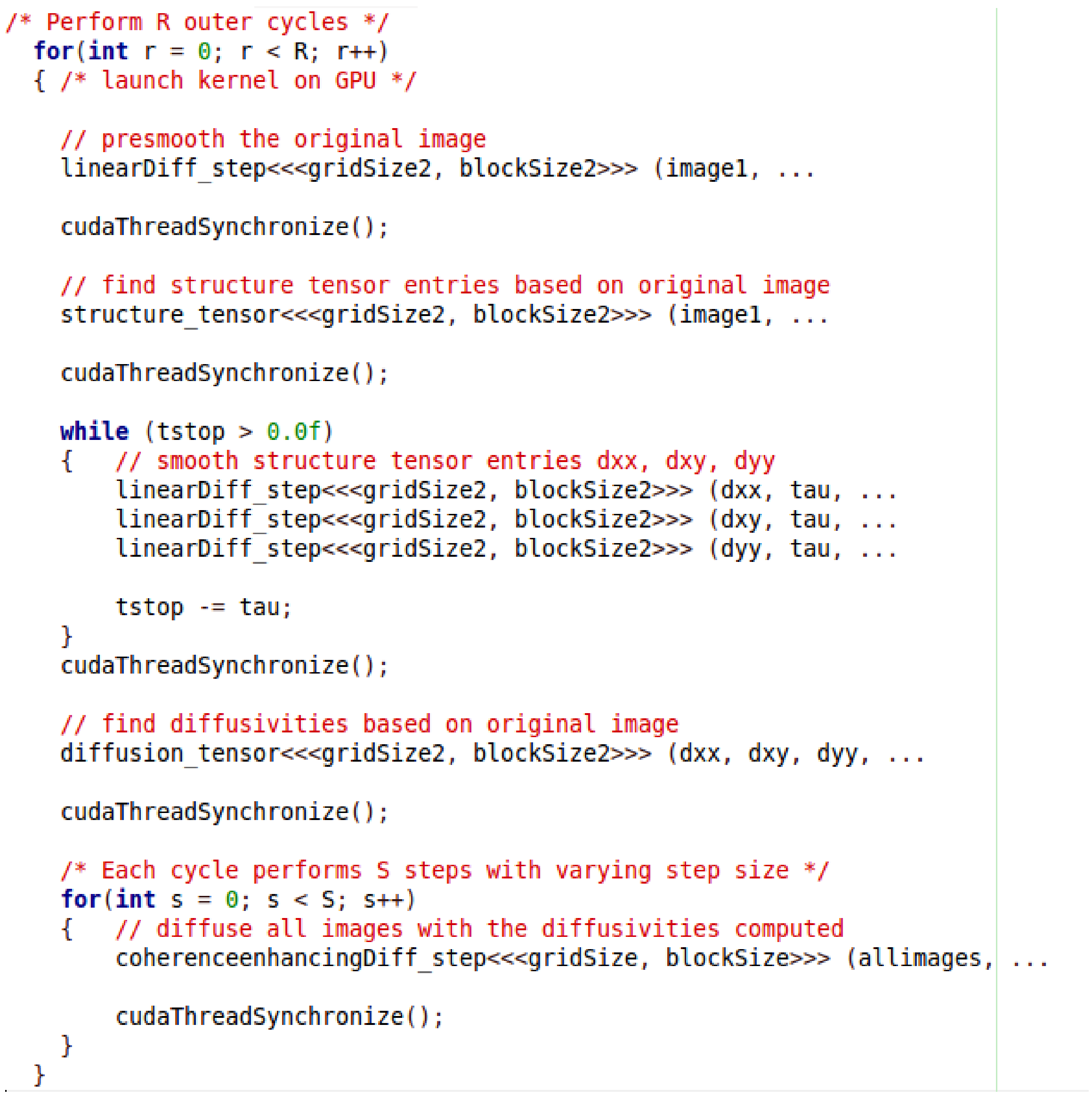
This parallelization is the initial intuitive idea: computing diffusion echoes simultaneously. Images in the sequence in Equation (4.1) are computed in a parallel way while the diffusion process within each image is computed sequentially. We can see how this kernel looks like in the code snippet in Figure 4.12.

Parallel Computation of Diffusion

Second one is the exact opposite of the first one. The diffusion process within each image is parallelized while the images in the sequence are visited sequentially. We see the corresponding code snippet in Figure 4.13.

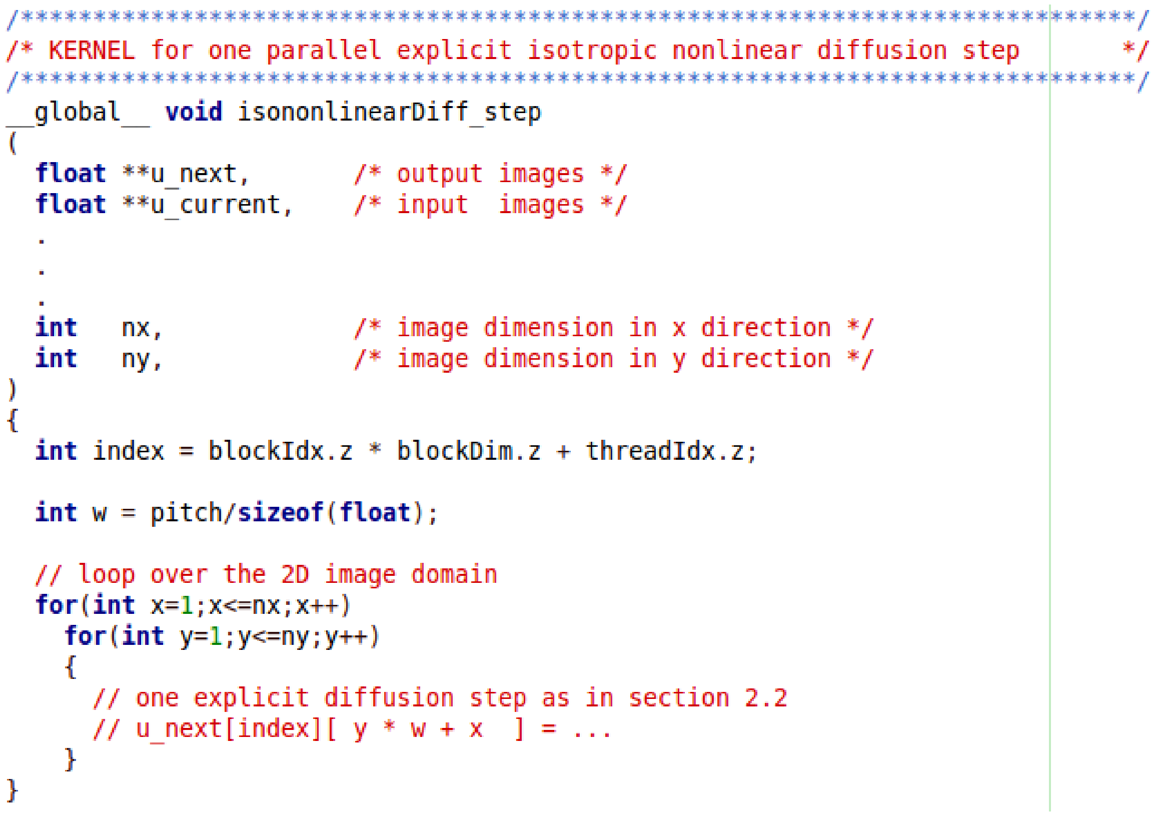
Full Parallelization

The beauty here is that both paralleizations are independent of each other. We see in Equation (4.1) that computation of each pixel is independent of not only all other pixels in the same image, but also all pixels in all other images. Therefore, these two parallelizations can be combined together into a full 3-D parallelization. Each pixel



```
/* Perform R outer cycles */
  for(int r = 0; r < R; r++)
  { /* launch kernel on GPU */

    // presmooth the original image
    linearDiff_step<<<gridSize2, blockSize2>>> (image1, ...

    cudaThreadSynchronize();

    // find structure tensor entries based on original image
    structure_tensor<<<gridSize2, blockSize2>>> (image1, ...

    cudaThreadSynchronize();

    while (tstop > 0.0f)
    {   // smooth structure tensor entries dxx, dxy, dyy
        linearDiff_step<<<gridSize2, blockSize2>>> (dxx, tau, ...
        linearDiff_step<<<gridSize2, blockSize2>>> (dxy, tau, ...
        linearDiff_step<<<gridSize2, blockSize2>>> (dyy, tau, ...

        tstop -= tau;
    }
    cudaThreadSynchronize();

    // find diffusivities based on original image
    diffusion_tensor<<<gridSize2, blockSize2>>> (dxx, dxy, dyy, ...

    cudaThreadSynchronize();

    /* Each cycle performs S steps with varying step size */
    for(int s = 0; s < S; s++)
    {   // diffuse all images with the diffusivities computed
        coherenceenhancingDiff_step<<<gridSize, blockSize>>> (allimages, ...

        cudaThreadSynchronize();
    }
  }
```

Figure 4.11: Parallel coherence-enhancing diffusion algorithm



```
/*****************************************************************************/
/* KERNEL for one parallel explicit isotropic nonlinear diffusion step      */
/*****************************************************************************/
__global__ void isononlinearDiff_step
(
  float **u_next,       /* output images */
  float **u_current,    /* input  images */
  .
  .
  .
  int   nx,             /* image dimension in x direction */
  int   ny,             /* image dimension in y direction */
)
{
  int index = blockIdx.z * blockDim.z + threadIdx.z;

  int w = pitch/sizeof(float);

  // loop over the 2D image domain
  for(int x=1;x<=nx;x++)
    for(int y=1;y<=ny;y++)
    {
      // one explicit diffusion step as in section 2.2
      // u_next[index][ y * w + x  ] = ...
    }
}
```

Figure 4.12: Parallel computation of diffusion echoes

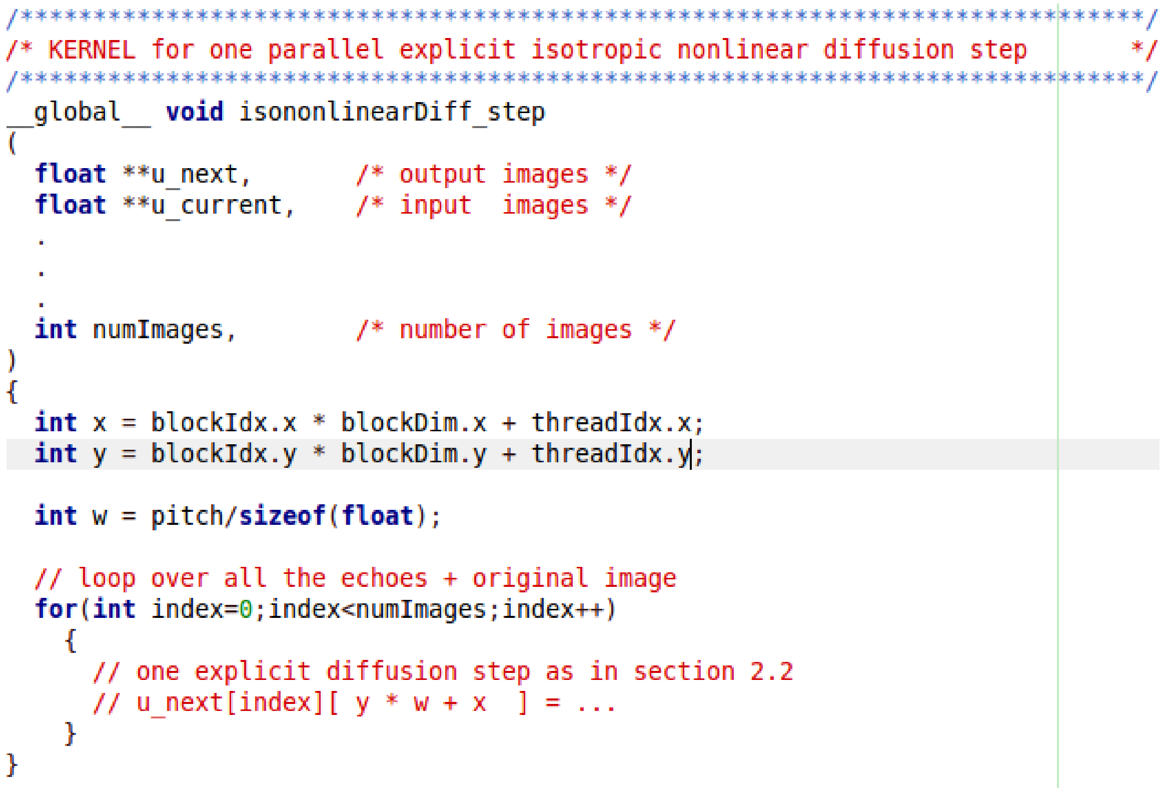
in any image in the sequence can be computed in parallel. We can see how this kernel function looks like in Figure 4.14.

4.4 Experiments

In this section, first, we experimented to see the effect of different types of parallelization. Then we focused on the effect of grid and block sizes on the performance. Moreover, we observed further speed up with texture and surface bindings. All the parallel implementations of the diffusion algorithms in this thesis are run on NVIDIA GeForce GTX 480 GPU with the specifications: 448 CUDA cores, 607 MHz. graphics clock, 1215 MHz. processor clock, and 133.9 (GB/sec) memory bandwidth [3].

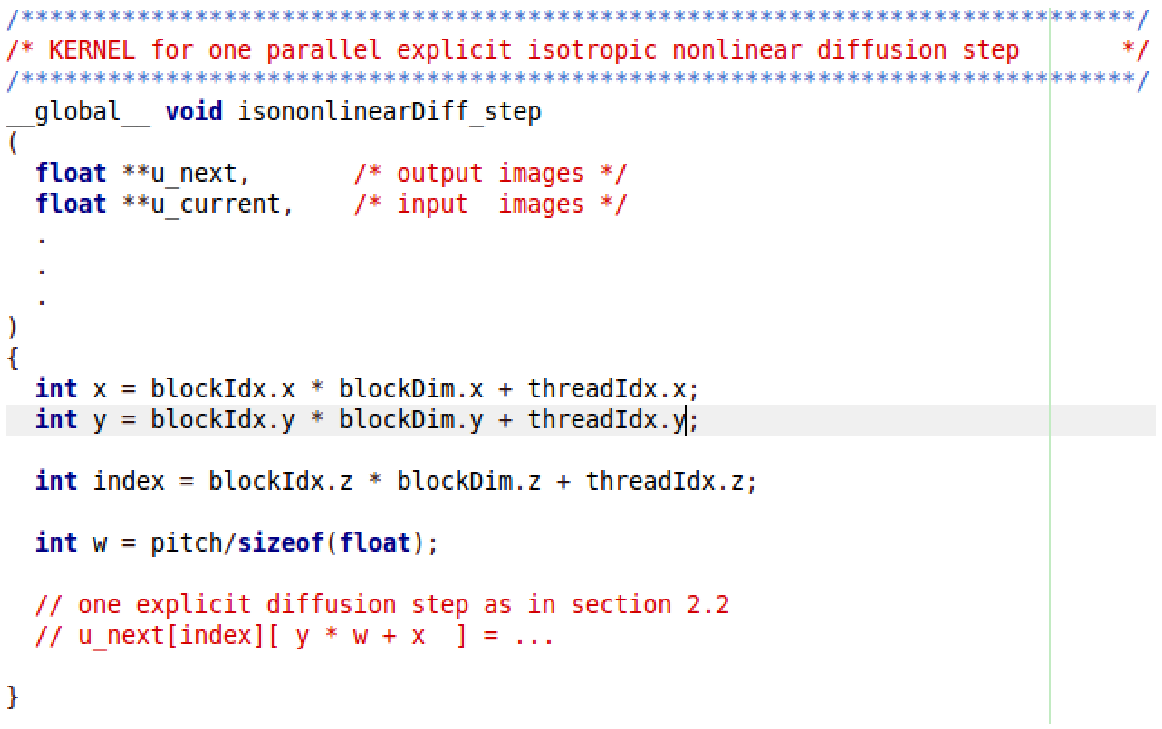
4.4.1 Effect of Different Parallelizations

In Table 4.1, we see the comparison of running times for the three types of parallelizations and the sequential implementation. The running times are acquired for the 256×256 head image (Figure 1.1a) and 64 uniformly sampled echoes. The isotropic nonlinear diffusion with Weickert's diffusivity is chosen since it is the slowest diffusion model among others due to the slow decay of the diffusivity function. Therefore, one needs large diffusion times compared to other models.



```
/****************************************************************************/
/* KERNEL for one parallel explicit isotropic nonlinear diffusion step     */
/****************************************************************************/
__global__ void isononlinearDiff_step
(
  float **u_next,       /* output images */
  float **u_current,    /* input  images */
  .
  .
  .
  int numImages,        /* number of images */
)
{
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int y = blockIdx.y * blockDim.y + threadIdx.y;

  int w = pitch/sizeof(float);

  // loop over all the echoes + original image
  for(int index=0;index<numImages;index++)
    {
      // one explicit diffusion step as in section 2.2
      // u_next[index][ y * w + x  ] = ...
    }
}
```

Figure 4.13: Parallel computation of diffusion



```
/****************************************************************************/
/* KERNEL for one parallel explicit isotropic nonlinear diffusion step     */
/****************************************************************************/
__global__ void isononlinearDiff_step
(
  float **u_next,       /* output images */
  float **u_current,    /* input  images */
  .
  .
  .
)
{
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int y = blockIdx.y * blockDim.y + threadIdx.y;

  int index = blockIdx.z * blockDim.z + threadIdx.z;

  int w = pitch/sizeof(float);

  // one explicit diffusion step as in section 2.2
  // u_next[index][ y * w + x  ] = ...

}
```

Figure 4.14: Full parallelization

Diffusion Time vs. Parallelization	**No Parallelization**	**Parallel Computation of Diffusion Echoes**	**Parallel Computation of Diffusion**	**Full Parallelization**
T=10	5900ms	1401676ms	3078ms	87ms
T=100	27330ms	-	12294ms	342ms
T=1000	150260ms	-	36885ms	1022ms
T=10000	997720ms	-	118309ms	3272ms
T=100000	1275310ms	-	376471ms	10410ms

Table 4.1: Running times for different type of parallelizations of isotropic nonlinear diffusion with Weickert's diffusivity, $\lambda = 3$, $\tau = 0.24$, $\sigma = 0.5$ and number of FED cycles= 25

It is interesting to see that the initial idea of computing echoes simultaneously fails despite how intuitive it was. Parallelization of the diffusion echoes runs even slower than the sequential code. However, this is not surprising. Because as can be seen from the Figure 4.12, looping over the whole image domain in a parallel kernel is nothing but running a sequential code on GPU. One thread running on one core of a GPU is less efficient than running on a CPU. This type of parallelization is not tested for further stopping times as marked in the table with "-". The second idea in the third column already offers remarkable speed ups. It has only one loop and the loop is limited to $m+1$ images in the sequence, which is much smaller than the number of pixels n in the image. In the third column we see that the full parallelization offers the most tremendous speed up. For stopping time 10 already a speed up by 67 is gained. Fully parallel code can compute 67 echoes while sequential code diffuses one image. Also as the diffusion time T increases, performance gain increase. From this part on, full parallelization is used in all the experiments.

4.4.2 Effect of Block Size

As described in Section 4.2, the programmer explicitly defines the block and grid sizes. They can be tuned according to the data size in the application in order to achieve higher performance gains. One should remember the limit of 1024 threads per block. Hence, multiplication of all three dimensions of the block size cannot exceed 1024. As an additional requirement, the z-dimension of a block can not exceed 64. These and more specifications can be found in [1]. In Table 4.2, we see the effect of different block sizes. These block sizes are tried for both $blocksize$ and $blocksize2$ variables mentioned in Subsection 4.3.2. The cases that exceed the limitation mentioned are marked as "-". In the experiment the same parameters as in the previous experiment are used with a stopping time of 100000 for better comparison.

Best performance is achieved with the combination $16 \times 16 \times 1$ as can be seen in bold. In each row, running times decrease until the optimized combination of block sizes in z-direction is reached. Specifically, we see a rapid decay from $z = 1$ to $z = 2$.

Block Size $(x \times y)$ **vs.** (z)	**1**	**2**	**4**	**8**	**16**	**32**	**64**
2×2	136350ms	74630ms	44302ms	35723ms	33235ms	36501ms	51001ms
4×4	39703ms	22754ms	15284ms	16102ms	21169ms	26061ms	45017ms
8×8	12686ms	11762ms	13256ms	15176ms	21884ms	-	-
16×16	**10410ms**	12889ms	15198ms	-	-	-	-
32×32	14267ms	-	-	-	-	-	-

Table 4.2: Effect of block size

Block Size $(x{\times}y{\times}z)$	$16{\times}16{\times}1$	$16{\times}16{\times}2$	$16{\times}16{\times}4$	$8 \times 8 \times 8$	$8{\times}8{\times}16$	$4{\times}4{\times}32$	$4{\times}4{\times}64$
run time	10410ms	12871ms	15195ms	15164ms	21862ms	26013ms	44917ms

Table 4.3: Effect of block size for *blocksize*2 when *blocksize* $= 16 \times 16 \times 1$

However, after the optimised point, increasing the size makes the performance worse. This is expected, since 2-D locality within each image in the sequence is prior to the locality in third dimension (Equation 4.1) in our problem. Same behaviour is also seen in each column. However, it is surprising that the block size $32 \times 32 \times 1$ does not provide with better performance.

Since two block size variables control 2 different configurations, they should also be analysed independently. It is obvious that increasing the size in z-direction of *blocksize*2 would not result in any performance gain. To observe this, another experiment is done by keeping *blocksize* constant to $16 \times 16 \times 1$ while changing *blocksize*2. Table 4.3 shows the running times for different *blocksize*2.z. x and y-directions are arranged so that they are maximum within the limitations. We see that this does not provide us with any gain, as we expected.

Intuitively, experimenting the other way around, i.e. increasing the z-direction of the *blocksize* while keeping *blocksize*2 constant to $16 \times 16 \times 1$, could have resulted in better performance. However as can be seen in Table 4.4, it nearly has no effect.
In the rest of the experiments block size $16 \times 16 \times 1$ will be used for both variables.

Block Size $(x{\times}y{\times}z)$	$16{\times}16{\times}1$	$16{\times}16{\times}2$	$16{\times}16{\times}4$	$8 \times 8 \times 8$	$8{\times}8{\times}16$	$4{\times}4{\times}32$	$4{\times}4{\times}64$
run time	10410ms	10412ms	10414ms	10417ms	10434ms	10453ms	10510ms

Table 4.4: Effect of block size for *blocksize* when *blocksize*2 $= 16 \times 16 \times 1$

Diffusion Time	**Run Time of IND**	**Run Time of EED**	**Run Time of CED**
T=10	68ms	109ms	134ms
T=100	204ms	326ms	353ms
T=1000	663ms	1058ms	1102ms
T=10000	2075ms	3307ms	3401ms
T=100000	6589ms	10490ms	10929ms

Table 4.5: Running times for isotropic nonlinear diffusion (IND), edge-enhancing diffusion (EED) and coherence-enhancing diffusion (CED) with parameters: $\tau_{stable} = 0.24$, $\sigma = 0.5$, and number of FED cycles= 10

4.4.3 Different Diffusion Models

In Table 4.5, the running times for the isotropic nonlinear, edge-enhancing and coherence-enhancing diffusions are shown for a 256×256 image with same number of FED cycles. The reason that the edge-enhancing diffusion runs slower is the computation of the additional diffusivity entries. Similarly, the reason that the coherence-enhancing diffusion is running slightly slower than all the others is that there are additional computations for the structure tensor and its smoothing.

4.4.4 Further Speed Up

CUDA offers further speed ups with the use of texture and surface bindings as mentioned in Section 4.2. This type of memory alignment is well suitable for our implementations. In our setting, the original image and m echoes can be treated as 3-D or 2-D layered data. In order to bind 3-D texture, $m + 1$ images can be seen as one 3-D image. However, by the nature of our work, 2-D layered texture seems more appropriate. The diffusion computation for each pixel in this 3-D image is computed exactly the same; however, each pixel value only depends on the neighboring pixels within each image. Hence, each image can be seen as one of the layers in 2-D layered texture or surface.

We tested both 3-D and 2-D layered texture and surface bindings. In Table 4.6, we see the comparison between naive parallel implementation, 3-D and 2-D layered texture and surface bindings. Experiments are run with the isotropic nonlinear diffusion with Weickert's diffusivity on a 256×256 image with different number of diffusion echo sample sizes. For a better comparison with the previous experiments, we used stopping time T=100000. We see a slight speed up for all the sample sizes for texture and surface bindings. However, the differences between 3-D and 2-D layered versions are not remarkable. For the sample sizes 64 and 256, 2-D layered texture and surface bindings runs slightly faster, while for the sample size 1024, 3-D version runs faster.

If required, the samples with more diffusion echoes can be computed manually by running the code according number of times. For example, the sample with all dif-

Sample Size	**Naive Implementation**	**3-D Texture/Surface Binding**	**2-D Layered Texture/Surface Binding**
64	10410ms	9703ms	9484ms
256	40822ms	39313ms	39158ms
1024	162850ms	157984ms	159041s

Table 4.6: Effect of Texture and Surface binding

fusion echoes of a 256×256 image, i.e. 65536 diffusion echoes, can be computed by running the code that can compute 1024 diffusion echoes 64 times.

4.5 Discussion

By exploiting parallelism on GPUs, tremendeous speed ups have been obtained. Still this is not the limit. Multiple GPUs can be used to linearly speed up each experiment. Also with the high performance acceleration in GPUs shown in Figure 4.1, more speed ups will be provided for the very same implementations in the future.

Furthermore, echoes can be computed for all kinds of filters. By iterating the auxiliary image together with the filter weights of the original image, one can obtain impulse response of the filter at any pixel location in an efficient way. This filter echo provides better understanding of the filter as it does to the diffusion filters.

One important point which is also worth discussing is the sampling. In the whole work of this thesis, we used uniform sampling on the image grid. Adaptive sampling remains as future work. With clever sampling methods, we can compute the echoes corresponding to individual segments.

Another method to analyse important segments can be Principal Component Analysis. This is the next part of this thesis. We will explain it in detail in the following chapter.

Chapter 5

Compact Representation of the Diffusion Echo

In this chapter, we will explain Principle Component Analysis in detail to form a basis for understanding how it can be implemented on diffusion echoes. Moreover, we will present an efficient way to compute the Principle Component Analysis. Then, we will explain how Nystroem approximation can be applied to diffusion echoes. In the following section, we will show experiments with wide range of images, diffusion schemes and parameters. Discussion will end this chapter.

5.1 Principal Component Analysis

Principal Component Analysis (PCA) is one of the most important statistical data analysis methods. It is widely used in various fields in image processing and computer vision such as pattern recognition and knowledge-based medical image analysis. Once the underlying dynamics of the PCA are well-studied, it can be a powerful data analysis tool. It is also parameter-free. Therefore, compact analysis of any data is obtained without any user interaction.

The data obtained in scientific experiments today is complex and high-dimensional. Since the underlying structure of the system which is to be discovered are not known before the experiments, the measurements are unclear and even redundant [19]. Also one has to take into consideration that the data can be noisy depending on the data acquisition methods. However, what an experimenter wants is to extract the quantities and the structures that represents the data in a compact way i.e. to find the real parameters defining the scientific phenomena being experimented. This is the actual goal of the PCA. It takes unclear and noisy data as input and outputs the principal basis that represents this complex data the best. Hence, it can also be used as a dimensionality reduction tool by eliminating redundant components [19].

In principle, the PCA finds another basis to represent the data which is a *linear* combination of the original basis. The word *linear* makes this process a change of basis. Let the original data be represented by an $n \times m$ matrix $\boldsymbol{X}$, which has n

measurements of a single experiment as rows and m trials of the same experiment as columns and with the PCA, let a new representation of $\boldsymbol{X}$ be denoted by another $n \times m$ matrix $\boldsymbol{Y}$. The change of basis process of the PCA can be expressed as

$$\boldsymbol{PX} = \boldsymbol{Y}. \tag{5.1}$$

The $n \times n$ matrix $\boldsymbol{P}$ which transforms $\boldsymbol{X}$ into $\boldsymbol{Y}$ contains the principle orthonormal basis vectors for $\boldsymbol{X}$ in each row $\boldsymbol{p}_i$. Therefore, the crucial part here is the choice of $\boldsymbol{P}$. It should be such a transformation matrix that $\boldsymbol{Y}$ will have minimum redundancy [19]. Minimum redundancy means that any of the measurements can not be expressed in a linear combination of any other. In other words any two rows of $\boldsymbol{Y}$, $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$, are uncorrelated.

In order to reach this goal, it is important to introduce a *covariance* measure. Covariance shows the correlation between two variables. For two row-vectors $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$, having zero-mean and size m, it is defined as

$$cov(\boldsymbol{x_1}, \boldsymbol{x_2}) = \frac{1}{m}\boldsymbol{x_1}\boldsymbol{x_2}^T. \tag{5.2}$$

A positive value for the covariance means that the variables are positively correlated i.e. they change together, while a negative value means they are negatively correlated i.e. they change inversely. A zero covariance means that they are uncorrelated. Since the data matrix $\boldsymbol{X}$ has more than 2 dimensions to be analysed, a covariance measure generalized to higher dimensions is necessary. *Covariance matrix* of the high dimensional data matrix $\boldsymbol{X}$ is defined as

$$\boldsymbol{C_X} := \frac{1}{m}\boldsymbol{X}\boldsymbol{X}^T \tag{5.3}$$

Diagonal entries of this $n \times n$ symmetric matrix are the variances of each measurement defined as $var(\boldsymbol{x}_i) = cov(\boldsymbol{x}_i, \boldsymbol{x}_i)$, while the off-diagonal entries are the covariances of any two different measurements. With the PCA, $\boldsymbol{C_X}$ is transformed into $\boldsymbol{C_Y}$, the covariance matrix of $\boldsymbol{Y}$. The goal of this transformation is that $\boldsymbol{C_Y}$ will have zero off-diagonals so that $\boldsymbol{y}_i$s are uncorrelated. This suggests that this transformation is a diagonalization [19]. In order to reach this diagonalization $\boldsymbol{C_Y}$ can be rewritten as

$$\begin{aligned}
\boldsymbol{C_Y} &= \frac{1}{m}\boldsymbol{Y}\boldsymbol{Y}^T \\
&= \frac{1}{m}(\boldsymbol{PX})(\boldsymbol{PX})^T \\
&= \frac{1}{m}\boldsymbol{P}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{P}^T \\
&= \boldsymbol{P}(\frac{1}{m}\boldsymbol{X}\boldsymbol{X}^T)\boldsymbol{P}^T \\
\boldsymbol{C_Y} &= \boldsymbol{P}\boldsymbol{C_X}\boldsymbol{P}^T.
\end{aligned}$$

For a symmetric matrix $\boldsymbol{C_X}$ there exists an eigenvalue decomposition as $\boldsymbol{C_X} = \boldsymbol{EDE}^T$, where $\boldsymbol{D}$ is an $n \times n$ diagonal matrix with eigenvalue λ_i of $\boldsymbol{C_X}$ in i^{th} diagonal entry and $\boldsymbol{E}$ has the corresponding eigenvector as i^{th} column for $i = 1, 2, ..., n$. Furthermore, since we want each row of $\boldsymbol{P}$ to be orthogonal eigenvectors of $\boldsymbol{C_X}$, we can substitute $\boldsymbol{P} = \boldsymbol{E}^T$. This yields

$$\begin{aligned}
\boldsymbol{C_Y} &= \boldsymbol{P C_X P}^T \\
&= \boldsymbol{P}(\boldsymbol{E}^T \boldsymbol{DE})\boldsymbol{P}^T \\
&= \boldsymbol{P}(\boldsymbol{P}^T \boldsymbol{DP})\boldsymbol{P}^T \\
&= (\boldsymbol{PP}^T)\boldsymbol{D}(\boldsymbol{PP}^T) \\
&= (\boldsymbol{PP}^{-1})\boldsymbol{D}(\boldsymbol{PP}^{-1}) \\
\boldsymbol{C_Y} &= \boldsymbol{D}.
\end{aligned}$$

Therefore, the eigenvalue decomposition of $\boldsymbol{C_X}$ provides the diagonalisation desired. With this result, we can conclude that the eigenvectors, $\boldsymbol{v}_i$s, of $\boldsymbol{C_X}$ are the principle components of $\boldsymbol{X}$ and the corresponding eigenvalues, λ_is, are the variances of $\boldsymbol{X}$ along each $\boldsymbol{p}_i$ [19]. One should normalize the eigenvectors after the computations in order to have orthonormal basis.

5.2 PCA of the Diffusion Echo

In Chapter 4, we explained how diffusion echoes of any diffusion model can be computed in an efficient way. In 160 seconds, it is possible to compute 1024 echoes diffused until the stopping time of 100000 (Table 4.6). Moreover, with an appropriate choice of diffusivity function, we can obtain segment-like echoes. This is helpful to analyse important structures. However, storing 1024 diffusion echoes for one image is not practical. First of all, it requires $1024 \times 256 \times 256$ bytes for a sample of 1024 diffusion echoes of a 256×256 image. Secondly, due to the uniform sampling, for each individual structure in the image, there exist many similar diffusion echoes. Finally, with increasing sampling rates, there will be more similar diffusion echoes of the same segment which are more redundant. Therefore, a compact representation of the important structures with less images is desired.

As described in Section 5.1, the PCA on the diffusion echoes can provide with the important structures as *eigenechoes* and their correspondent importance measure as eigenvalues. This leads us to our second conjecture.

Conjecture 2: Eigenechoes corresponding to the largest eigenvalues will represent the individual segments of the image.

When each diffusion echo is treated as an experiment on the image structures, each pixel of the diffusion echo can be interpreted as an individual measure in the experiment. Therefore, each column of the data matrix $\boldsymbol{X}$ corresponds to one diffusion echo. This can be achieved by concatenating each column of the 2-D diffusion echo into a 1-D column vector of size $n \times 1$, where n is the number of pixels in the image. Furthermore, the mean of each diffusion echo is subtacted in order to obtain zero-mean 1-D diffusion echoes as column vectors. Then the sampled m diffusion echoes can be represented as $\boldsymbol{f_1}, \boldsymbol{f_2}, ..., \boldsymbol{f_m} \in \mathbb{R}^n$, where m corresponds to the number of experiments and each $\boldsymbol{f}_i$ has zero-mean. In general $m << n$. Moreover, deviations from the average image are in the center of our interest, so the average image $\bar{\boldsymbol{f}}$ as in Equation (5.4) should be subtracted from each $\boldsymbol{f}_i$ [24].

$$\bar{\boldsymbol{f}} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{f}_i \tag{5.4}$$

Then the data matrix becomes $\boldsymbol{X} = [\boldsymbol{f}_1 - \bar{\boldsymbol{f}}, \boldsymbol{f}_2 - \bar{\boldsymbol{f}}, ..., \boldsymbol{f}_m - \bar{\boldsymbol{f}}]$ with covariance matrix

$$\boldsymbol{C_X} := \frac{1}{m} \boldsymbol{X} \boldsymbol{X}^T \tag{5.5}$$

having eigenvalues $\lambda_1 \geq \lambda_2 \geq ...\lambda_m > 0$ and the corresponding eigenechoes $\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_m$.

Since PCA is a change of basis, the original diffusion echoes can also be fully reconstructed as

$$\boldsymbol{f_i} = \bar{\boldsymbol{f}} + \sum_{j=1}^{m} < (\boldsymbol{f_i} - \bar{\boldsymbol{f}}), \boldsymbol{v}_j > \boldsymbol{v}_j, \tag{5.6}$$

where $<,>$ represents the dot product.

The eigenechoes corresponding to the largest eigenvalues represent the direction with the most variation in the image, i.e. important structures. Therefore, using only k eigenechoes corresponding to the k largest eigenvalues can be used to approximate the diffusion echoes [24] as

$$\boldsymbol{f_i} \approx \bar{\boldsymbol{f}} + \sum_{j=1}^{k} < (\boldsymbol{f_i} - \bar{\boldsymbol{f}}), \boldsymbol{v}_j > \boldsymbol{v}_j, \tag{5.7}$$

A good number for k can be determined by analysing the eigenvalues in decreasing order. If the decay is rapid enough, then k eigenvectors corresponding to the first k eigenvalues which are significantly larger than 0 can be chosen. This offers a compact representation with a subspace consisting of $\bar{\boldsymbol{f}}$ and k eigenechoes. After the computation of the PCA, eigenechoes should be reshaped from the 1-D column vector to a 2-D image as inverse to the concatenation performed in the beginning.

5.3 Efficient Computation of the PCA

Even though the PCA offers a compact representation, computation of it is cumbersome for images. For an image of size 256×256, $\boldsymbol{C_X}$ has size 65536×65536 which is even large for storing. Therefore, we need more efficient algorithms to compute eigenvalue decomposition. The trick for the efficient computation is to find the eigenvalue decomposition of the $m \times m$ matrix

$$\boldsymbol{T} = \frac{1}{m}\boldsymbol{X}^T\boldsymbol{X} \tag{5.8}$$

instead of the $n \times n$ matrix $\boldsymbol{C_X}$ by exploiting the connections between $\boldsymbol{T}$ and $\boldsymbol{C_X}$ [24]:

- The m eigenvalues of $\boldsymbol{T}$ are eigenvalues of $\boldsymbol{C_X}$.
- $\boldsymbol{T}$ contains all non-vanishing eigenvalues of $\boldsymbol{C_X}$.
- The remaining $(n - m)$ eigenvalues are zero.
- If $\boldsymbol{w}_i$ is an eigenvector of $\boldsymbol{T}$, then $\boldsymbol{v}_i = \boldsymbol{X}\boldsymbol{w}_i$ is an eigenvector of $\boldsymbol{C_X}$.

This algorithm computes eigenvectors of a much smaller $m \times m$ matrix instead of an $n \times n$ matrix. It offers remarkable speed ups since $m << n$ in general. One should remember to normalize eigenvectors after computations.

5.4 The Power Iteration for the Dominant Eigenvectors

As mentioned in the previous sections, we need the eigenvalues and the corresponding eigenvectors of the matrix $\boldsymbol{T}$ in order to do PCA on diffusion echoes in an efficient way. One way of finding the most dominant eigenvector with the corresponding eigenvalue in an iterative way is the power iteration method. In order to extend it to the k most dominant eigenvectors and eigenvalues, we can use the method of deflation [24]. The whole algorithm then reads:

```
for $i = 0, 1, 2, ..., k$ do
    initialize $\boldsymbol{w}_i^0 \in \mathbb{R}^m$ with a random normalized vector
    for $j = 0, 1, 2, 3, ...$ do
        compute $\boldsymbol{w}_i^{j+1} = \boldsymbol{T}\boldsymbol{w}_i^j$.
        compute $\lambda_i^{j+1} = |\boldsymbol{w}_i^{j+1}|$.
        normalize $\boldsymbol{w}_i^{j+1} = \boldsymbol{w}_i^{j+1}/\lambda_i^{j+1}$.
    end for
    update $\boldsymbol{T} \leftarrow \boldsymbol{T} - \lambda_i \boldsymbol{w}_i \boldsymbol{w}_i^T$
end for
```

The power iteration method guarantees the convergence of λ_i^j to the i^{th} dominant eigenvalue and the convergence of $\boldsymbol{w}_i^j$ to the corresponding i^{th} dominant eigenvector.

5.5 Nystroem Approximation

The Nystroem approximation is a method for estimating the eigenvectors of a symmetric similarity matrix

$$\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Pi}\boldsymbol{\Phi}^T, \tag{5.9}$$

where $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, ..., \boldsymbol{\phi}_n]$ has orthonormal eigenvectors of $\boldsymbol{K}$ in its columns and $\boldsymbol{\Pi}$ is a diagonal matrix with corresponding eigenvalues, $\pi_1, \pi_2, ..., \pi_n$, in its diagonal entries [14]. In order to compute these eigenvectors, all the entries of $\boldsymbol{K}$ are required. However, Nystroem suggests an approximation to these eigenvectors from the sampled entries of $\boldsymbol{K}$ [21].

When m pixels are sampled from an image with n pixels, an $m \times m$ similarity matrix $\boldsymbol{K_A}$ for the image $\boldsymbol{A}$, which consists these sampled pixels, can be obtained. Similarly, $\boldsymbol{K_B}$ can be obtained for the image $\boldsymbol{B}$, which consists $(n-m)$ non-sampled pixels, as well as the $m \times (n-m)$ $\boldsymbol{K_{AB}}$, which contains the similarity measures between sampled and non-sampled pixels. Then $\boldsymbol{K}$ can be permuted into the form:

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K_A} & \boldsymbol{K_{AB}} \\ \boldsymbol{K_{AB}^T} & \boldsymbol{K_B} \end{bmatrix} \tag{5.10}$$

The first m eigenvectors of $\boldsymbol{K}$ can be approximated as

$$\tilde{\boldsymbol{\Phi}} = \begin{bmatrix} \boldsymbol{\Phi_A} \\ \boldsymbol{K_{AB}^T}\boldsymbol{\Phi_A}\boldsymbol{\Pi_A^{-1}} \end{bmatrix}, \tag{5.11}$$

where the first m entries of $\tilde{\boldsymbol{\Phi}}$ are the eigenvectors, $\boldsymbol{\Phi_A}$, of $\boldsymbol{K_A}$ which are computed as $\boldsymbol{K_A} = \boldsymbol{\Phi_A}\boldsymbol{\Pi_A}\boldsymbol{\Phi_A}^T$ [21]. The remaining $(n-m)$ entries are approximated as a weighted projection of $\boldsymbol{K_{AB}}$ on $\boldsymbol{\Phi_A}$ with weights $\boldsymbol{\Pi_A}^{-1}$. Then the similarity matrix $\boldsymbol{K}$ can be approximated as

$$\begin{aligned} \tilde{\boldsymbol{K}} &= \tilde{\boldsymbol{\Phi}}\boldsymbol{\Pi_A}\tilde{\boldsymbol{\Phi}}^{\boldsymbol{T}} \\ &= \begin{bmatrix} \boldsymbol{\Phi_A} \\ \boldsymbol{K_{AB}^T}\boldsymbol{\Phi_A}\boldsymbol{\Pi_A^{-1}} \end{bmatrix} \boldsymbol{\Pi_A} \left[\boldsymbol{\Phi_A^T}\boldsymbol{\Pi_A^{-1}}\boldsymbol{\Phi_A^T}\boldsymbol{K_{AB}}\right] \\ &= \begin{bmatrix} \boldsymbol{K_A} & \boldsymbol{K_{AB}} \\ \boldsymbol{K_{AB}^T} & \boldsymbol{K_{AB}^T}\boldsymbol{K_A^{-1}}\boldsymbol{K_{AB}} \end{bmatrix} \end{aligned}$$

5.5.1 Nystroem Approximation for Image Denoising

Talebi and Milanfar [21] use the Nystroem approximation in denoising for efficiency concerns. Since filters for methods like BM3D and NLM are based on a similarity measure between pixels in an image, an $n \times n$ filter needs to be computed for an image having n pixels. Their algorithm to approximate the whole filter is as follows:

- Pre-smooth the image.
- Uniformly sample m pixels from the image.
- Find filters for the sampled m pixels.
- Permute the filters to form $\boldsymbol{K_A}$ and $\boldsymbol{K_{AB}}$.
- Use Nystroem approximation to estimate the eigenvectors of the whole filter.
- Sort them in decreasing order according to their corresponding eigenvalues.
- Reverse the permutation to recover the original locations.
- Shrink the eigenvalues for denoising.

5.5.2 Nystroem Approximation for the Diffusion Echoes

The concurrent work [21] by Talebi and Milanfar is very similar to our work. We also start with uniform sampling as they do. Although we follow a different method, we try to reach the same goal: finding eigenvectors and eigenvalues of filters. In our context, each diffusion echo of a sampled pixel is nothing but an affinity measure based on a diffusion process. It contains diffusion weights between the sampled pixel and both non-sampled and sampled pixels. Hence, we can also use Nystroem approximation to estimate eigenechoes as follows:

- Permute all sampled diffusion echoes having zero-mean into the matrices $\boldsymbol{K_A}$ and $\boldsymbol{K_{AB}}$.
- Use the Nystroem approximation to estimate the eigenechoes of all the diffusion echoes.
- Sort them in decreasing order according to their corresponding eigenvalues.
- Reverse the permutation to recover the original locations.

5.6 Experiments

In this section, we present the results of the PCA on the diffusion echoes computed in Chapter 4. First, we analyse the effect of diffusion time on the eigenechoes. Second, we present the eigenechoes for different diffusion models. Then, we experiment how Nystroem approximation performs on the diffusion echoes. Finally, we analyse the effect of sample size. Due to the normalization of the eigenechoes at the end of PCA, the eigenecho values are in the range of [-1, 1]. Therefore, all the eigenechoes presented are rescaled to the range of [0, 255] by preserving average value 0 as 127.5. We implemented the PCA in C as explained in Sections 5.2, 5.3, and 5.4. In the experiments, we iterated power method 10000 times in order to have a good convergence. We adapted the implementation of Nystroem approximation from the MATLAB code provided by Milanfar and Talebi in their website [4]. We run all the implementations on an Intel Core i5 CPU.

5.6.1 Effect of Diffusion Time

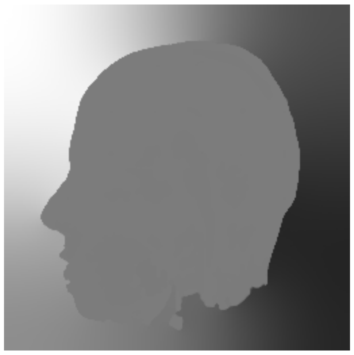
Our aim in PCA on diffusion echoes is being able to analyse important image structures. Therefore, parameter selection is playing a significant role. Parameters should be tuned such that diffusion echoes are good representatives of the segments they belong to. All the parameters except the diffusion time can be selected as in the denoising settings of diffusion models. However, diffusion time should be chosen carefully to make the pixel spread around and get the best segment representation as the diffusion model allows.

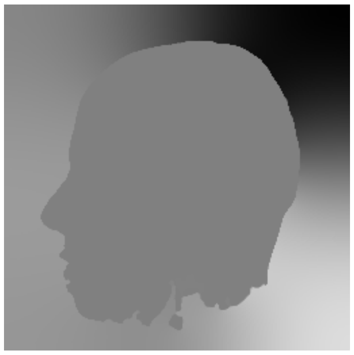
In order to see the importance of the diffusion time, 64 diffusion echoes of the 256×256 head image is diffused with the isotropic nonlinear diffusion with Weickert's diffusivity until a small diffusion time of $T = 1000$. Then, the PCA is used to obtain 10 eigenechoes corresponding to the 10 largest eigenvalues. As we see in Figure 5.1, the eigenechoes show important structures of the head image such as the skull and small segments to which the pixel spreads its value around within the diffusion time. However, most of the segments are not complete. They suffer from the incomplete diffusion echoes due to the small diffusion time.

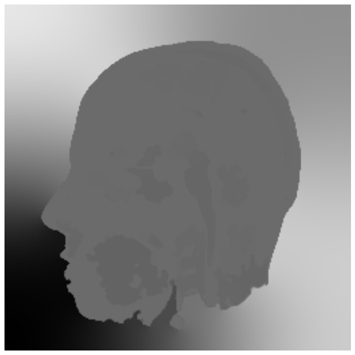
In Figure 5.2, we repeated the same experiment with a larger diffusion time of T=250000. We see the effect of diffusing through the segment boundaries due to the large diffusion time clearly in the eigenechoes 1, 9 and 10.

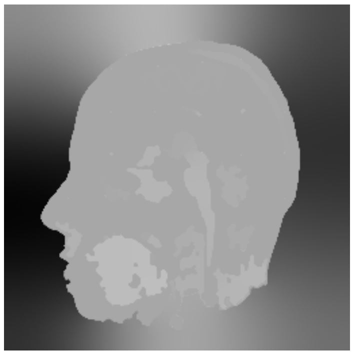
5.6.2 Different Diffusion Models

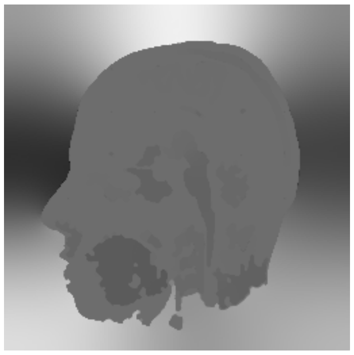
In this subsection, we show the resulting first 10 eigenechoes of 64 diffusion echoes of different diffusion models together with their eigenvalue plots in decreasing order. The diffusion time for each diffusion model is found experimentally.

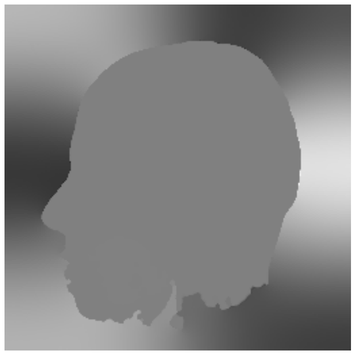

(a) eigenecho 1

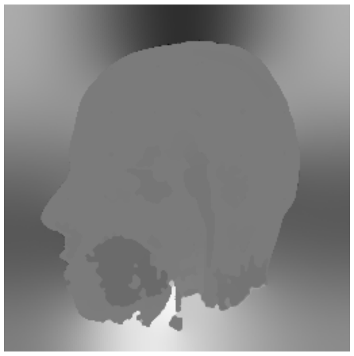

(b) eigenecho 2

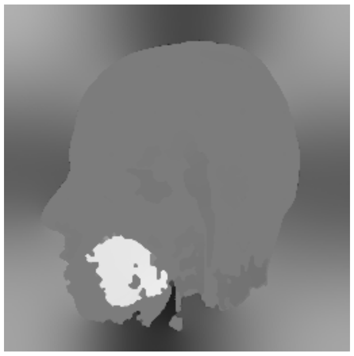

(c) eigenecho 3

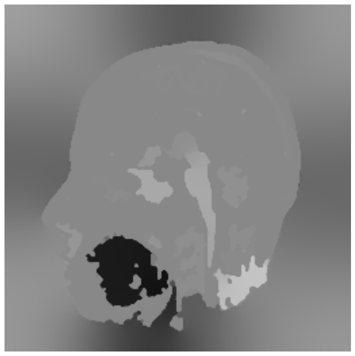

(d) eigenecho 4


(e) eigenecho 5


(f) eigenecho 6


(g) eigenecho 7


(h) eigenecho 8


(i) eigenecho 9

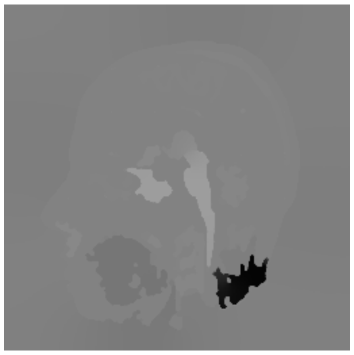
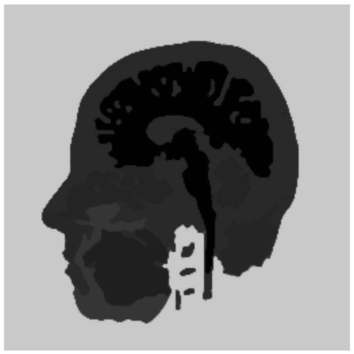
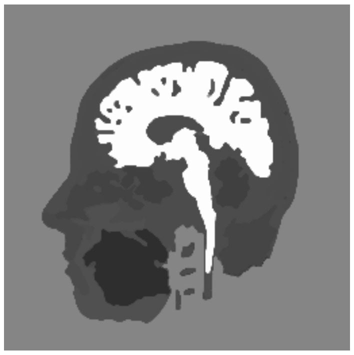
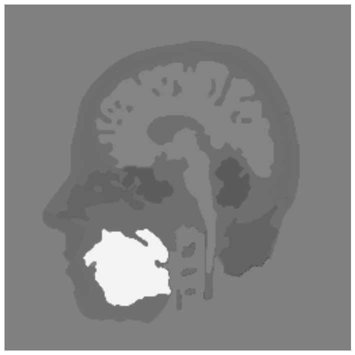
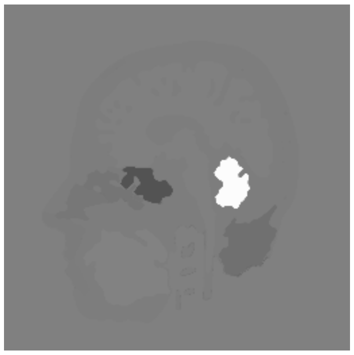

(j) eigenecho 10

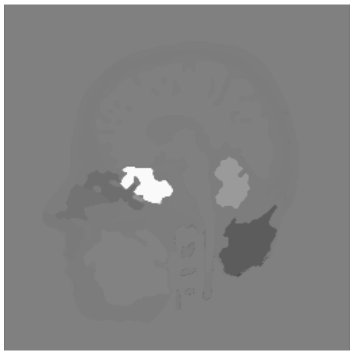
Figure 5.1: Eigenechoes for isotropic nonlinear diffusion with Weickert's diffusivity with $\lambda = 3$, stopping time T=1000, $\tau_{stable} = 0.24$, $\sigma = 0.5$, FED cycles=25, sample size=64, and running time=2.3 sec

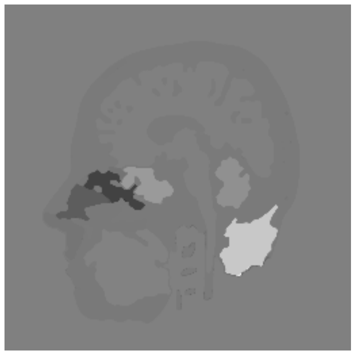

(a) eigenecho 1

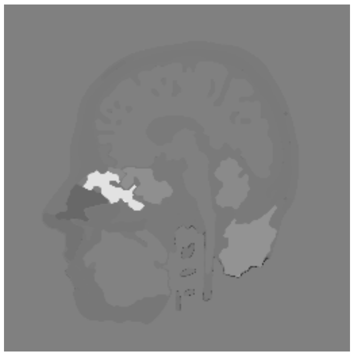

(b) eigenecho 2

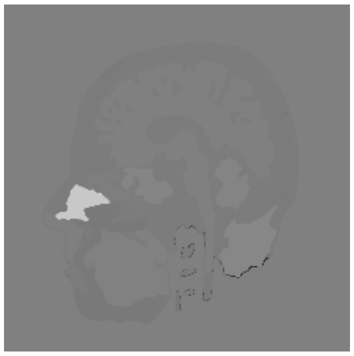

(c) eigenecho 3

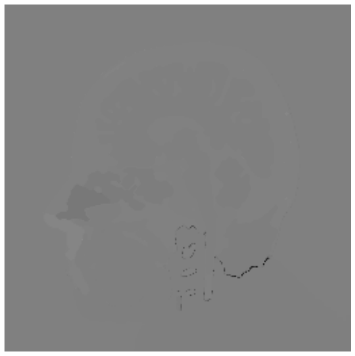

(d) eigenecho 4


(e) eigenecho 5


(f) eigenecho 6


(g) eigenecho 7


(h) eigenecho 8


(i) eigenecho 9

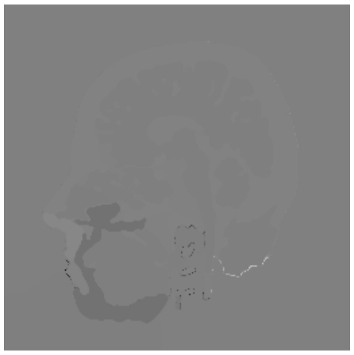

(j) eigenecho 10

Figure 5.2: Eigenechoes for isotropic nonlinear diffusion with Weickert's diffusivity with $\lambda = 3$, stopping time T=250000, $\tau_{stable} = 0.24$, $\sigma = 0.5$, FED cycles=25, sample size=64, and running time=2.4 sec

Linear Diffusion

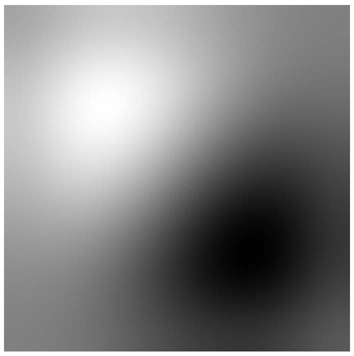
In Figure 5.3, we see the first 12 eigenechoes of the linear diffusion with diffusion time of T=500. Since Gaussians are not localized at the origin, it is difficult to know what to expect as a result. However, the PCA performs quite well. Especially the fourth eigenecho gives a very good representation of all Gaussians. It is also interesting that the eigenechoes look like the basis of the discrete cosine transform as a result of eigenvalue decomposition of the Laplacian operator. Moreover, we see the logarithmically scaled plot of all the non-zero eigenvalues in decreasing order in Figure 5.4. The linear-like decay in the plot suggests that the PCA works well on this sample. Therefore, it reveals the 12 most important structures of the sample in these first 12 eigenechoes.

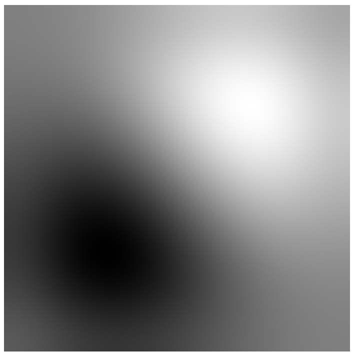
Isotropic Nonlinear Diffusion

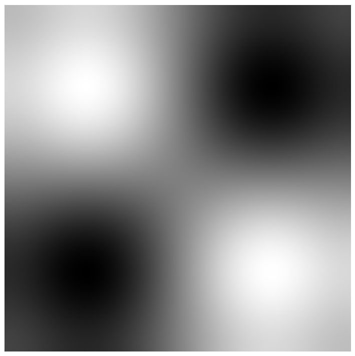
In Figure 5.5, we show the first 10 eigenechoes resulting from the PCA on the diffusion echoes of the same head image diffused with the isotropic nonlinear diffusion with Perona-Malik diffusivity. The eigenvalue plot in Figure 5.6 shows the good linear decay for this experiment. Each eigenecho highlights a coarse representation of a segment. We see that eigenechoes suffer from non-sharp segments as the diffusion echoes due to the slow decay of the diffusivity function. Weickert's diffusivity can address this problem. In Figure 5.7, we show the eigenechoes of the head image diffused with the isotropic nonlinear diffusion with Weickert's diffusivity. We see that the eigenechoes are not representing individual segments as we expected; however, they are quite representative and interesting. This can also be seen well with the rapid decay in the eigenvalue plot in Figure 5.8. These 10 eigenechoes are good representatives of the whole sample. In the first eigenecho, the most fundamental and frequent segment, skull, is found. In the second eigenecho, brain is the most visible shape as the second biggest and frequent segment in the image. Towards the tenth eigenecho, this ordering is not as ideal as the first 6. This might be caused by under-sampling of the diffusion echoes so that the smaller segments are not represented well enough in a sample of 64. Experiments in Section 5.6.4 will investigate this hypothesis.

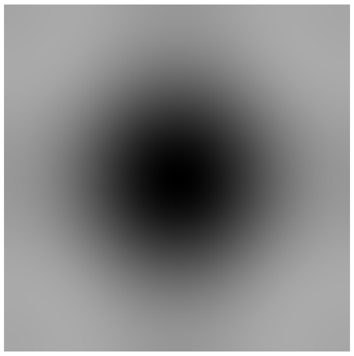
Edge-Enhancing Diffusion

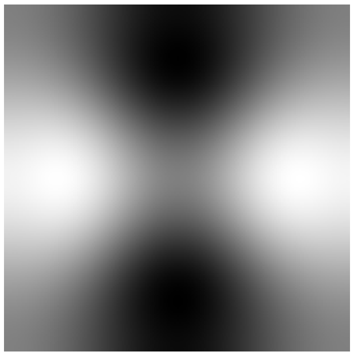
Figure 5.9 shows the first 12 eigenechoes of the diffusion echo sample obtained for the edge-enhancing diffusion on 128×128 pruebab image. They represent the main structures of the image, i.e. the triangle and the rectangle, quite well, especially, in the fifth eigenecho. We see the corresponding logarithmically scaled eigenvalue plot in Figure 5.10. The rapid decay in logarithmic scale suggests that using the first 20 eigenechoes we can approximate the whole sample quite well.

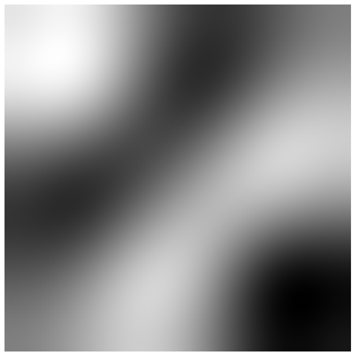

(a) eigenecho 1

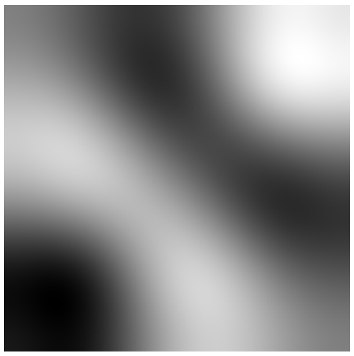

(b) eigenecho 2

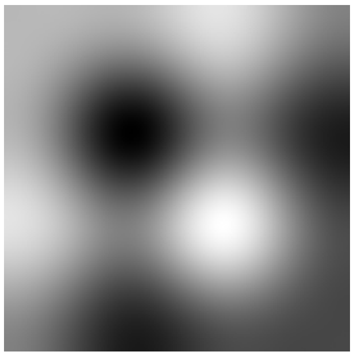

(c) eigenecho 3

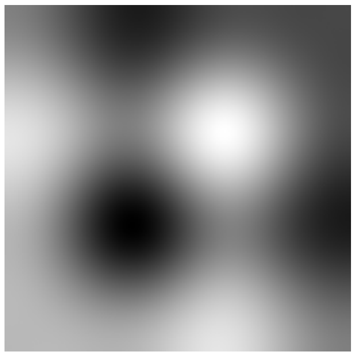

(d) eigenecho 4

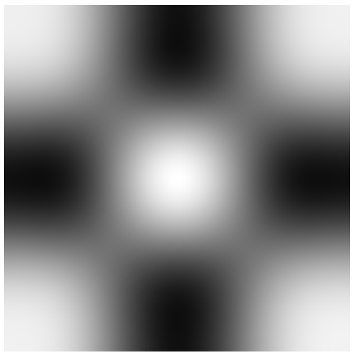

(e) eigenecho 5

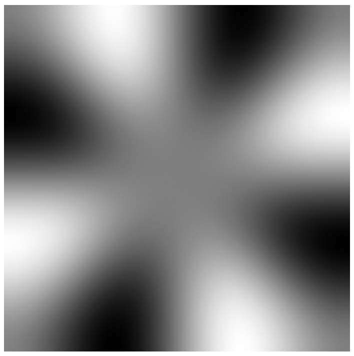

(f) eigenecho 6


(g) eigenecho 7


(h) eigenecho 8


(i) eigenecho 9


(j) eigenecho 10


(k) eigenecho 11

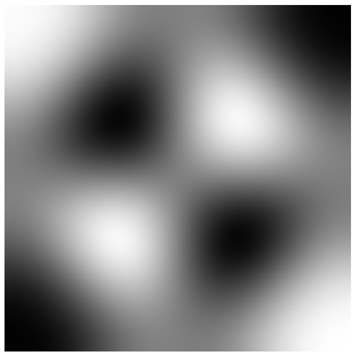

(l) eigenecho 12

Figure 5.3: Eigenechoes for linear diffusion with stopping time T=500, $\tau_{stable} = 0.25$, FED cycles=5, and running time=3.0 sec

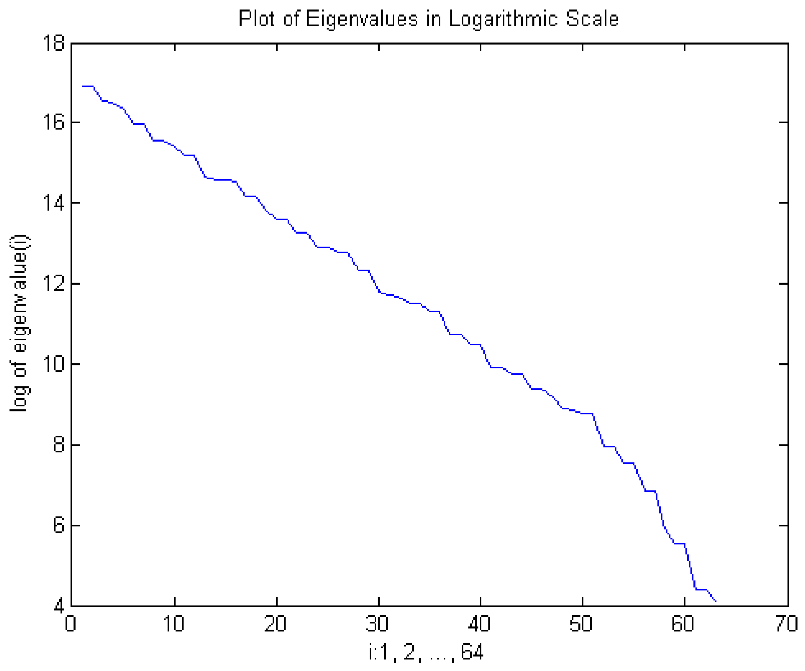


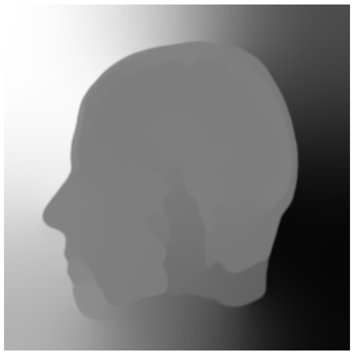
Figure 5.4: Logarithmically scaled plot of the eigenvalues of corresponding eigenechoes of experiment shown in Figure 5.3 in decreasing order

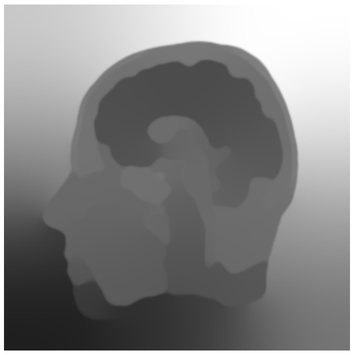
Coherence-Enhancing Diffusion

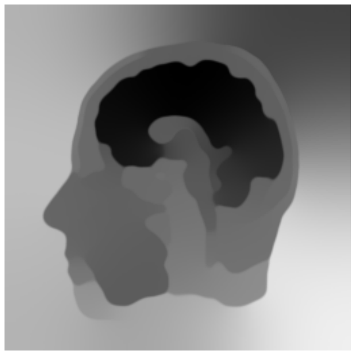
Although coherence-enhancing diffusion is not a good candidate for segment-like diffusion echoes, it raises curiosity about whether PCA can analyse small and distributed coherent structures resulting from this model. In order to analyse this, we ran PCA on the coherence-enhancing diffusion echoes of the finger image. We see in Figure 5.11 that PCA combines all small coherent structures into a silhouette of the image. This is especially visible in the forth eigenecho. However, the eigenechoes are not representative enough for the whole sample. Corresponding eigenvalue plot in Figure 5.12 shows how badly PCA works for this sample.

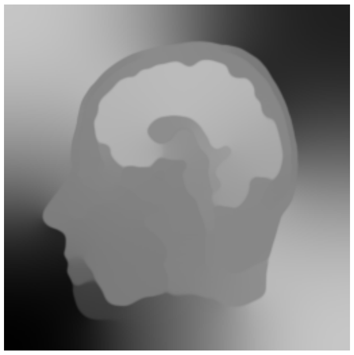
5.6.3 Nystroem Approximation

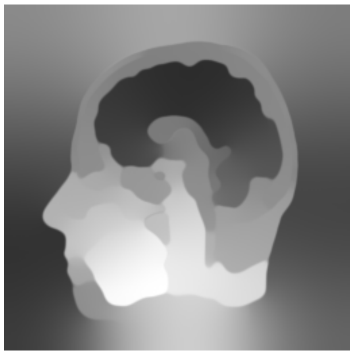
The Nystroem approximation counterparts are tested for each PCA experiment. The results of both methods result in similar representations of the image structures except at some pixel locations. Figure 5.13 shows the Nystroem counterpart of the experiment seen in Figure 5.7. We see that the problematic pixels are coinciding with the sampled pixels. The reason for these sampled points being not well approximated by Nystroem approximation is that it tries to approximate the whole eigenechoes only with these sampled points. This works well for the similarity based methods like BM3D and NLM as the work of Milanfar and Talebi [21] shows. These similarity measures are more global then diffusion that they work the same for any two pixels in the image as well as for two sampled points. However, this is not the case in diffusion.

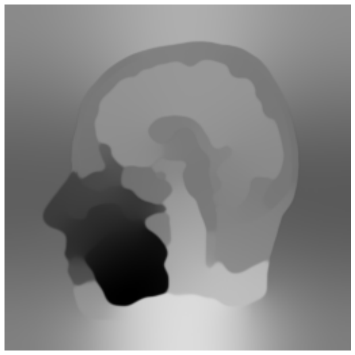

(a) eigenecho 1

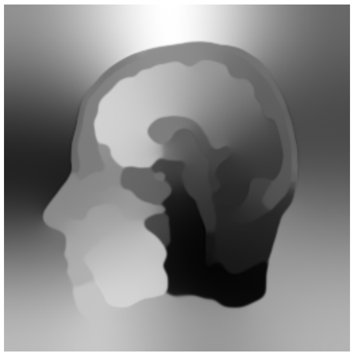

(b) eigenecho 2

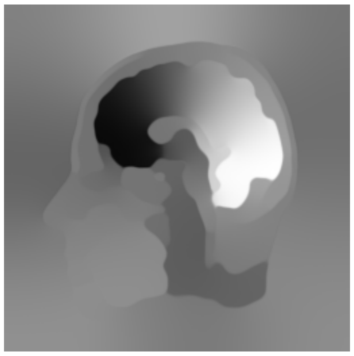

(c) eigenecho 3

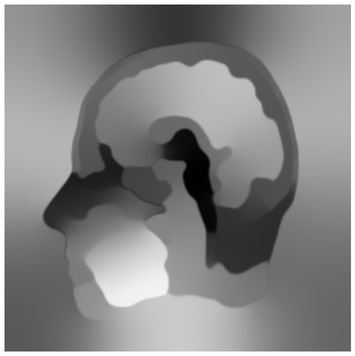

(d) eigenecho 4


(e) eigenecho 5


(f) eigenecho 6


(g) eigenecho 7


(h) eigenecho 8


(i) eigenecho 9

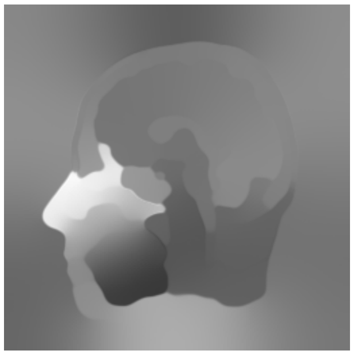

(j) eigenecho 10

Figure 5.5: Eigenechoes for isotropic nonlinear diffusion with Perona-Malik diffusivity with $\lambda = 1$, stopping time T=3000, $\tau_{stable} = 0.24$, $\sigma = 0.5$, FED cycles=10, and running time=2.6 sec

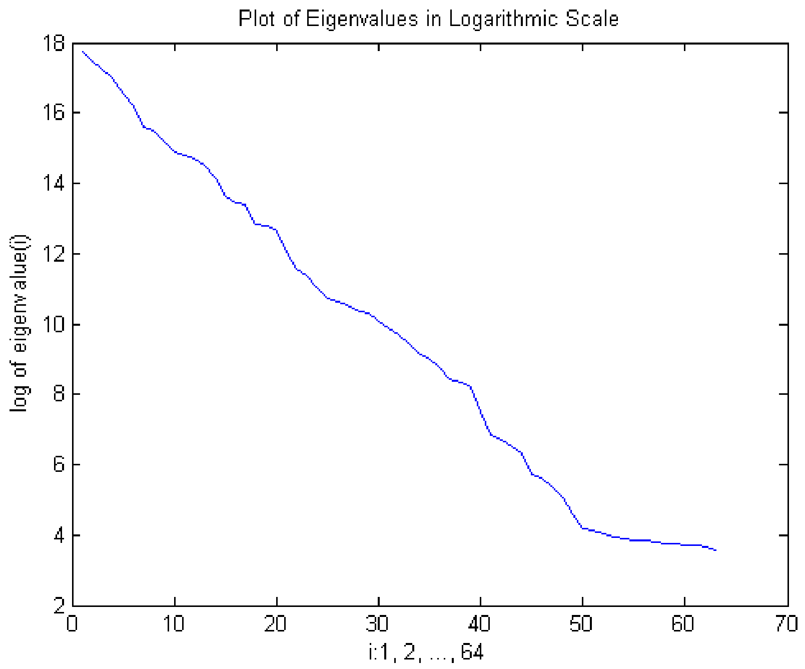


Figure 5.6: Logarithmically scaled plot of the eigenvalues of corresponding eigenechoes of experiment shown in Figure 5.5 in decreasing order

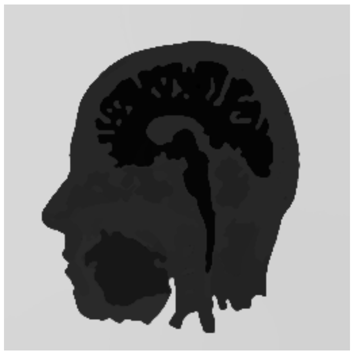
The diffusion echo of a sampled pixel can only reach the other sampled pixels within the neighbourhood or in the best case within the same segment depending on the diffusion time. Hence, using these sampled points that does not provide a similarity measure for each other causes a bad approximation at those locations. This is also visible in the eigenvalue plot in Figure 5.14.

5.6.4 Effect of Sample Size

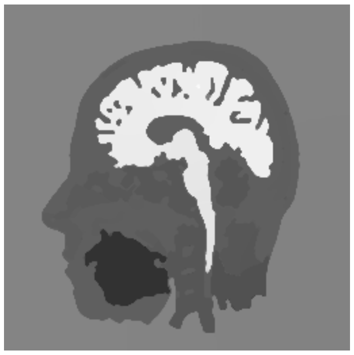
All the experiments up to now use 64 diffusion echoes in a sample. Increasing the sample size can improve the PCA. When the segments are well represented in the samples, they will also be well represented in the eigenechoes. Hence, when the sampling is more dense, more segments have a chance to be represented. Figures 5.15 and 5.17 shows the first 10 eigenechoes of the samples with 256 and 1024 diffusion echoes obtained for the isotropic nonlinear diffusion with Weickert's diffusivity, respectively. It is visible that as the sample size increases, the better the representations get. This can also be seen in the corresponding eigenvalue plots in Figures 5.16 and 5.18.

5.6.5 Representing All Diffusion Echoes

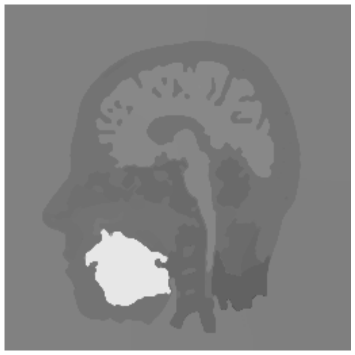
Eigenvalue plots show how the first 10 eigenechoes are good for representing the whole samples. However, we are interested in representing all the diffusion echoes of an image by fewer eigenechoes. In order to test this, we computed all the 65536 echoes for the head image of size 256×256 with the edge-enhancing diffusion with parameters:



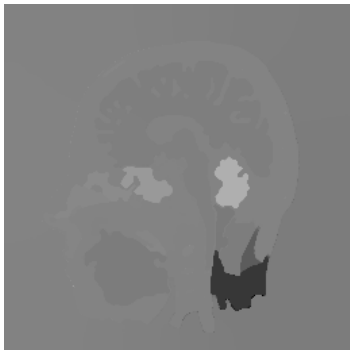
(a) eigenecho 1



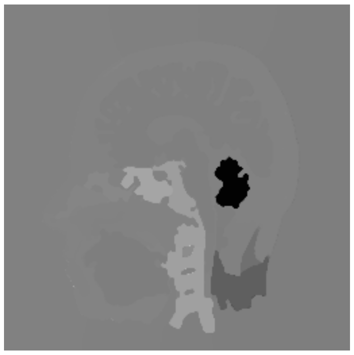
(b) eigenecho 2



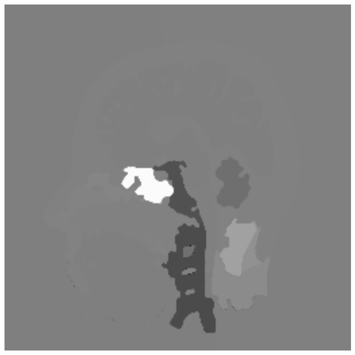
(c) eigenecho 3



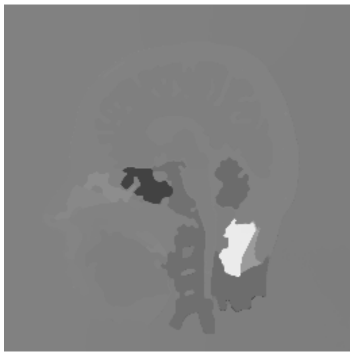
(d) eigenecho 4



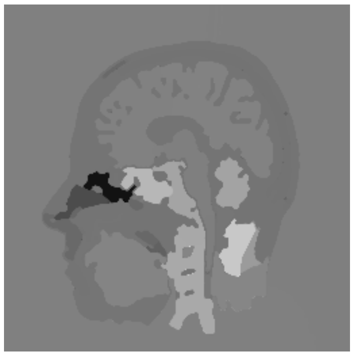
(e) eigenecho 5



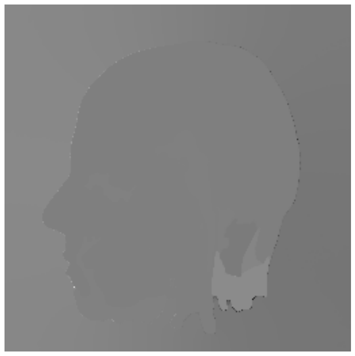
(f) eigenecho 6



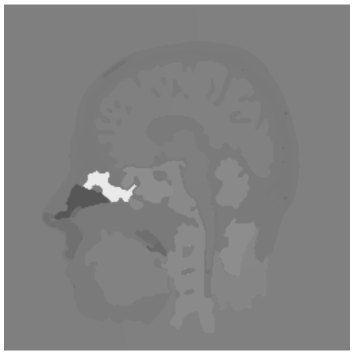
(g) eigenecho 7



(h) eigenecho 8



(i) eigenecho 9



(j) eigenecho 10

Figure 5.7: Eigenechoes for isotropic nonlinear diffusion with Weickert's diffusivity with $\lambda = 3$, stopping time T=100000, $\tau_{stable} = 0.24$, $\sigma = 0.5$, FED cycles=25, and running time=2.2 sec

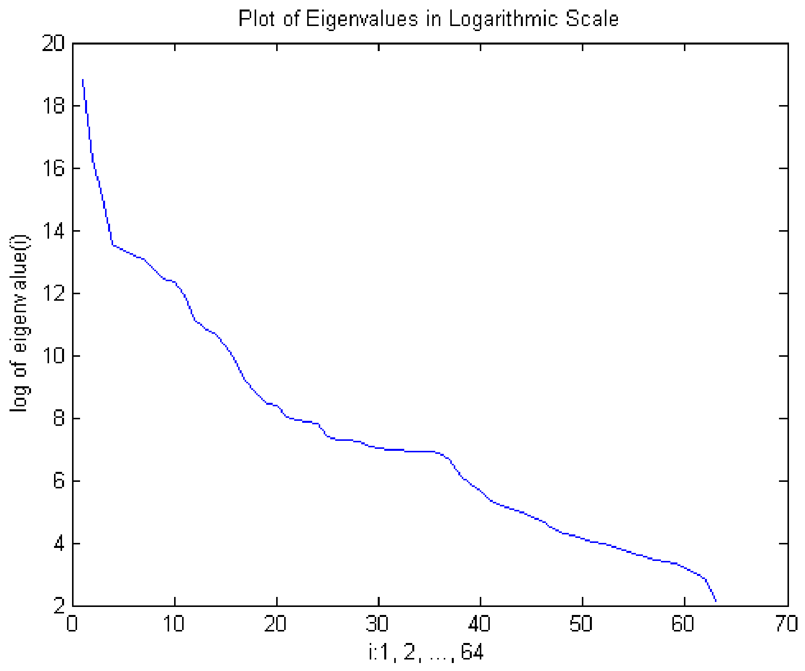
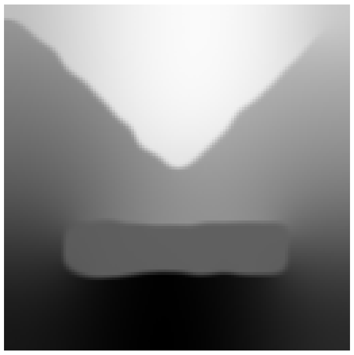
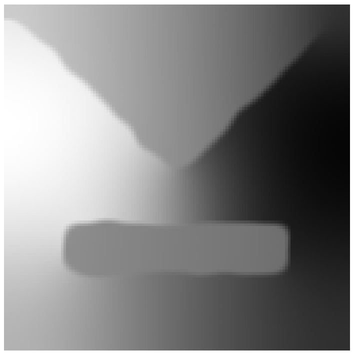
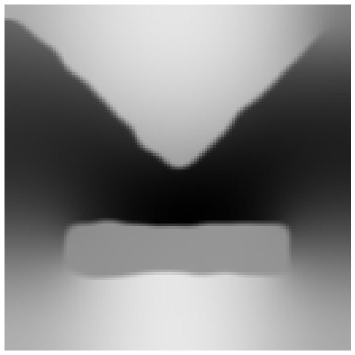


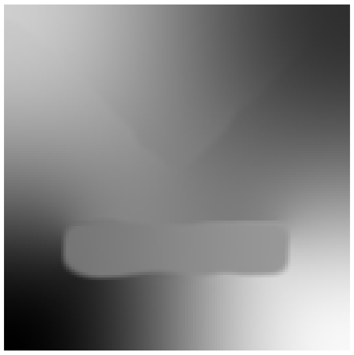
Figure 5.8: Logarithmically scaled plot of the eigenvalues of corresponding eigenechoes of experiment shown in Figure 5.7 in decreasing order

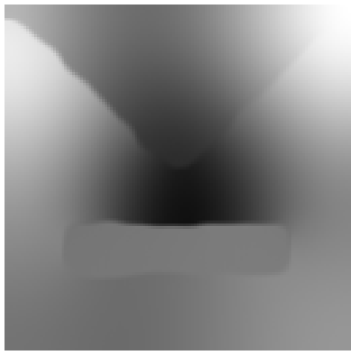
$\lambda = 2$, $\sigma = 0.5$, $\tau_{stable} = 0.24$, FED cycles= 20, and $T = 1000$. The model choice depends on the good trade-off between the diffusion time and the segment representations. With edge-enhancing diffusion, it is possible to obtain sharp segments which do not contain small details in a short diffusion time. The sample is too big to display here; however, the resulting image after diffusion in Figure 5.19 might give an idea about the diffusion echoes. Since this sample is big, PCA may take weeks to compute the eigenechoes. That is why we also computed 4096 uniformly sampled diffusion echoes with the same diffusion model with the same parameters of the head image. Our goal is to represent all 65536 diffusion echoes by the first eigenechoes of 4096 diffusion echoes. We chose 4096 as sample size so that we can perform both PCA and the reconstruction within a realistic time and also we can have less error compared to the sample sizes 64, 256, and 1024.

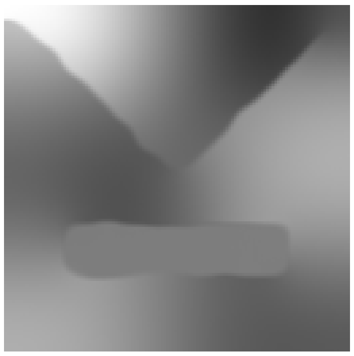
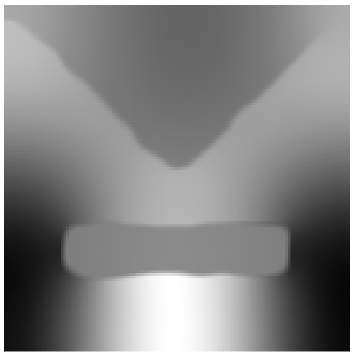
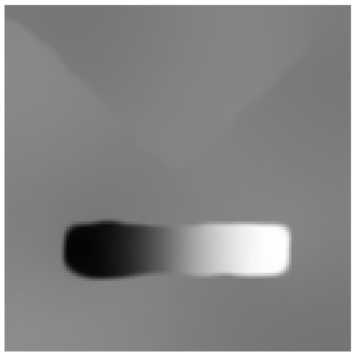
After the PCA with 100 power iterations on the 4096 sampled diffusion echoes, we used the resulting first 30 eigenechoes to reconstruct each mean-subtracted diffusion echo $\boldsymbol{g}$ of the sample of 65536. Equation 5.12 is used for reconstruction [24]. In order to measure the error of the reconstruction we computed the MSE between the original echo $\boldsymbol{g}$ and its reconstruction $\tilde{\boldsymbol{g}}$. The PCA takes 39 min. to compute first 30 eigenechoes and the reconstruction of the all diffusion echoes takes 73 min.

$$\tilde{\boldsymbol{g}} = \sum_{i=1}^{30} (\boldsymbol{g}^T \boldsymbol{v}_i)\boldsymbol{v}_i \tag{5.12}$$


(a) eigenecho 1

(b) eigenecho 2

(c) eigenecho 3


(d) eigenecho 4

(e) eigenecho 5

(f) eigenecho 6


(g) eigenecho 7

(h) eigenecho 8

(i) eigenecho 9

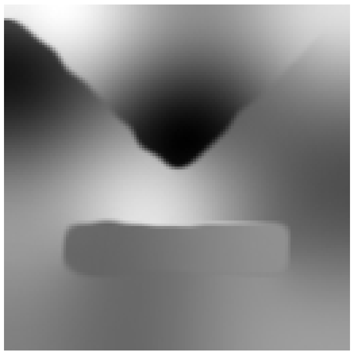
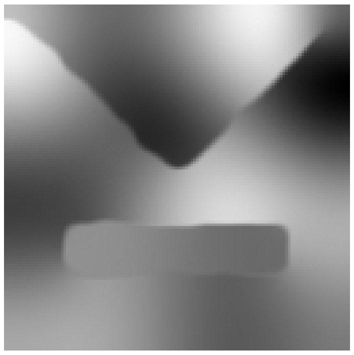
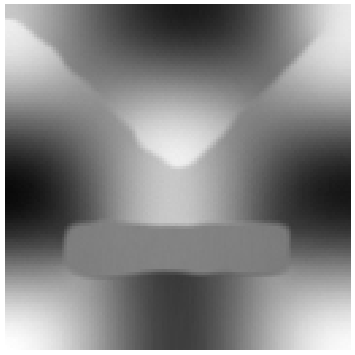

(j) eigenecho 10

(k) eigenecho 11

(l) eigenecho 12

Figure 5.9: Eigenechoes for edge enhancing diffusion with $\lambda = 3$, stopping time T=700, $\tau_{stable} = 0.24$, $\sigma = 2.5$, FED cycles=10, sample size=64, and running time=2 sec

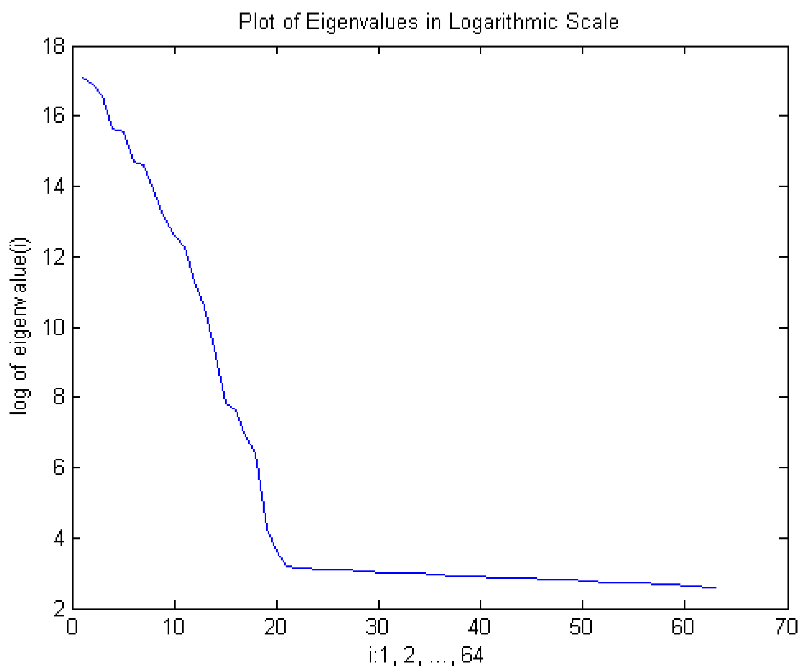
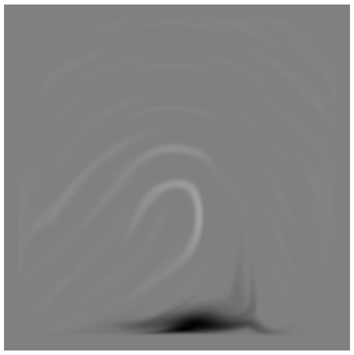


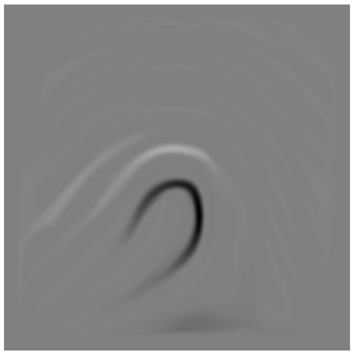
Figure 5.10: Logarithmically scaled plot of the eigenvalues of corresponding eigenechoes of experiment shown in Figure 5.9 in decreasing order

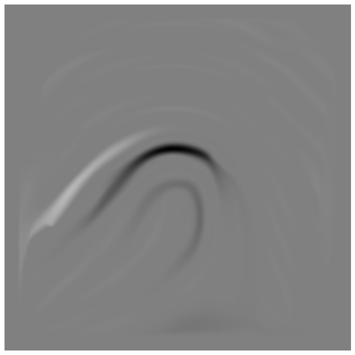
As a result of the reconstruction of all 65536 diffusion echoes, we get minimum MSE of 0.34, maximum MSE of 57.38, and average MSE of 10.83. The difference between the minimum and the maximum MSE is large. This is due to the fact that some of the diffusion echoes are represented well in the sample of 4096 diffusion echoes; therefore, they are reconstructed quite well. However, some of the diffusion echoes are not represented well due to the coarse sampling; therefore, they are reconstructed with higher errors.

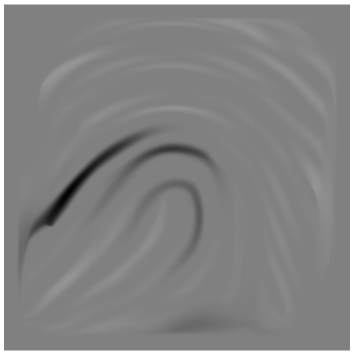
5.7 Discussion

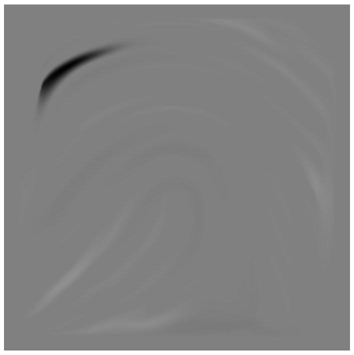
Our experiments show us that the eigenechoes do not give individual segments of the images as we expected with our second conjecture, especially for the diffusion models like coherence-enhancing diffusion. However, with an appropriate choice of diffusion model and diffusivity function, eigenechoes might give interesting representations of the images by highlighting different image structures. For example, isotropic nonlinear diffusion with Weickert's diffusivity with large diffusion times provides with segment-like eigenechoes. It does not suffer from incomplete segments as isotropic nonlinear diffusion with Perona-Malik diffusivity does or rounding of corners as edge-enhancing diffusion does. Also with the increasing sample sizes, more segments of the image have chance to be well-represented in eigenechoes.

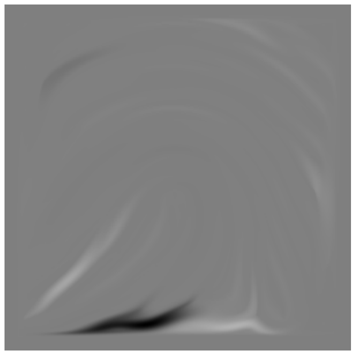

(a) eigenecho 1

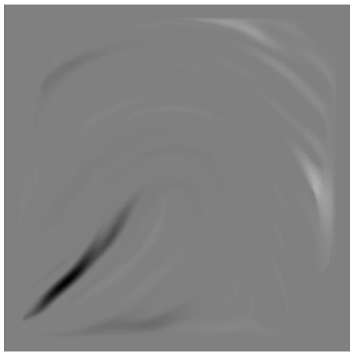

(b) eigenecho 2

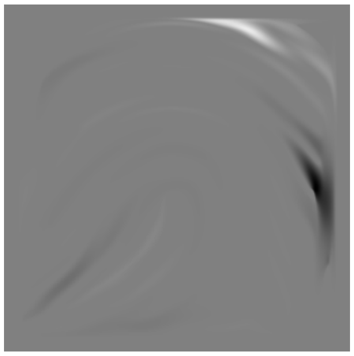

(c) eigenecho 3

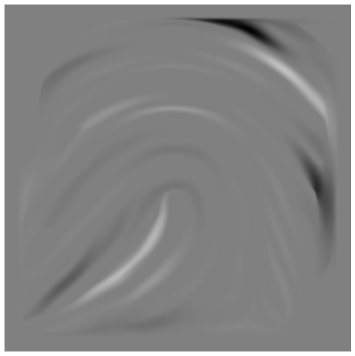

(d) eigenecho 4

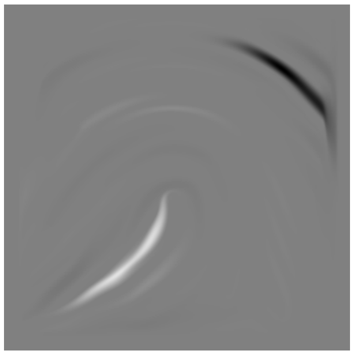

(e) eigenecho 5

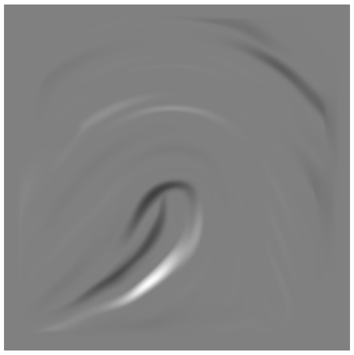

(f) eigenecho 6


(g) eigenecho 7


(h) eigenecho 8


(i) eigenecho 9


(j) eigenecho 10


(k) eigenecho 11

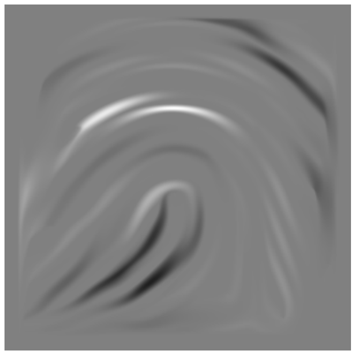

(l) eigenecho 12

Figure 5.11: Eigenechoes for coherence-enhancing diffusion with $C = 1$, $\alpha = 0.001$ stopping time T=300, time step size $\tau_{stable} = 0.24$, FED cycles=10, sample size=64 and running time=2.8 sec

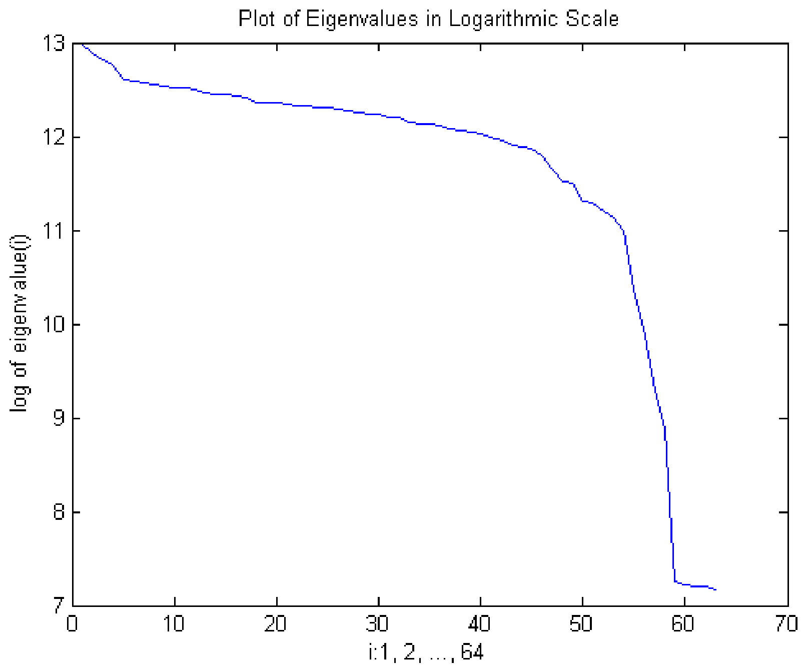
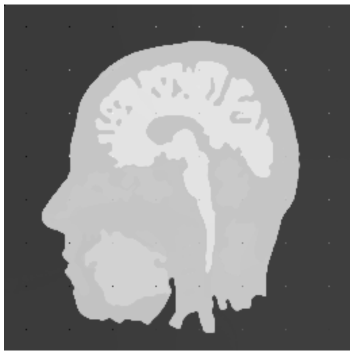


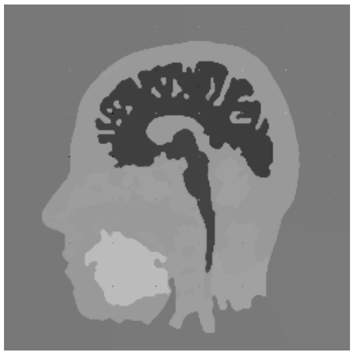
Figure 5.12: Logarithmically scaled plot of the eigenvalues of corresponding eigenechoes of experiment shown in Figure 5.11 in decreasing order

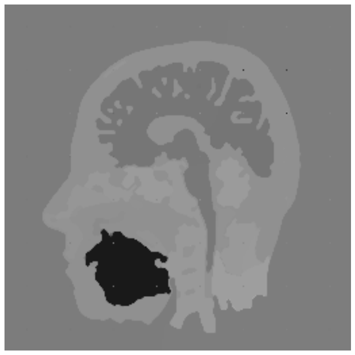
Considering the resulting eigenechoes and their rapidly decreasing eigenvalue plots, we can say that the PCA works well in most of our experiments. The noise which highly effects the quality of the results does not exist in our setting. Each diffusion echo is a well-smoothed representation of the structure it refers to.

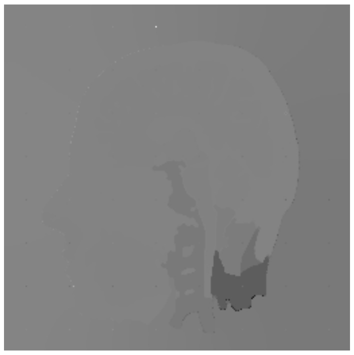
In Subsection 5.6.5, we showed that we are able to represent all 65536 diffusion echoes of the head image with an average MSE of 10.83 using the eigenechoes of the sample with 4096 diffusion echoes. Yet the error range between the minimum and the maximum MSE is too wide, likewise the average MSE itself. In this case, identifying the worst represented diffusion echoes might help us to improve the result of PCA by including them in our samples. Moreover, increasing the sample size or the number of eigenechoes used in the reconstruction would yield less average MSE. Another way of decreasing the error can be finding more appropriate diffusion models with more appropriate parameters.

Another point, worth to discuss, is sampling. While the uniform sampling is more efficient, the nonlinear sampling methods yield more accurate results for the Nystroem approximation [12]. Considering the similarities between the PCA and the Nystroem approximation, it is likely that clever sampling methods can also improve our results. Moreover, with an appropriate sampling method we can already find distinct individual segments so that PCA is not necessary.


(a) eigenecho 1

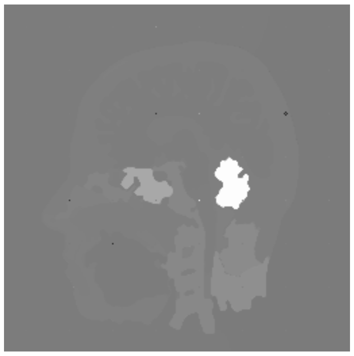

(b) eigenecho 2

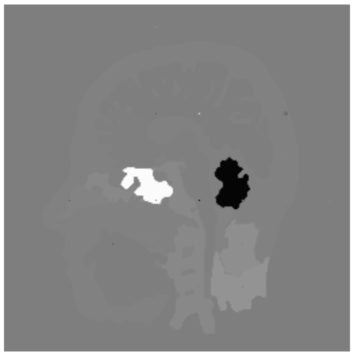

(c) eigenecho 3

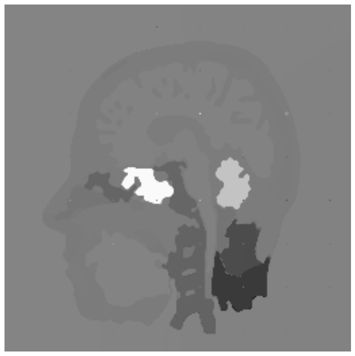

(d) eigenecho 4

(e) eigenecho 5

(f) eigenecho 6


(g) eigenecho 7


(h) eigenecho 8


(i) eigenecho 9

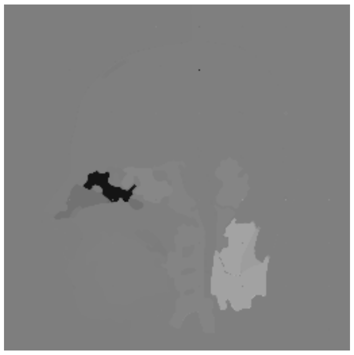

(j) eigenecho 10

Figure 5.13: Eigenechoes with Nystroem approximation for isotropic nonlinear diffusion with Weickert's diffusivity with $\lambda = 3$, stopping time T=100000, $\tau_{stable} = 0.24$, $\sigma = 0.5$, FED cycles=25, and running time=0.56 sec

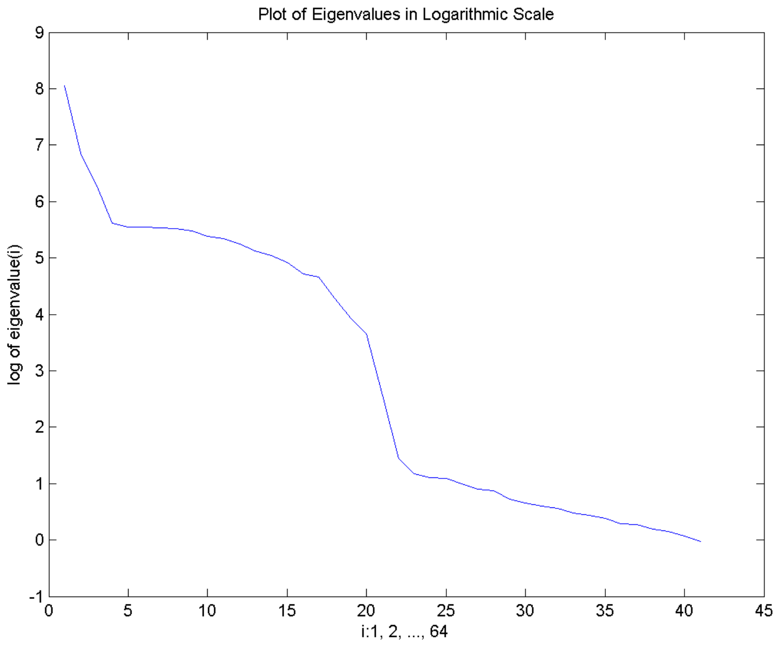
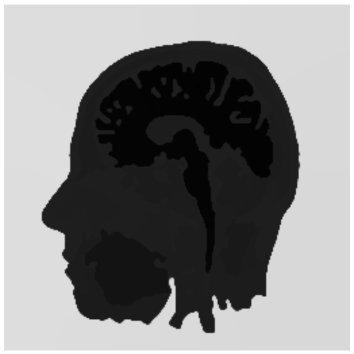
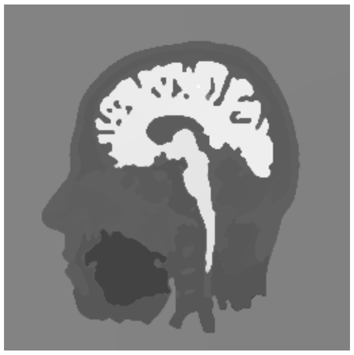
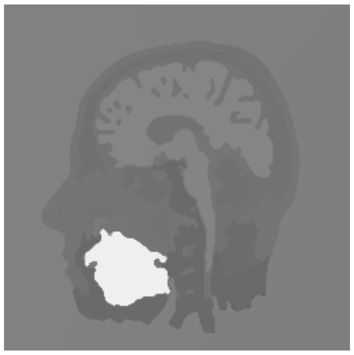
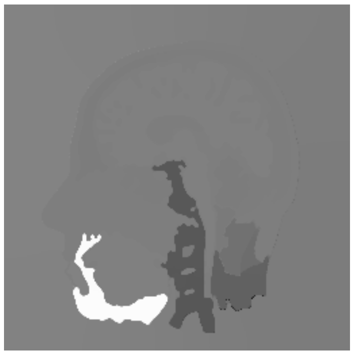


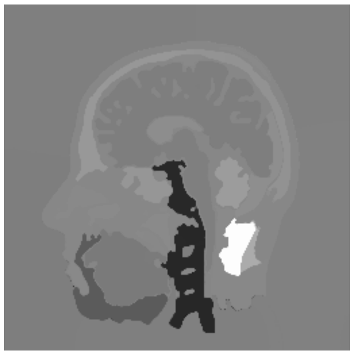
Figure 5.14: Logarithmically scaled plot of the eigenvalues of corresponding eigenechoes of experiment shown in Figure 5.13 in decreasing order

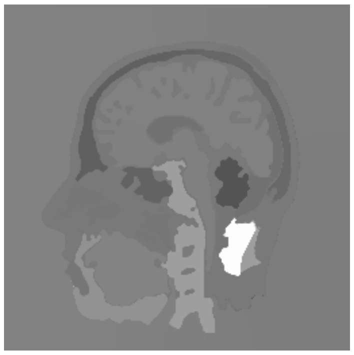

(a) eigenecho 1

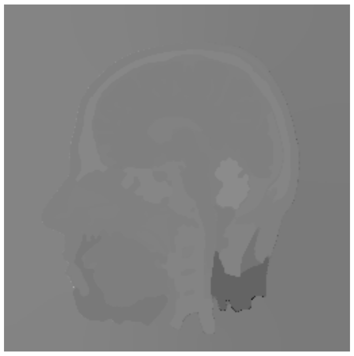

(b) eigenecho 2

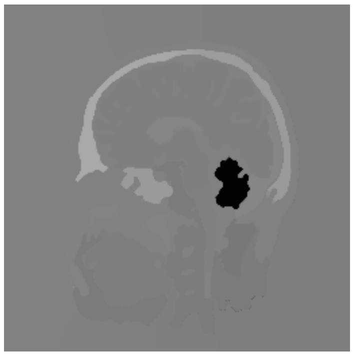

(c) eigenecho 3

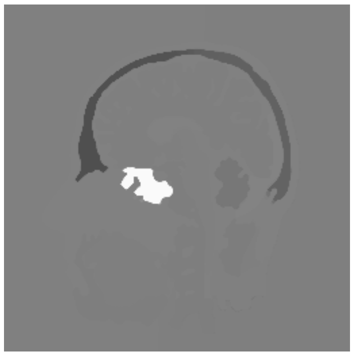

(d) eigenecho 4


(e) eigenecho 5


(f) eigenecho 6


(g) eigenecho 7


(h) eigenecho 8


(i) eigenecho 9

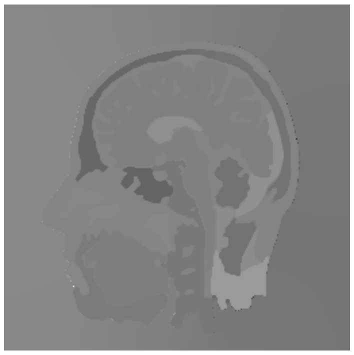

(j) eigenecho 10

Figure 5.15: Eigenechoes for isotropic nonlinear diffusion with Weickert's diffusivity with $\lambda = 3$, stopping time T=100000, $\tau_{stable} = 0.24$, $\sigma = 0.5$, FED cycles=25, sample size=256, and running time=33.1 sec

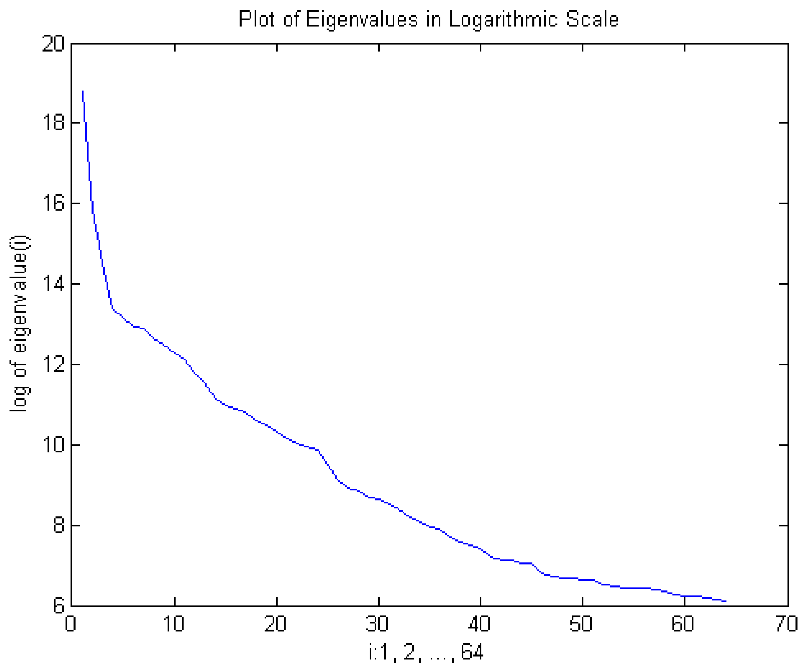
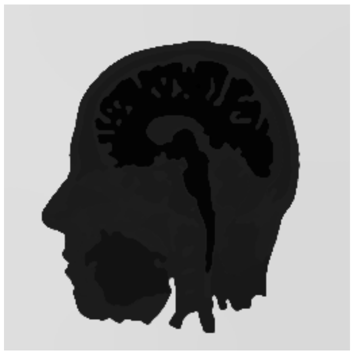
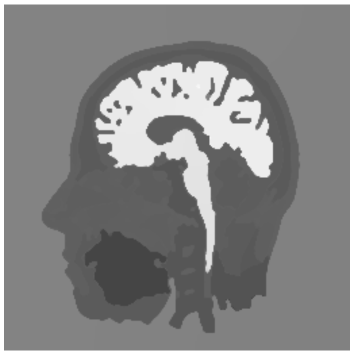
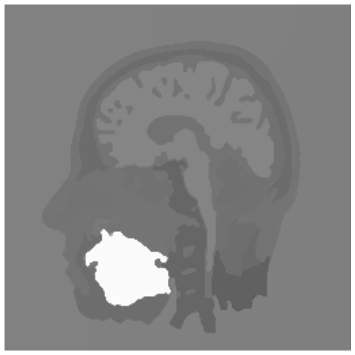
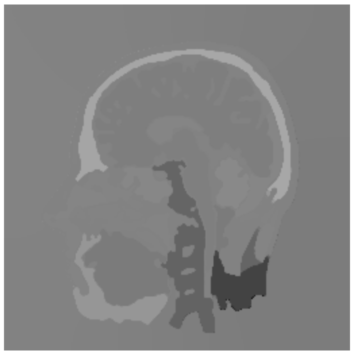


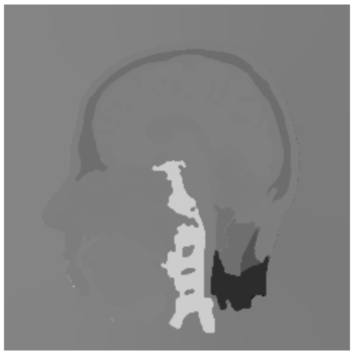
Figure 5.16: Logarithmically scaled plot of the eigenvalues of corresponding eigenechoes of experiment shown in Figure 5.15 in decreasing order

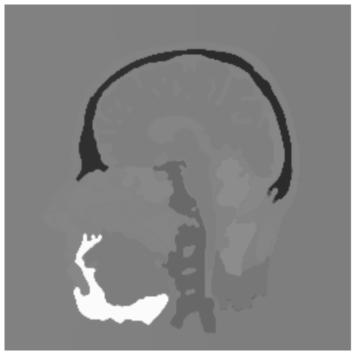

(a) eigenecho 1

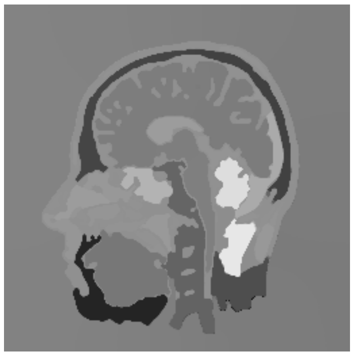

(b) eigenecho 2

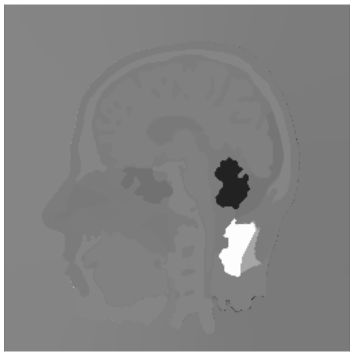

(c) eigenecho 3

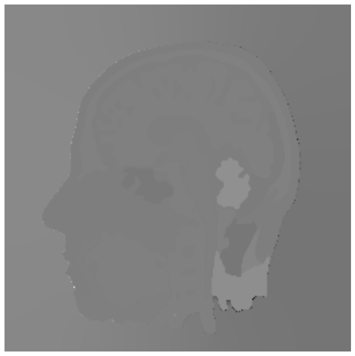

(d) eigenecho 4


(e) eigenecho 5


(f) eigenecho 6


(g) eigenecho 7


(h) eigenecho 8


(i) eigenecho 9

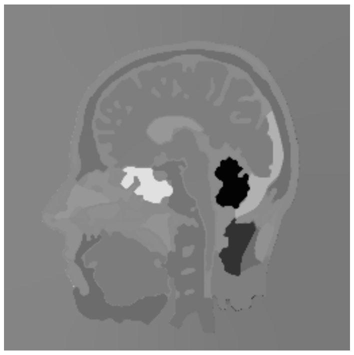

(j) eigenecho 10

Figure 5.17: Eigenechoes for isotropic nonlinear diffusion with Weickert's diffusivity with $\lambda = 3$, stopping time T=100000, $\tau_{stable} = 0.24$, $\sigma = 0.5$, FED cycles=25, sample size=1024, and running time= 20 min

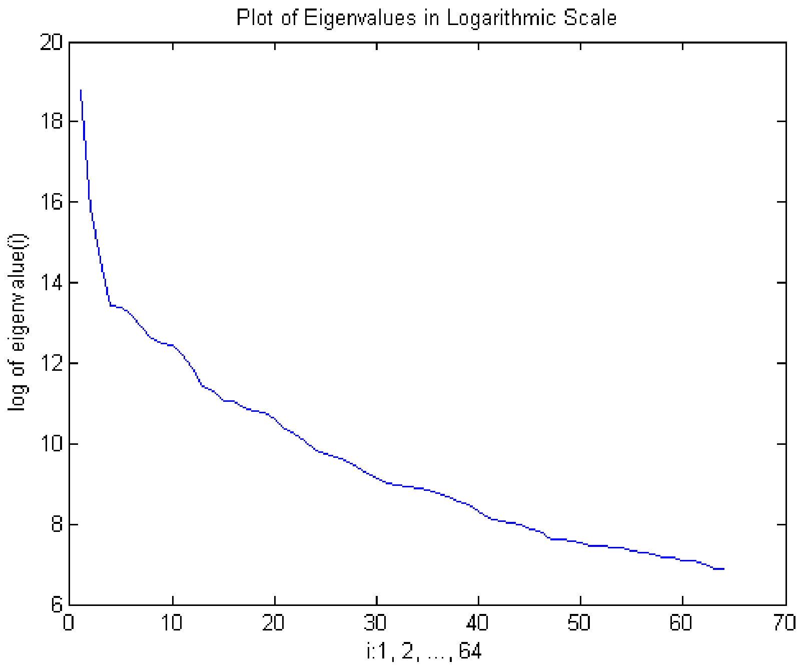


Figure 5.18: Logarithmically scaled plot of the eigenvalues of corresponding eigenechoes of experiment shown in Figure 5.17 in decreasing order

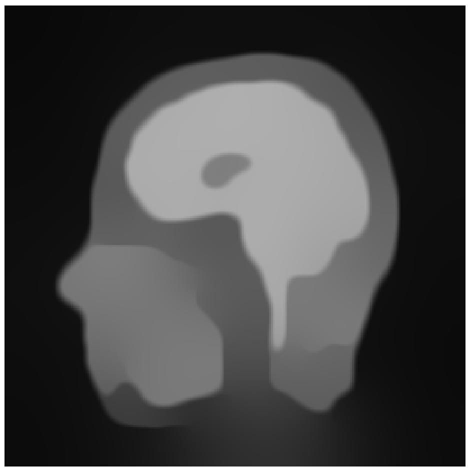


Figure 5.19: Resulting image after edge-enhancing diffusion with $\lambda = 2$, $\sigma = 0.5$, $\tau_{stable} = 0.24$, FED cycles$= 20$, and $T = 1000$

Chapter 6

Diffusion Echo-Driven Segment Scale-Space

In Chapter 5, we analysed important structures of an image by performing PCA on uniformly sampled diffusion echoes of it. With isotropic nonlinear diffusion with Weickert's diffusivity, we even obtained interesting segment-like eigenechoes with a good choice of diffusion time. However, the eigenechoes are highly influenced by the diffusion echoes and these diffusion echoes are highly dependent on the diffusion times. A diffusion echo gives better localization with small diffusion times. In this case, the resulting diffusion echo is not a complete segment. When it is diffused more, the boundaries of the segments are delocalized.

However, we know that at any scale the diffusion echo holds the affinity measure between pixels. More precisely, at each iteration of the diffusion process the weights that a pixel contributes to each of its neighboring pixels are computed. These contributions reflect how the pixel is *similar* to each neighboring pixel based on a diffusivity function. By computing diffusion echo at each iteration, we also iterate these contributions. Therefore, the diffusion echo holds the affinity measures between the pixel and its neighborhood.

Dam and Nielsen [6] used this affinity measure to merge segments. They decide merging two segments if the average affinity measure between the pixels in two regions are higher than a threshold and vice versa. Getting inspired by this affinity measure and the scale-space representation, which we metioned in Section 2.4, we came up with another idea: "Identify a segment in a coarse scale and trace it back in time for better localization using diffusion echo." We perform the identification of a segment by thresholding any of the diffusion echoes at a coarse scale. Then we trace this segment back in time. At each explicit step backwards, we compute the diffusion echoes of the pixels in 3×3 neighbourhood ($\mathcal{N}(3 \times 3)$) of the segment boundaries using the diffusivities of the previous step. If the pixel contributes more to the inside of the segment, the pixel joins to the segment and vice versa. This process is repeated until time reaches to 0. The detailed algorithm is as follows:

```
$\boldsymbol{s} \leftarrow$ auxiliary image for pixel in the target segment, $i \leftarrow 1$
**for** $i \leq k+1$ **do**
    presmooth image $\boldsymbol{u}$
    find diffusivities for $\boldsymbol{u}$ and save them in $\boldsymbol{d}[i]$
    diffuse $\boldsymbol{u}$ and $\boldsymbol{s}$ with $\boldsymbol{d}[i]$
**end for**
threshold $\boldsymbol{s}$ to have *binary* representation of the segment
**while** $k > 0$ **do**
    **for all** pixels $p \in \mathcal{N}(3 \times 3)$ of segment boundaries of $\boldsymbol{s}$ **do**
        compute diffusion echo $\boldsymbol{e}$ at $p$ with $\boldsymbol{d}[k]$
        $pos \leftarrow 0, neg \leftarrow 0$
        **for** $j \in \mathcal{N}(3 \times 3)$ of $p$ and $j \neq p$ **do**
            **if** $\boldsymbol{s}_j == 1$ **then**
                $pos += \boldsymbol{e}_j$
            **else**
                $neg += \boldsymbol{e}_j$
            **end if**
        **end for**
        **if** $pos > neg$ **then**
            mark $\boldsymbol{s}_p$ as 1
        **else**
            mark $\boldsymbol{s}_p$ as 0
        **end if**
    **end for**
    $k \leftarrow k - 1$
**end while**
```

In Figure 6.1, we show one example of a diffusion echo-driven segment scale-space for isotropic nonlinear diffusion with stopping time T= 4096 together with the resulting image. Since the diffusion process gets slower with increasing diffusion time, we sampled the segments through the scales exponentially as stated for each scale in the figure. This means that we increased our sampling rate exponentially as we get closer to time zero. In this experiment we chose the learned diffusivity which decays slower in order to have a coarser segment and to be able to observe clear changes between scales. The reason for the choice of $\lambda = 0.14$ comes from the natural image statistics [17]. Since we are interested in precise localizations, therefore, in each stable explicit diffusion step, we did not use FED scheme in this experiment. Moreover, we did not apply pre-smoothing prior to diffusivity computations.

We see from Figure 6.1 that the diffusion echoes help us trace the segment boundary back in time. We can see that the boundary moves towards the actual boundary of the segment. However, the final locations of the boundary pixels are not the exact locations. They are stuck at strong edges through the scales. Yet this scale-space representation can be more useful with edge-enhancing diffusion. It can trace back

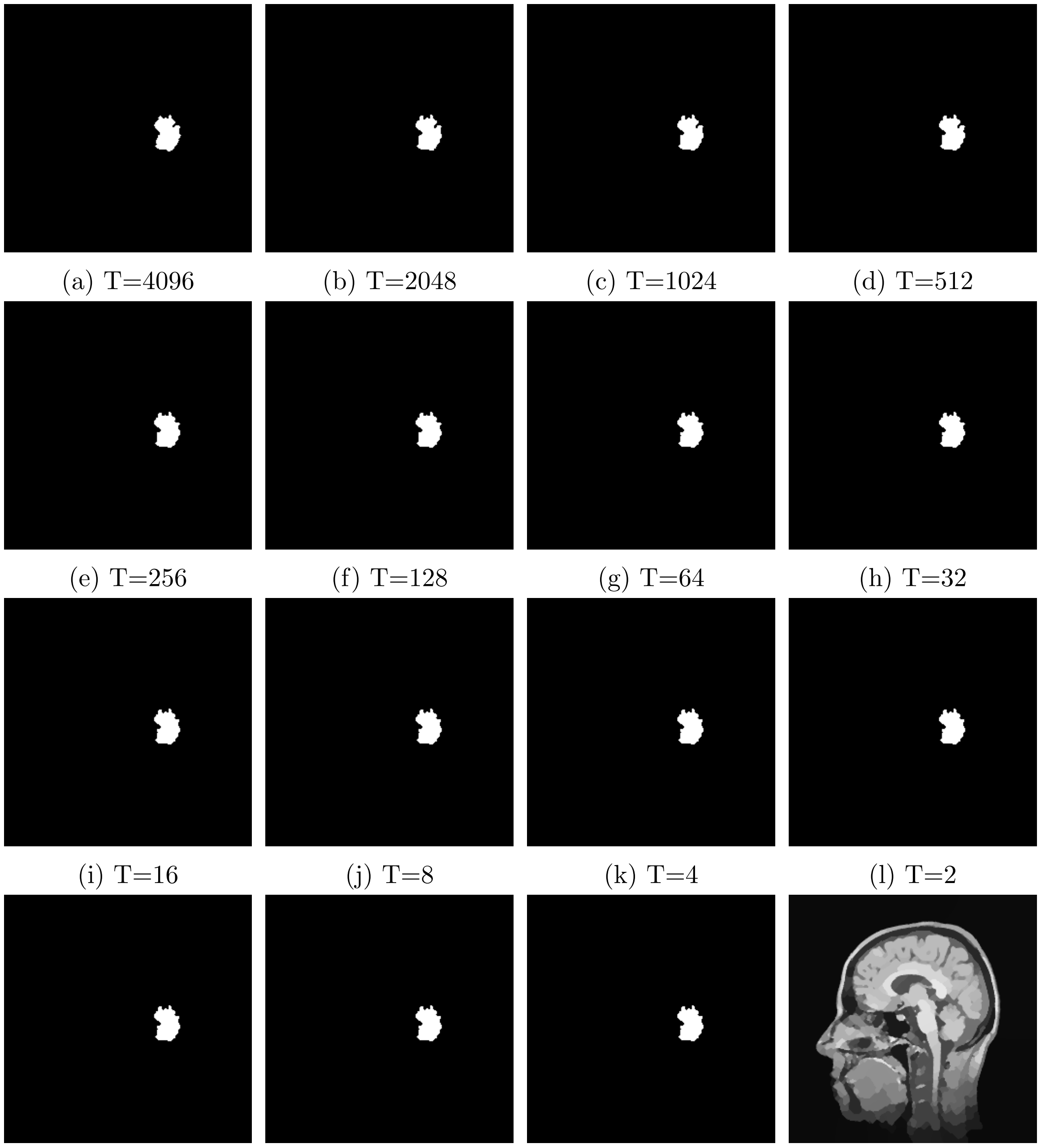

(a) T=4096 (b) T=2048 (c) T=1024 (d) T=512
(e) T=256 (f) T=128 (g) T=64 (h) T=32
(i) T=16 (j) T=8 (k) T=4 (l) T=2
(m) T=1 (n) T=0.5 (o) T=0.25 (p) result

Figure 6.1: Diffusion echo-driven segment scale-space for isotropic nonlinear diffusion with learned diffusivity with T=4096, $\tau = 0.25$, and $\lambda = 0.14$

the rounded corners caused by this diffusion model. Moreover, it can be more useful for localizations of coarser segments in coarser scales.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Diffusion processes are used widely in image processing and computer vision. They offer well-founded models from linear settings to nonlinear and anisotropic settings. Each model differs in how they use the underlying image structures to steer the diffusion. However, in nonlinear models, understanding this behaviour is not trivial. One good way of visualizing this behaviour is computing diffusion echoes. They tell everything about the diffusion filter. However, in sequential settings, it has computational complexity of $O(n^2)$ for an image with n pixels. Also it is unpractical to store all n diffusion echoes having n pixels as the original image. To this end, in our work we exploited parallel programming on GPU with CUDA along with the FED scheme for the efficient computation of the diffusion echoes and used the PCA to represent them in a compact way.

The Efficient Computation of The Diffusion Echo. Firstly, we introduced the need for the diffusion echoes as better visual descriptors for nonlinear diffusion schemes. Then we established a background knowledge for the reader by introducing various diffusion schemes, their discretizations, the FED scheme, the scale-space concept and the MSE. We introduced the diffusion echo in a detailed way, with examples of various diffusion models. In order to be able to compute many diffusion echoes in a fast way, we focused on efficient parallel algorithms. We introduced GPGPU computing to show that our problem is a good fit for parallel computations. Then we introduced the software environment CUDA in detail to show how one can steer their applications to parallel algorithms to gain speed ups if applicable. Following the necessary information, we explained how to re-implement sequential diffusion algorithms with the FED scheme in a parallel way. With the assumption that pixels within a segment give similar diffusion echoes, we only computed the diffusion echoes of the sampled pixels of an image. In the following experiments, we saw that with the full parallelization one can reach tremendous speed ups. Within 160 seconds, it is possible to compute 1024 diffusion echoes of a 256×256 image diffused until large stopping times. Furthermore, using texture and surface bindings in CUDA, better memory alignments for images can be obtained. In our experiments, we also gained a further speed up by texture and surface bindings.

All these efficient computations allowed us to analyse many diffusion echoes sampled from an image and diffused until large stopping times. We experimented with different diffusion models and different parameters to better visualize the underlying image structures that each model tries to emphasize. These image structures led us to finding a compact representation for all the diffusion echoes of an image.

The Compact Representation of the Diffusion Echo. With these representation and memory concerns, we focused on an important statistical data analysis method known as PCA. We explained how to apply the PCA on diffusion echoes in detail. Following the PCA, we mentioned the Nystroem approximation to be able to compare the results of both methods. In the experiments with the PCA, we obtained interesting eigenechoes corresponding to the largest eigenvalues. They are good representations for the image structures emphasized by the diffusion models. Furthermore, we showed that the first 30 eigenechoes resulting from the PCA on 4096 edge-enhancing diffusion echoes can represent all 65536 diffusion echoes of 256×256 head image with average MSE of 10.83. Also we experimented with the Nystroem approximation and analysed the differences between two methods and the possible reasons for these differences. The PCA outperforms the Nystroem extension in our diffusion context.

Diffusion Echo-Driven Segment Scale-Space. We presented an application using the diffusion echo. The diffusion echo is used to create a scale-space that traces back the segments in time for better localization. We displayed an example of this scale-space.

7.2 Future Work

The work in this thesis leads to various future work.

Adaptive Sampling. Uniform sampling is computationally cheap, however, it causes redundancy in diffusion echo samples. The reason is that pixels within the same segment give similar diffusion echoes. Using adaptive sampling methods this redundancy can be reduced to minimum. This sampling method should select pixels within the segments rather than at the segment boundaries.

Error Minimizing Using the PCA With a Gradient Descent Method. When we used the first 30 eigenechoes of the sample with 4096 diffusion echoes in order to reconstruct all 65536 diffusion echoes, some diffusion echoes were reconstructed with higher error. This is due to the fact that those diffusion echoes are not present in the sample of 4096 or less frequent. Hence, exchanging one of the diffusion echoes in the sample of 4096 with one of the diffusion echoes that gives the highest error and re-running the PCA on this new sample might cause decrease in the error. This can be embedded into a gradient descent algorithm as follows. After each run of the PCA the reconstruction error is computed. Then the exchange mentioned is performed. If the exchange of the segments causes a decrease in the average reconstruction error, the exchange is kept, otherwise rejected. This iteration can be repeated until the difference between the minimum and the maximum error is small enough.

Stochastic PCA. Instead of the gradient descent algorithm above which is computationally expensive, stochastic PCA [18] can be implemented. This may allow us to add diffusion echoes on-the-fly when the PCA is computed. Therefore, at the end of the PCA only the important diffusion echoes yielding a better presentation are present in the sample.

Automated Segmentation. The efficient computation and the compact representation methods in this thesis can be thought as a pipeline to find segmentation. However, computing diffusion echoes are dependent on the parameters of the diffusivity functions. A very recent research [17] on natural image statistics offers a family of diffusivities whose parameters are estimated from the natural images. When this learned diffusivity is embedded in the efficient computation part, a segmentation method which only depends on the diffusion time can be obtained. This has partly been tried, however, most of the images in the samples were not suitable for segmentation. A new sample appropriate for segmentation with the corresponding learned diffusivity might lead to an automated segmentation.

Osmosis Echo. Similar to diffusion, osmosis is one of the processes that requires a strong mathematical intuition, hence, a better visual description. Computing the osmosis echo can help us understand the underlying dynamics of this process.

Bibliography

[1] CUDA. http://en.wikipedia.org/wiki/CUDA. Accessed: 2014-11-06.

[2] CUDA programming guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide/#from-graphics-processing-to-general-purpose-parallel-computing. Accessed: 2014-11-03.

[3] GeForce GTX 480 specifications. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-480/specifications. Accessed: 2014-11-06.

[4] Global image denoising. http://users.soe.ucsc.edu/∼htalebi/GLIDE.php. Accessed: 2014-11-09.

[5] F. Catté, P.-L. Lions, J.-M. Morel, and T. Coll. Image selective smoothing and edge detection by nonlinear diffusion. *SIAM Journal on Numerical Analysis*, 32:1895–1909, 1992.

[6] E. Dam and M. Nielsen. Exploring non-linear diffusion: the diffusion echo. In M. Kerckhove, editor, *Scale-Space and Morphology in Computer Vision*, volume 2106 of *Lecture Notes in Computer Science*, pages 264–272. Springer, Berlin Heidelberg, 2001.

[7] S. Grewenig, J. Weickert, and A. Bruhn. From box filtering to fast explicit diffusion. In M. Goesele, S. Roth, A. Kuijper, B. Schiele, and K. Schindler, editors, *Pattern Recognition*, volume 6376 of *Lecture Notes in Computer Science*, pages 533–542. Springer, Berlin, 2010.

[8] S. Grewenig, J. Weickert, C. Schroers, and A. Bruhn. Cyclic schemes for pde-based image analysis. Technical Report 327, Dept. of Mathematics, Saarland University, Saarbrücken, Germany, October 2013.

[9] G. Hellwig. *Partial Differential Equations.* Teubner, Stuttgart, 1977.

[10] T. Iijima. Basic theory on normalization of pattern (in case of typical one-dimensional pattern). *Bulletin of the Electrotechnical Laboratory*, 26:368–388, 1962. In Japanese.

[11] S. Jennewein. Interpretation nichtlinearer Bildverarbeitungsmethoden anhand ihres Filterechos. Master's thesis, Saarland University, Saarbrücken, 2014.

[12] S. Kumar, M. Mohri, and A. Talwalkar. Sampling methods for the Nyström method. *The Journal of Machine Learning Research*, 13(1):981–1006, 2012.

[13] M. Mainberger. Correspondence problems in computer vision. Lecture conducted in Saarland University, Saarbrücken. 2013.

[14] E.J. Nyström. Über die praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben. *Acta Mathematica*, 54(1):185–204, 1930.

[15] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, and J.C. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.

[16] P. Perona and J. Malik. Scale space and edge detection using anisotropic diffusion. In *Proc. IEEE Computer Society Workshop on Computer Vision*, pages 16–22, Miami Beach, FL, November 1987. IEEE Computer Society Press.

[17] P. Peter, J. Weickert, A. Munk, T. Krivobokova, and H. Li. Justifying tensor-driven diffusion from structure-adaptive statistics of natural images. In X.-C. Tai, E. Bae, T.F. Chan, S.Y. Leung, and M. Lysaker, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, volume 3459 of *Lecture Notes in Computer Science*. Springer, Berlin, 2015. to appear.

[18] O. Shamir. A stochastic PCA algorithm with an exponential convergence rate. *arXiv preprint arXiv:1409.2848*, 2014.

[19] J. Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[20] P. Slusallek, J. Kalojanov, and P. Danilewsky. Parallel programming with CUDA. Lecture conducted in Saarland University, Saarbrücken. 2014.

[21] H. Talebi and P. Milanfar. Global image denoising. *IEEE Transactions on Image Processing*, 23(2):755–768, Feb 2014.

[22] J. Weickert. *Anisotropic Diffusion in Image Processing.* Teubner, Stuttgart, 1998.

[23] J. Weickert. Differential equations in image processing and computer vision. Lecture conducted in Saarland University, Saarbrücken. 2013.

[24] J. Weickert. Image processing and computer vision. Lecture conducted in Saarland University, Saarbrücken. 2014.

[25] N. Wilt. *The CUDA Handbook: A Comprehensive Guide to GPU Programming.* Pearson Education, 2013.

[26] A. P. Witkin. Scale-space filtering. In *Proc. Eighth International Joint Conference on Artificial Intelligence*, volume 2, pages 945–951, Karlsruhe, West Germany, August 1983.