Lissage multi-echelle sur GPU des images et volumes avec preservation des details

Nassim Jibai

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques et Informatique**

Arrêté ministérial : 7 Août 2006

Présentée par

Nassim Jibai

Thèse dirigée par **Nicolas Holzschuch**
et codirigée par **Jean-Philippe Farrugia**

préparée au sein du **Laboratoire Jean Kuntzmann**
dans **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Multi-scale Feature-Preserving Smoothing of Images and Volumes on GPU

Thèse soutenue publiquement le **24 mai 2012**,
devant le jury composé de :

Laurent Desbat
Université Joseph Fourier - Grenoble 1, Président
Wilfrid Lefer
Université de Pau et des Pays de l'Adour, Rapporteur
Charles Hansen
University of Utah, Rapporteur
Tamy Boubekeur
Télécom Paristech, Examinateur
Nicolas Holzschuch
INRIA Grenoble - Rhône-Alpes, Directeur de thèse
Jean-Philippe Farrugia
Université Claude Bernard - Lyon 1, Co-Directeur de thèse

Summary

Two-dimensional images and three-dimensional volumes have found their way into our life and became a staple ingredient of our artistic, cultural, and scientific appetite. Images capture and immortalize an instance such as natural scenes, through a photograph camera. Moreover, they can capture details inside biological subjects through the use of CT (computer tomography) scans, X-Rays, ultrasound, *etc.* Three-dimensional volumes of objects are also of high interest in medical imaging, engineering, and analyzing cultural heritage. They are produced using tomographic reconstruction, a technique that combine a large series of 2D scans captured from multiple views. Typically, penetrative radiation is used to obtain each 2D scan: X-Rays for CT scans, radio-frequency waves for MRI (magnetic resonance imaging), electron-positron annihilation for PET scans, *etc.*

Unfortunately, their acquisition is influenced by noise caused by different factors. Noise in two-dimensional images could be caused by low-light illumination, electronic defects, low-dose of radiation, and a mispositioning tool or object. Noise in three-dimensional volumes also come from a variety of sources: the limited number of views, lack of captor sensitivity, high contrasts, the reconstruction algorithms, etc. The constraint that data acquisition be noiseless is unrealistic. It is desirable to reduce, or eliminate, noise at the earliest stage in the application. However, removing noise while preserving the sharp features of an image or volume object remains a challenging task.

We propose a multi-scale method to smooth 2D images and 3D tomographic data while preserving features at a specified scale. Our algorithm is controlled using a single user parameter — the minimum scale of features to be preserved. Any variation that is smaller than the specified scale is treated as noise and smoothed, while discontinuities such as corners, edges and detail at a larger scale are preserved. We demonstrate that our smoothed data produces clean images and clean contour surfaces of volumes using standard surface-extraction algorithms. Our method is inspired by anisotropic diffusion within the volume. We compute our diffusion tensors from the local continuous histograms of gradients around each pixel in images and around each voxel in volume. Since our smoothing method works entirely on the GPU, it is extremely fast.

Keywords: Feature-preserving smoothing, multi-scale, GPU.

Résumé

Les images et les données volumiques prennent une place de plus en plus importante dans notre vie quotidienne que ce soit sur le plan artistique, culturel, ou scientifique. Les données volumiques ont un intérêt important dans l'imagerie médicale, l'ingénierie, et l'analyse du patrimoine culturel. Ils sont créées en utilisant la reconstruction tomographique, une technique qui combine une large série de scans 2D capturés de plusieur points de vue. Chaque scan 2D est obtenu par des methodes de rayonnement : Rayons X pour les scanners CT, ondes radiofréquences pour les IRM, annihilation électron-positron pour les PET scans, *etc.*

L'acquisition des images et données volumiques est influencée par le bruit provoqué par différents facteurs. Le bruit dans les images peut être causée par un manque d'éclairage, des défauts électroniques, faible dose de rayonnement, et un mauvais positionnement de l'outil ou de l'objet. Le bruit dans les données volumiques peut aussi provenir d'une variété de sources : le nombre limité de points de vue, le manque de sensibilité dans les capteurs, des contrastes élevé, les algorithmes de reconstruction employés, *etc.* L'acquisition de données non bruitée est iréalisable. Alors, il est souhaitable de réduire ou d'éliminer le bruit le plus tôt possible dans le pipeline. La suppression du bruit tout en préservant les caractéristiques fortes d'une image ou d'un objet volumique reste une tâche difficile.

Nous proposons une méthode multi-échelle pour lisser des images 2D et des données tomographiques 3D tout en préservant les caractéristiques à l'échelle spécifiée. Notre algorithme est contrôlé par un seul paramètre—la taille des caractéristiques qui doivent être préservées. Toute variation qui est plus petite que l'échelle spécifiée est traitée comme bruit et lissée, tandis que les discontinuités telles que des coins, des bords et des détails à plus grande échelle sont conservés. Nous démontrons les données lissées produites par notre algorithme permettent d'obtenir des images nettes et des iso-surfaces plus propres. Nous comparons nos résultats avec ceux des methodes précédentes. Notre méthode est inspirée par la diffusion anisotrope. Nous calculons nos tenseurs de diffusion à partir des histogrammes continues locaux de gradients autour de chaque pixel dans les images et autour de chaque voxel dans des volumes. Comme notre méthode de lissage fonctionne entièrement sur GPU, il est extrêmement rapide.

Keywords: Lissage en préservant les caractéristiques, multi-échelle, GPU.

Acknowledgments

The decision to do a PhD is never quite a simple one. Specially when it necessitate to travel abroad and away from ones family. It is an adventure full of surprises and discoveries involving sciences, cultures, and one-self. More importantly, it requires passion, courage, and lots of sacrifices specially from the parents.

To my parents, I want to present my unlimited gratitude for their constant and unconditional continuing supports, encouragements, and sacrifices. I want to thank them for supporting my long absence away from them and for their constant presence and advices whenever I needed them.

I want to express my appreciation to my advisor, Prof. Nicolas Holzschuch, for his help and encouragements throughout my PhD work and specially during the writing of my manuscript. I also would like to reserve a special thanks to Prof. Cyril Soler as the idea of this project was initiate by him. I also like to thanks him for allowing me to discover the environment of volumes filtering and for his help and explanation whenever I needed it.

I would like to thank the Artis team for the wonderful three years I spent among you. It was a great pleasure to have met each one of you and a great experience to have worked and shared ideas with you. These moments will forever be engraved in my memory. I would like to specially thanks my friend and colleague, Pierre Landes, for his support and advices during this thesis. Moreover, for his help in proofreading and correcting my French translation of this manuscript. To Olivier and Pierre B., thank you for the wonderful roadtrip we had together in the Californian state. It was my first roadtrip and it was a great one. I also would like to thanks Imma for her great work, perseverance, and help. Finally, I would like to thank: Manu, Laurent, Éric, Mahdi, J-D, Fabrice, Joelle, Hed, Frank, and Charles for their pleasant company.

Last but not least, I would like to thank my friends. A special thanks to Marie Guillard for her repeated help with the French translation of my manuscript. I would like to extend this thanks to the Grillet family, specially Jean-François and Alex, who made me part of their family and made my séjour in Grenoble smooth and pleasant. I would like to thank the girls of the African dance group that I am part of directed by Marielle for the memorable moments I shared with you which will hopefully continue to be. Finally, I would like to thanks my neighbor Antoine for all the pleasant evenings we spent talking around a nice a hot dish.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Contents

1.1 Objective

Two-dimensional images and three-dimensional volumes have become a staple ingredient of our artistic, cultural, and scientific appetite. Images capture and immortalize instances such as natural scenes. They can also capture details inside biological subjects such as the liver. Three-dimensional volumes of objects are also of high interest in medical imaging, engineering, and analyzing cultural heritage. They are produced using tomographic reconstruction. Unfortunately, the acquisition of both, images and volumes, is unavoidably polluted by noise from a variety of sources. We propose two effective and efficient methods for filtering these unwanted noise. However first, we need to define the notions of noise and filtering.

1.2 Noise and Filtering

Noise in everyday life can be define as a sound that is loud, unpleasant, and occasionally causes disturbance. Dogs barking, loud music, mechanical engine running, and road traffic sounds are such examples.

A signal in general is an act to convey a message. Mathematically, it is made from a collection of data that defines a certain phenomenon. Noise is unwanted data over-imposed on the signal. In analog and digital electronics, unwanted data is random perturbation to a wanted signal such as the snow display on television sets. In signal processing, unwanted data is meaningless data such as the "hiss" sound in a recording and radio broadcast produced of unwanted by-product of other activities. Figure 2.1(a) shows a clean signal and Figure 2.1(b) shows a perturbed signal.

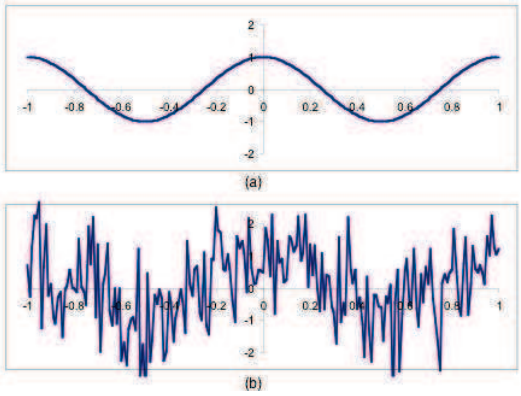
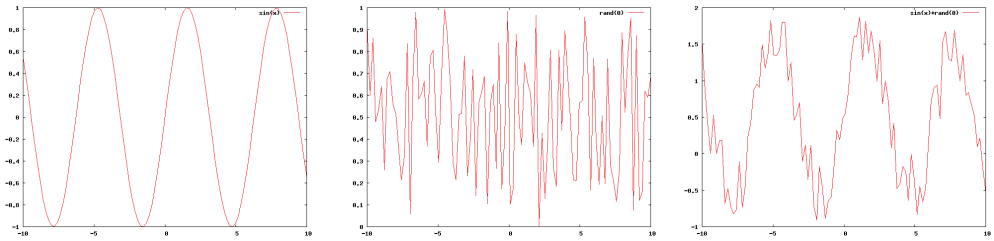


Figure 1.1: *(a) a clean signal and (b) a noisy signal generated from the clean signal.*

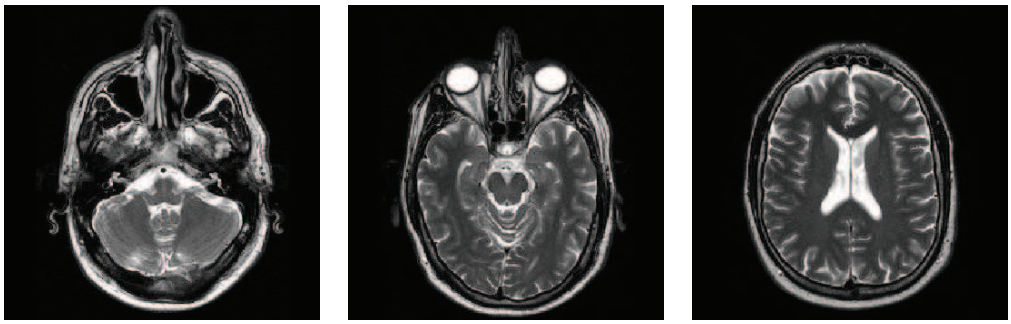


(a) Clean sine wave (b) Random noise value (c) Noisy sine wave

Figure 1.2: *(c) shows a sine wave (a) with random noise (b) added.*

It is possible to add or remove noise from signals—adding noise is easy and can be done for various reasons, *e.g.* to block or alter data transmission. Noise can also be added to encrypt messages or for artistic reasons such as in digital paintings and photos. A simple way to add noise is adding a random or computed signal to the signal transmitted. Figure 2.2(c) shows a sine wave jammed with noise—the result of adding random values (Figure 2.2(b)) to a sine wave (Figure 2.2(a)).

Removing noise from a signal is more difficult as some information about the original signal has been lost. It is usually done using filtering methods. Filtering is a technique used to separate mixture of different substances from one another such as separating the sand from its mixture with water and removing contaminants from engine oil. The concept of filtering can be migrated onto signal analysis where the incoming signal is processed by mathematical operations with transfer functions. In signal processing, noise is most often the high frequencies in the signal and the wanted data is the low frequencies. Thus, filtering out the high frequencies removes the noise and keeps the data needed.



(a) (b) (c)

Figure 1.3: *MRI scans of a brain at different slices level. (a) is a slice of the head. (b) is a deeper slice of the brain showing the eyeballs. (c) is a slice showing the cortex of the brain.*

1.3 Noise in 2D Images

Although the concept of pictures covers a broad range of entities, such as photographs, drawings or medical images, all these type of images can suffer from noise issues. Photos of natural scenes and family reunions are instances captured by film or digital cameras. Medical images capture details inside biological subject such as the brain and knee. Such images are acquired through the use of CT (computer tomography) scans, echography, X-Rays, ultrasound, MRI (Magnetic Resonance Imaging), *etc.* The use of those instruments is not limited to only biological subjects but also to industrial objects such as mechanical parts for analysis. Figure 2.3 shows MRI slices of the brain at different level.

Unfortunately, all type of images are prone to have noise during their acquisition; regardless of the means employed:

- Images produced from cameras are subject to noise due to low-light environment, high exposure index (ISO) setting, high temperature in sensors, or electronic defect such as fluctuation of the electric signal.
- Images acquired through penetrative radiation instruments are also influenced by noise for multiple reasons: lack of sensor sensitivity, high contrasts, non-monochromaticity of the X-Ray source, pixel defects on the sensors, *etc.*

1.4 Noise in 3D Images

Three-dimensional volumes of objects are of high interest in medical imaging, engineering, and analyzing cultural heritage. They can be constructed by techniques such as tomographic reconstruction that combine a large series of 2D scans captured

(a) (b)

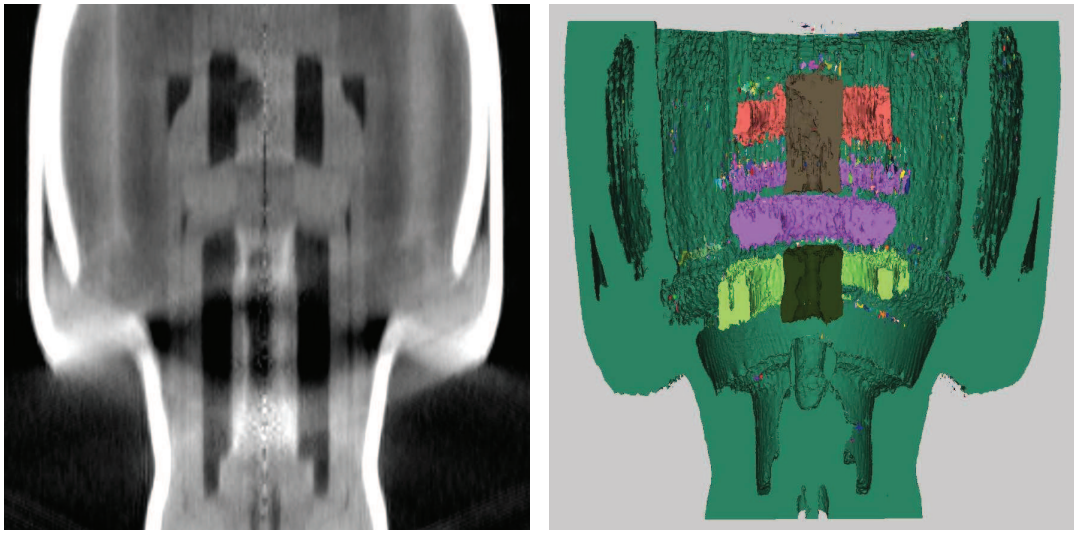
Figure 1.4: *(a) Noisy image of Lena. (b) Noisy MRI of a knee.*

from multiple views. Each 2D scan is obtained by typical penetrative radiation instruments: X-Rays for CT scans, radio-frequency waves for MRI, electron-positron annihilation for PET (positron emission tomography) scans, gamma-rays, sonar, *etc.*

Scanned volumes are generally explored as 2D slices of the volume, as 3D volumetric data, or as 3D meshes after extracting iso-surfaces form the volume data. Similarly to the acquisition of 2D images, there is noise in the reconstructed density data. The sources of noise are numerous: the limited number of views, lack of captor sensitivity, high contrasts, non-monochromaticity of the X-Ray source, imperfect stability of the radiation source, pixel defects on the captor, errors in the reconstruction algorithms, non-uniformity of the absorption process across wavelengths, *etc.*

1.5 Motivation and Challenges

Noise is a universal problem in signals, images, and volumes. They are unavoidable and produce a layer of obscurity over the wanted data making it sometimes hard to interpret them. In images, noise is seen as an artifact made from grains of sand like structure (see Figure 2.4). They make images unpleasant to look at and more importantly, could hide vital features for analysis such as veins in medical images. In volumes, noise is similarly observed when explored as 2D slices (images) of the volume. But when explored as 3D meshes, noise shows itself as crude surfaces contaminated by artifacts such as disconnected components, bridges and multiple holes in the surface. Figure 2.5 shows a CT slice of an inhaler model along its extracted iso-surfaces composed of ~ 2500 components.



(a) (b)

Figure 1.5: *(a) CT slice of an inhaler model. (b) The extracted iso-surfaces of (a) composed of* ~ 2500 *components.*

Images and volumes have become an essential part of our daily life be it culturally, socially, or scientifically. Therefore, it is useful to remove the noise from them for several reasons. From a social and cultural point-of-view, the images and reconstructed objects would be better appreciated. And from a scientific point-of-view, noiseless samples can be studied, analyzed, and understood more effectively and diagnosed correctly be it for medical cases or elsewhere. However, using naive methods, such as averaging, to remove noise would destroy important information. Thus, it is important to filter the noise in an intelligent manner to keep the significant information of the images and volumes. This is not a simple task but rather challenging. In addition to, scalability to large dataset is a problem and the filtering computation costs are expensive.

We are interested in removing noise in images and volumes while preserving the significant information, *i.e.* the sharp features, of their subjects. Furthermore, we want to do so effectively and efficiently since the sizes of images and volumes are constantly growing as the instruments capturing them evolve and larger memory space is created. Today, images and volumes are composed of sizes $> 4096^2$ ($\sim$ 17 million) pixels and $> 1000^3$ (10K million) voxels respectively.

1.6 Contributions

In this thesis, we address the problem of smoothing noisy two-dimensional images and three-dimensional volumes to improve their quality while preserving their sharp

features. We propose a new fast multi-scale smoothing kernel that adapts to the local features of images and volumes. Then, we apply anisotropic diffusion using the computed kernels. The results are smooth images and volumes where the high frequency variation below a user-specified threshold is cancelled. Our algorithm consistently preserves sharp features such as edges and corners. It runs efficiently on parallel processors (such as GPUs) and is scalable to large datasets. Tuning is simple, with only one parameter we call *feature size*—the threshold for the size of features to be preserved.

Our main contributions in this thesis are:

- In Chapter 3, we describe the previous approaches for removing noise in two-dimensional and three-dimensional images. In addition to, we present a detail analysis of two methods very similar to our work and highlight their weaknesses and differences with respect to ours. We start the chapter by explaining what filtering is and its application in one-dimension. Along this, we present the theoretical tools we need and use for filtering. Then, we extend the explanation to two-dimensional and three-dimensional images.

- In Chapter 4, we present the theory of our multi-scale smoothing method for two-dimensional images. We review two tools we use to build our method—*anisotropic diffusion* and *local continuous histogram*. Then, we show how we compute the adaptive diffusion tensors and filter the images using these tools. Finally, we analyze the behavior of our method and compare our results on different types of noisy images to other smoothing methods.

- In Chapter 5, we extent our smoothing method from two-dimensional space presented in Chapter 4 to three-dimension to smooth three-dimensional images (volumes). We also extend the anisotropic diffusion and local continuous histogram from their two-dimensional space to three-dimensional space to use for computing 3D adaptive diffusion tensors and smoothing noisy volumes. Finally, we present results of smoothed volumes and compare them to other smoothing methods.

- We filter images and volumes very fast as we take advantage of the massively parallel processing power of GPU. Its single-instruction-multiple-data (SIMD) architecture allows us to compute the diffusion tensors and anisotropic diffusion in parallel for entire blocks of pixels and voxels since both operations are *embarrassingly parallel*. The GPU implementation of our method gains in performance by a large factor compared to a CPU run-time.

Chapter 2

Introduction

Contents

2.1 Objectif

Les images 2D et volumes 3D prennent une place de plus en plus importante dans notre vie quotidienne que ce soit au niveau artistique, culturel, ou scientifique. Les images 2D capturent et immortalisent des instants comme les paysages naturels en utilisant des appareils photographiques. Elles peuvent également capturer des détails dans des sujets biologiques en utilisant des outils comme la tomographie aux rayons X, ultrasons, *etc.* Les volumes 3D des objets ont aussi un intérêt important dans l'imagerie médicale, l'ingénierie, et l'analyse du patrimoine culturel. Ils sont créés en utilisant la reconstruction tomographique, une technique qui combine une large série de *scans* 2D capturés de plusieurs points de vue. Malheureusement, l'acquisition des images et des volumes est affectée par le bruit provoqué par différents facteurs. Nous proposons deux méthodes effectives et efficaces pour filtrer ces bruits indésirables. En premier lieu, il est nécessaire de définir les notions de "bruit" et de "filtrage".

2.2 Bruit et filtrage

Le bruit peut être défini comme un son bruyant, désagréable, et parfois comme une source de dérangement. L'aboiement des chiens, la musique forte, un moteur en marche, et le son de la circulation routière sont quelques exemples de bruits.

Un signal est, en général, un acte pour transmettre un message. Mathématiquement, un signal est composé à partir d'une collection de données qui définit un phénomène. Le bruit correspond á des données indésirables ajoutées au signal.

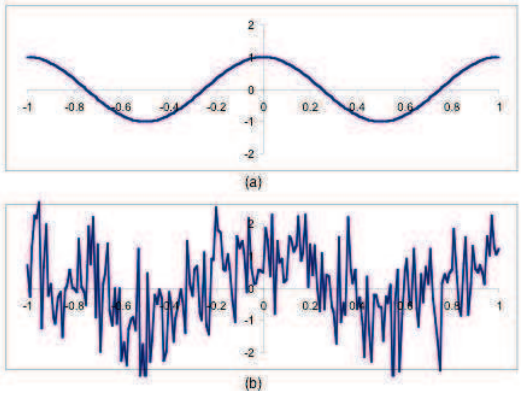
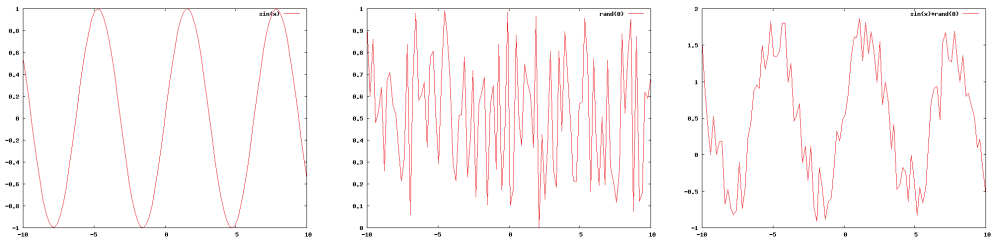


Figure 2.1: *(a) un signal net et (b) un signal bruité généré à partir du signal net.*



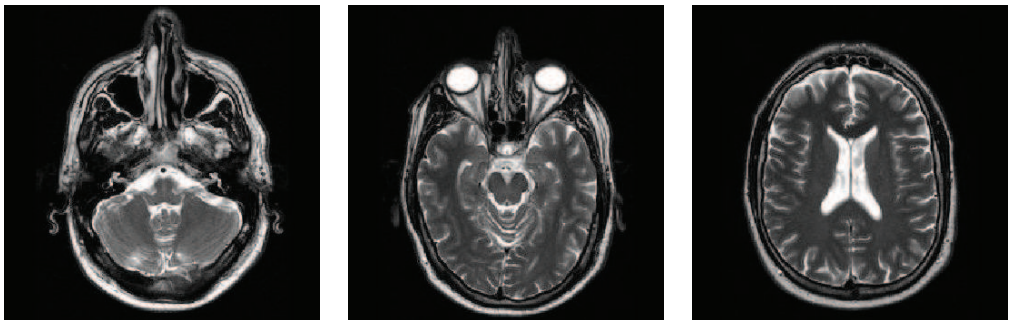
(a) Onde sinusoïdale nette (b) Bruit aléatoire (c) Onde sinusoïdale bruitée

Figure 2.2: *(c) montre une onde sinusoïdale (a) additionnée avec des valeurs de bruit aléatoires (b).*

Dans l'analogique et l'électronique numérique, le bruit correspond á des perturbations aléatoires sur un signal comme l'affichage de neige sur une chaîne de télévision. En traitement du signal, les données indésirables sont sans signification comme le son de "sifflement" dans une radio. Figure 2.1(a) montre un signal net et Figure 2.1(b) montre un signal perturbé.

Il est possible d'ajouter et de supprimer du bruit á partir des signaux. Ajouter du bruit est facile et peut être fait pour de multiples raisons comme par exemple pour bloquer ou changer la transmission des données. Le bruit peut aussi être ajouté pour crypter des messages ou pour des raisons artistiques comme dans les dessins et photographies numériques. Afin d'ajouter du bruit il suffit simplement d'insérer un signal aléatoire au signal transmis. Figure 2.2(c) montre une onde sinusoïdale bruitée qui est le résultat de la somme d'un signal composé de valeur aléatoire (Figure 2.2(b)) et d'une onde sinusoïdale (Figure 2.2(a)).

En revanche, il est plus difficile d'enlever le bruit dans un signal car des informations du signal original peuvent être perdues. La suppression du bruit dans



(a) (b) (c)

Figure 2.3: *Images IRM d'un cerveau à différents niveaux. (a) est un scan IRM d'un cerveau. (b) correspond à un niveau plus profond et montre les globes oculaires. (c) dévoile une partie du cortex.*

les signaux est habituellement faite par des méthodes de filtrage. Le filtrage est une technique utilisée pour séparer le mélange de différents éléments comme par exemple la séparation du sable mélangé à de l'eau. Le concept de filtrage peut être utilisé dans le traitement du signal en agissant sur le signal d'entrée avec des fonctions de conversions mathématiques. Le bruit dans un signal est la plupart du temps représenté par les hautes fréquences et les données importantes du signal sont représentées par les basses fréquences. En conséquence, en enlevant les hautes fréquences du signal on enlève le bruit et préserve les données nécessaires.

2.3 Bruit dans des images

Les images sont utilisées dans une grande variété de domaines comme la photographie, le dessin, et l'imagerie médicale. Cependant, leur qualité est très souvent altérée par des problèmes de bruit. Les photos de paysages naturels et de réunions familiales sont des instants capturés par des appareils photographiques. Les images médicales capturent les détails intérieurs de sujets biologiques comme le cerveau ou le genou. Ce genre d'images est acquis par des scanners tomographiques, échographie, rayon-X, ultrasons, IRM, *etc.* Ces instruments sont non seulement utilisés sur les sujets biologiques mais aussi sur les objets industriels pour l'analyse des pièces mécaniques. Figure 2.3 montre des tranches IRM d'un cerveau à différents niveaux.

Malheureusement, tout type d'image peut être affecté par du bruit pendant son acquisition quelle que soit la manière avec laquelle elle a été génée :

- Les images produites par des appareils photographiques peuvent avoir du bruit à cause de la faible luminosité de l'environnement, de la sensibilité de l'appareil, de la température élevée des capteurs, ou encore de défauts électroniques.

(a) (b)

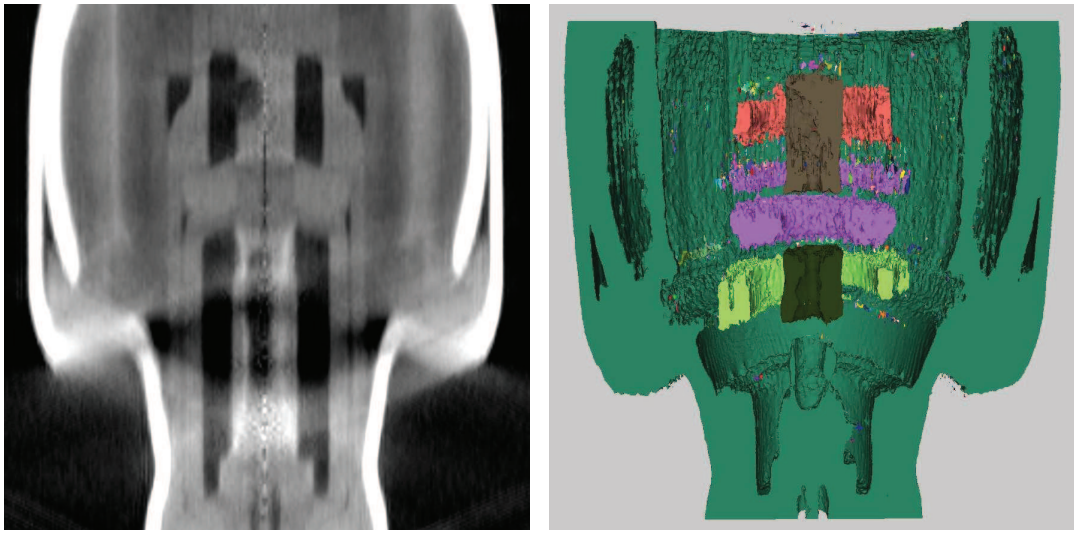
Figure 2.4: *(a) Image bruitée de Léna. (b) Image IRM bruitée d'un genou.*

- Les images acquises par des instruments de radiation sont eux aussi influencés par du bruit pour plusieurs raisons : manque de sensibilité ou défauts des capteurs, contraste élevé, *etc.*

2.4 Bruit dans les images 3D

Les volumes 3D des objets sont très importants dans l'imagerie médicale, l'ingénierie, et l'analyse du patrimoine culturel. Ils peuvent être créé en utilisant la reconstruction tomographique, une technique qui combine une large série de *scans* 2D capturés à partir de plusieurs points de vue. Chaque *scan* 2D est obtenu par des appareils à radiations : des rayons X pour les *scans* CT, des fréquences radio pour l'IRM, sonar, rayons gamma, *etc.*

Les volumes scannés sont généralement explorés via des sections 2D du volume, via les données 3D du volume, ou via des maillages obtenus après l'extraction des iso-surfaces des données du volume. Comme dans les images 2D, la construction des données volumiques est affectée par du bruit généré par de nombreuses sources : le nombre limité de points de vue, le manque de sensibilité des capteurs, des contrastes élevés, des problèmes avec la source du rayon-X, des erreurs dans les algorithmes de reconstruction, *etc.*



(a) (b)

Figure 2.5: *(a) Section tomographique d'un inhalateur. (b) Iso-surface extraite de (a) et composée de* $\sim$ 2500 *composantes.*

2.5 Motivation et défis

Le bruit est un problème universel et inévitable dans les signaux, images, et volumes. Il produit une couche d'obscurité sur les données importantes qui rend parfois difficile l'interprétation de ces données. Dans les images, le bruit apparaît comme des grains de sable (voir Figure 2.4). Il rend les images désagréables à regarder et plus important encore, le bruit peut cacher des caractéristiques vitales comme des veines dans des images médicales. Dans les volumes, le bruit peut être observé de la même façon que dans les images quant ils sont explorés via des sections 2D. Cependant, quand le volume est regardé en tant que maillages 3D, le bruit est visible sur le maillage par la rudesse de sa surface ou la présence d'artefacts comme des composantes déconnectées, des ponts, ou des trous. Figure 2.5 montre une tranche CT d'un modèle d'inhalateur dont l'iso-surface est composée de $\sim$ 2500 composantes.

Les images et les volumes constituent une partie essentielle de notre vie quotidienne que ce soit au niveau artistique, culturel, ou scientifique. Par conséquent, il est utile d'enlever le bruit dans les images et les volumes pour de multiples raisons. D'un point de vue social et culturel, les images et les objets reconstruits seront plus appréciés. D'un point de vue scientifique, les échantillons sans bruit seront mieux étudiés et analysés pour des raisons médicales ou autres. Cependant, l'utilisation de méthodes naïves pour enlever le bruit peut détruire des informations importantes. Ainsi, il est important de filtrer le bruit d'une façon intelligente afin de garder les informations significatives des images et volumes. Cela n'est pas une tâche facile mais plutôt un défi. De plus, le passage à l'échelle vers des données plus larges est

un probléme et le filtrage est très coûteux d'un point de vue calculatoire.

Il est intéressant d'enlever le bruit dans les images et volumes tout en préservant les informations significatives du sujet, c'est-à-dire ses lignes caractéristiques. Par ailleurs, il est important que les méthodes utilisées soient effectives et efficaces puisque les tailles des images et volumes grandissent constamment avec l'évolution des instruments qui les capturent. Aujourd'hui, les images et volumes peuvent être de taille $> 4096^2$ (~ 17 million) pixels et $> 1000^3$ (10K million) voxels respectivement.

2.6 Contributions

Dans cette thèse, on s'intéresse au problème de lissage des images et volumes bruités pour améliorer leur qualité tout en préservant leurs caractéristiques significatives. Nous proposons un nouveau noyau de lissage qui est rapide, multi-échelle et s'adapte aux caractéristiques locales des images et des volumes. Ensuite, on utilise d'autres noyaux calculés pour appliquer une diffusion anisotrope. Les résultats sont des images et volumes lissés, dont les hautes fréquences en-dessous d'une valeur spécifiée par l'utilisateur sont éliminées. Notre algorithme tourne efficacement sur processeur parallèle (comme les GPUs) et peut être appliqué à des données plus larges. Le réglage est relativement facile, avec un seul paramètre que l'on appelle *taille caractéristique*—la taille minimale des caractéristiques que l'on souhaite préserver.

Nos principales contributions présentées dans cette thèse sont:

- Dans le Chapitre 3, nous décrirons les approches antérieures sur la suppression du bruit appliquée à des images et des volumes. De plus, nous présenterons une analyse détaillée de deux méthodes très proches de la nôtre en surlignant leurs faiblesses et différences par rapport à cette dernière. Nous commencerons le chapitre par expliquer le filtrage et ses applications à une dimension. Puis, nous présenterons les outils théoriques relatifs au filtrage de signaux. Enfin, nous expliquerons le filtrage d'images 2D et 3D.

- Dans le Chapitre 4, nous présenterons la théorie de notre méthode multi-échelle pour lisser les images 2D. Nous rappellerons deux outils utilisés ensuite: la *diffusion anisotrope* et *l'histogramme continu local*. Ensuite, nous montrerons comment sont calculés les tenseurs adaptifs de diffusion et de filtrage en utilisant ces outils. Enfin, nous analyserons les comportements de notre méthode et comparerons nos résultats sur différents types d'images bruitées avec d'autres méthodes de lissage.

- Dans le Chapitre 5, nous prolongerons notre méthode de lissage de l'espace 2D à l'espace 3D pour lisser des volumes 3D. Nous étendrons aussi la diffusion anisotrope et l'histogramme continu au cas 3D pour calculer les tensors de diffusion adaptifs nécessaires au filtrage de volumes bruités. Finalement, nous présenterons les résultats des volumes filtrés et nous comparerons ces résultats à d'autres méthodes de filtrage.

- Nous filtrons les images et volumes très rapidement car nous tirons avantage de la puissance de calcul parallèle du GPU. L'architecture du GPU nous permet de calculer les tensors de diffusion et d' appliquer la diffusion anisotrope à des blocs entriers de pixels ou voxels en parallèle. L'implémentation de notre méthode sur GPU nous donne une performance accrûe par rapport à une implémentation sur CPU.

Chapter 3

Theoretical Background and Review of Previous Work

Contents

3.1 Introduction

Digital data such as television, photography images, medical imaging, and meshes have become a significant part of daily life. However, their acquisition is prone to be contaminated by parasites. These parasites also known as noise can be generated from several different sources such as low light environment, thermal variation, and the processing and discretization of the data.

In this chapter, we describe the previous approaches for removing noise in two-dimensional and three-dimensional images. We start by explaining what filtering is and its application in one-dimension. Along this, we present the theoretical tools we need and use for filtering. Then, we extend the explanation to 2D and 3D images.

Introduction

Les données numériques, télévisuelles, photographiques ou médicales, constituent une partie importante de notre vie quotidienne. Cependant, leur acquisition est contaminée par des signaux parasites. Ces parasites, aussi connus comme du bruit, sont produits par de nombreuses sources différentes, comme la faible luminosité de l'environnement, la variation thermique, et le traitement et la discrétisation des données.

Dans ce chapitre, nous décrirons les approches antérieures permettant d'enlever le bruit dans les images 2D et 3D. Nous commencerons par expliquer le filtrage et ses applications dans le cas unidimensionnel. Puis, nous présenterons les outils théoriques relatifs au filtrage de signaux. Enfin, nous expliquerons le filtrage d'images 2D et 3D.

3.2 Signal Processing

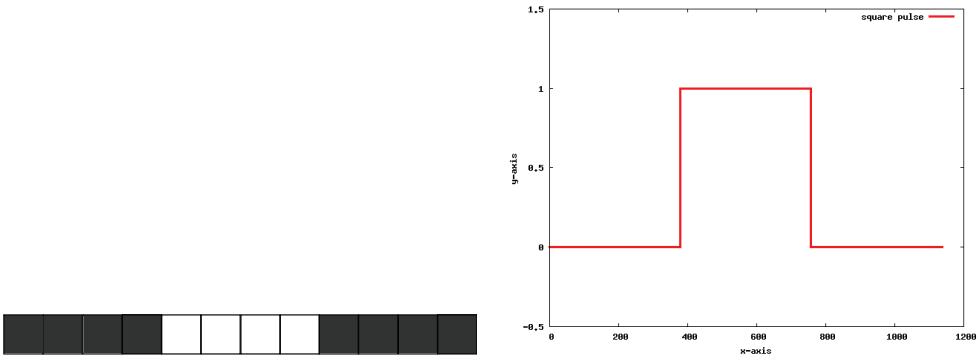
A signal is an abstract, symbolic, or physical representation of information such as communication, audio, music, speech, language, images, graphics, sonar, radar, *etc.* *Signal processing* is essential for converting information from one format to another as well as design complex systems to interact with human and environment. These include generating, transforming, transmitting the signal as well as interpreting it with the aid of software and hardware.

3.3 Signal Processing and Smoothing in 1D Signal

Figure 3.1 shows an example of a one-dimensional signal. This signal is free from noise and has sharp discontinuities. The sharp discontinuities, zero-to-one and one-to-zero values, of the signal signify a change of region and thus can be used to identify areas separating these regions.

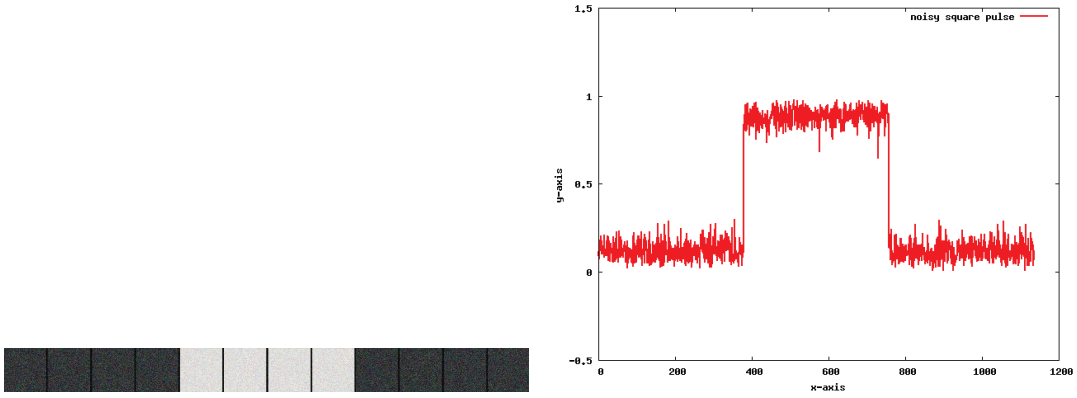
Figure 3.2 shows the one-dimensional signal of Figure 3.1(a) with random noise. Noise is present by the bright and dark spots added in the black and white regions respectively. Figure 3.2(b) shows the signal graphically. The noise in the signal can be removed by filtering out the signal from the rapid fluctuating values.

Two of the most well known filtering methods are convolution and bilateral filtering. We discuss them next.



(a) (b)

Figure 3.1: *(a) A one-dimensional signal and (b) its respective plot. The signal is free from noise and has sharp discontinuities separating regions.*



(a) (b)

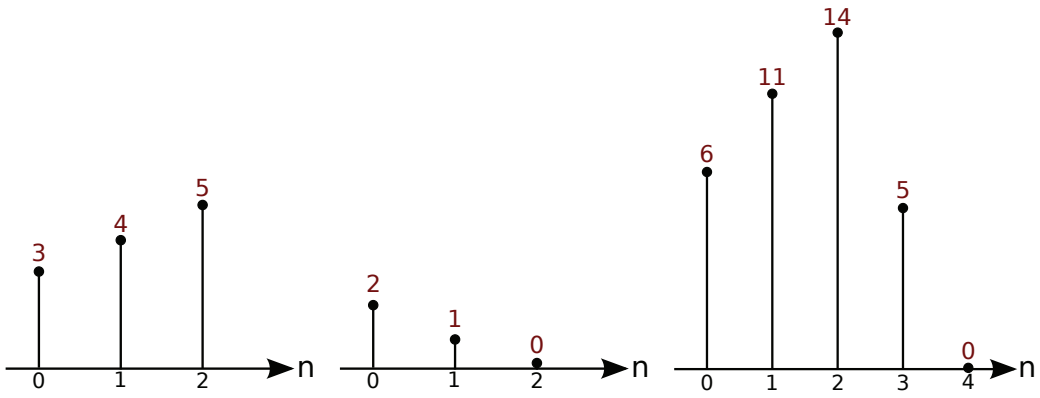
Figure 3.2: *(a) A one-dimensional signal with random noise and (b) its respective plot.*

3.3.1 Convolution

A convolution [Funkhouser 1995] is a mathematical operation applied on two functions f and g; an original function and a mask function also known as a filter function. The operation produces a third function h which is a modified version of the original one altered by the mask:

$$h(y) = (f \otimes g)(y) = \int_{-\infty}^{+\infty} f(x)g(y-x)\,dx. \tag{3.1}$$

where f is the input function, g is the filter function, and $\otimes$ denotes the convo-



(a) Input function f (b) Filter function g (c) Output of $f \otimes g$

Figure 3.3: *Convolving an input function (a) and a filter function (b). (c) The result output function.*

---

lution operator. If the signal is discrete rather than continuous, each output value of a convolution is the sum of the product of the two functions over a finite range:

$$h[n] = (f \otimes g)[n] = \sum_{m=-\omega}^{\omega} f[m]g[n-m] \tag{3.2}$$

where $-\omega$ and ω define the domain range of f. Figure 3.3 shows an example of discrete convolution.

Convolution is a very powerful concept and a fundamental tool in signal processing and analysis. It can be seen as centering the filter function at each point along the input function, multiplying the filter everywhere by the value of the function, and then summing them up. This is visualized in Figure 3.4 where we convolve two rectangle pulses. The input signal, in green, is stationary, centered at the origin. The filter function, in red, slides along the horizontal axis from left to right. As the functions overlap, the output value of the point centered by the filter function is equal to their intersected area; shown in grey. The output function starts with an area of zero as they are disjoint. This value increases linearly as the functions start to overlap reaching a maximum when they superimpose. It then decreases linearly till they separate again. The result is a triangle function shown on the bottom line of Figure 3.4.

Noise in signals consists mostly of high frequencies. Removing, or smoothing the high frequencies from the signal reduces the noise. This can be done by convolving an input signal with a *low-pass filter*. Low-pass filters reduce the amplitude of high frequencies and pass the lower ones. The **sinc function** is an ideal low-pass filter (Figure 3.5(b)). It has the general form: $\sin(x)/x$ and is given by:

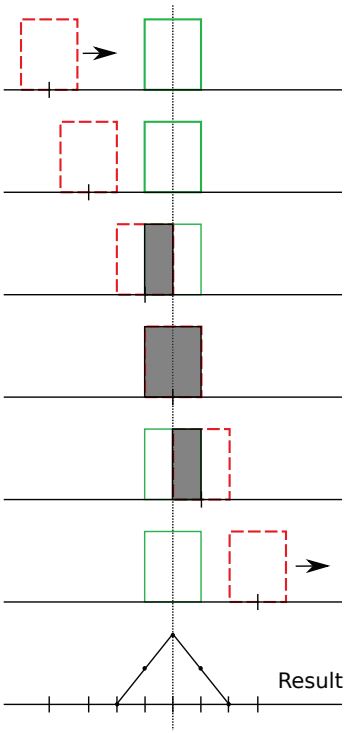
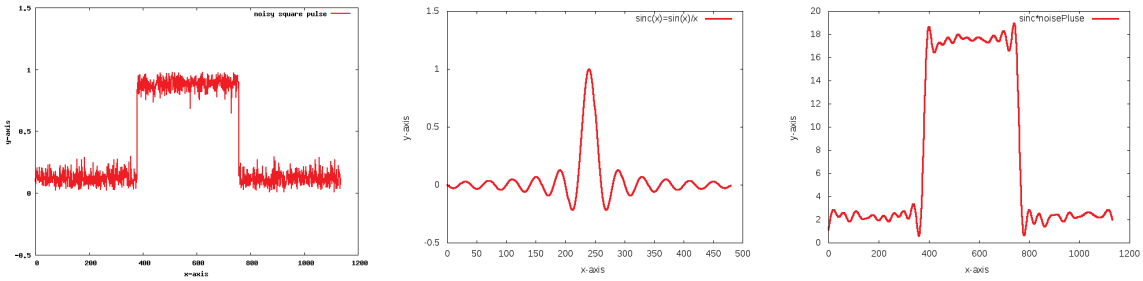


Figure 3.4: *Convolution of two rectangle pulses. The result is a triangle shown at the bottom line.*



(a) Input function f (b) Sinc function g (c) Convolution Output

Figure 3.5: *Convolving a noisy input function (a) with a Sinc function (b). (c) the result function of the convolution. Note how the perturbed data is reduced while maintaining the sharp discontinuities in the data.*

$$\text{sinc}(x) = \frac{\sin(2\pi f_c x)}{x\pi} \tag{3.3}$$

The sinc function blocks all frequencies above the cutoff frequency f_c and passes other frequencies below it. Figure 3.5 shows a noisy input signal (a) convolved with a sinc function (b). The result function (Figure 3.5(c)) shows a reduction in the high frequencies and the preservation of the sharp discontinuities (*i.e.* low frequencies).

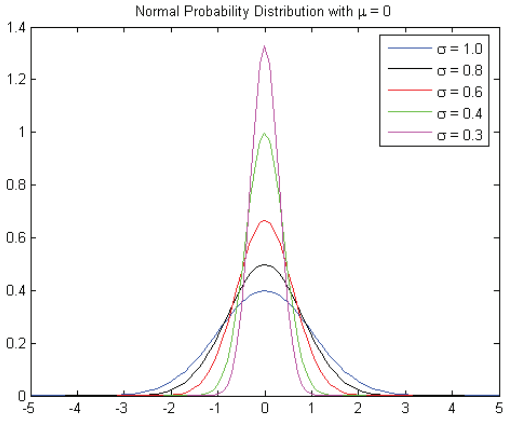


Figure 3.6: *Gaussian distribution with various values of* σ.

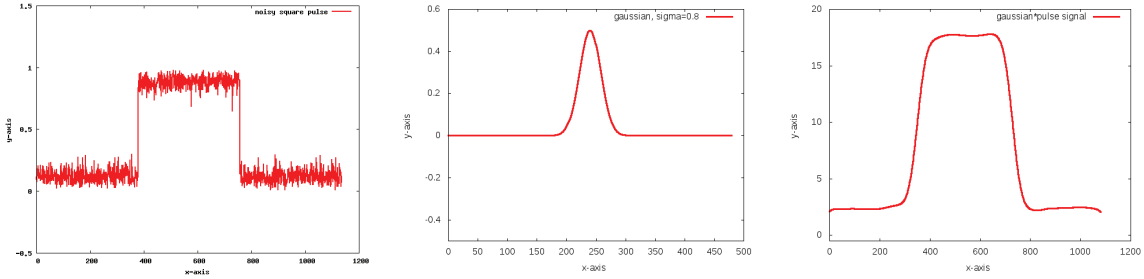
Another popular low-pass filter for removing noise from a signal is the *Gaussian* function. The Gaussian distribution has a symmetric bell shape where its width is defined by a parameter σ. Its mathematical equation is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \tag{3.4}$$

The Gaussian distribution describes a vast variety of physical and probabilistic phenomena. It emphasizes what happens on nearby points over more distant ones when centered at the point of interest. Moreover, its extremities fall as fast as possible to nearly zero making it a great low-pass filtering candidate. The width of the distribution, defined by σ, also defines the nearby sampling points that will contribute to the point of interest (in 2D, the σ variable represents an area). A small σ value results in a steep distribution covering a small region. As its value increases the distribution flattens covering a larger region (see Figure 3.6). Figure 3.7 shows a noisy input signal f (a) convolved with a Gaussian function with $\sigma = 0.8$ (b); the result is shown in (c).

3.3.2 Fourier Transform

Convolution is a great way to filter signals. However, its computation is expensive; roughly $O(n^2)$ where n is the number of sampling points in the signal. Its process can be accelerated by computing it in the frequency domain. This is done by first converting the signal to a frequency domain where its manipulation corresponds to attenuating the frequency coefficients. Thus, convolution is applied on the frequency coefficients. Finally, the filtered signal is then converted back to its original domain. This conversion is done using *Fourier transforms* [Bracewell 1999, Yoo 2001].



(a) Input function f (b) Gaussian function g ($\sigma^2 = 0.8$) (c) Convolution Output

Figure 3.7: *Convolving a noisy input function (a) with a Gaussian function (b). (c) the result function of the convolution. Note how the perturbed data is filtered away.*

The Fourier transform is composed of two components; *forward Fourier* and *inverse Fourier*. Forward Fourier converts a signal from its spatial domain to a frequency domain. Its mathematical definition in a continuous one-dimensional space is:

$$F(u) = \int_{-\infty}^{+\infty} f(x) e^{-i2\pi ux}\, dx \text{ where } u \text{ is the frequency variable} \tag{3.5}$$

$f(x)$ is the input signal and $F(u)$ are the coefficients of each sine and cosine. $F(u)$ is called the *spectrum* of the function $f(x)$ and contains the frequency information. The signal can be converted back; if it is invertible, to the spatial domain by applying the *inverse Fourier* transform:

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(u) e^{i2\pi ux}\, du \tag{3.6}$$

These two equations convert to and from the frequency domain using a simple sum of products. Notice that the only difference between the forward and inverse equation is the negation in the exponential. The term $\frac{1}{2\pi}$ in the inverse Fourier delimits the range domain to that of the input signal in its spatial domain.

The Fourier transform is involved in all sorts of applications which can be represented as signals such as television and radio transmission. It can identify periodic

components in a signal and as such can separate and block regular unwanted pattern. More importantly is its property with respect to convolution. In Fourier space, the convolution of two signals is equal to the product of their Fourier transforms:

$$F(f \otimes g) = F(f)F(g) \text{ where } f \text{ and } g \text{ are two functions.} \tag{3.7}$$

$F(f)$ and $F(g)$ are their respective Fourier transform. In particular, the Fourier transform of a Gaussian distribution is also a Gaussian distribution.

The discrete equations of the *forward Fourier* (Equation 3.5) and *inverse Fourier*(Equation 3.6) in one-dimensional are respectively:

$$F(u) = \sum_{x}^{N-1} f(x) e^{\frac{-i2\pi ux}{N}} \tag{3.8}$$

for $u = 0, 1, 2, ..., N-1$ and

$$f(x) = \frac{1}{N} \sum_{u=0}^{N-1} F(u) e^{\frac{i2\pi ux}{N}} \tag{3.9}$$

for $x = 0, 1, 2, ..., N-1$. The *discrete Fourier transform* is very easy to compute on the computer but is of complexity $O(N^2)$.

The fast Fourier Transform (FFT) [Brigham 1974, Brigham 1988] efficiently computes the discrete Fourier transform and its inverse. FFT decomposes the components of the Equations 3.8 and 3.9 to reduce the complexity from $O(N^2)$ to $O(N \log_2 N)$.

3.3.3 Bilateral Filter

Bilateral Filtering [Tomasi 1998] is a non-linear filtering method that preserves the sharp changes in a signal. It is also considered as a low-pass filter. It extends the Gaussian concept by taking into account both the *closeness* and *similarity* of neighboring sampling points to that of the central point. More weight is put on points that are close to the central point; either in distance or in value. A point with a value very different from that of the central point contributes less to the overall averaging value even if it is spatially near the central point. Mathematically, bilateral filtering is composed of two Gaussian filters: one that weights in the spatial domain known as the *domain filter* and the other weights in the value domain known as the *range filter*. Its continuous definition in one-dimension is:

$$h(x) = k(x)^{-1} \int_{-\infty}^{+\infty} f(u)c(u,x)s(f(u), f(x))du \tag{3.10}$$

with the normalization

$$k(x) = \int_{-\infty}^{+\infty} c(u,x)s(f(u), f(x))du$$

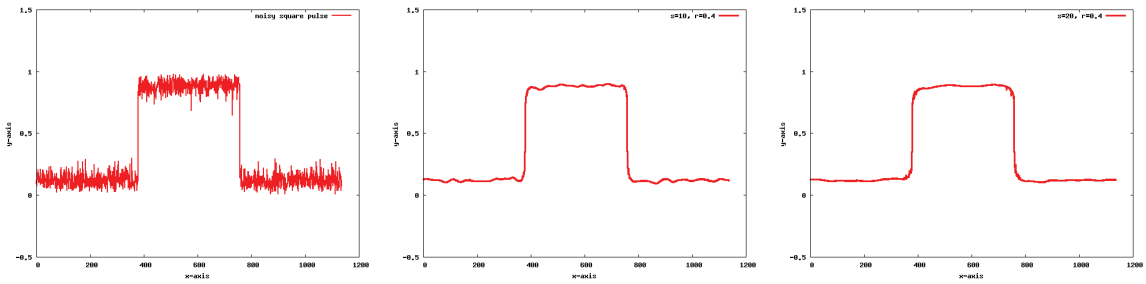
The output and input functions are h and f respectively. x is the central point over which the bilateral filter is computed and u is a neighbor point of x in f. The closeness function $c(u, x)$ and the similarity function $s(f(u), f(x))$ are Gaussian functions of the Euclidean distance between their arguments:

$$c(u, x) = \frac{1}{\sigma_d \sqrt{2\pi}} e^{-\frac{\|\|u-x\|\|}{2\sigma_d^2}}$$

and

$$s(f(u), f(x)) = \frac{1}{\sigma_r \sqrt{2\pi}} e^{-\frac{\|\|f(u)-f(x)\|\|}{2\sigma_r^2}}$$

The geometric spread σ_d defines the bandwidth of the filter. A large σ_d will combine values of more distant points. Similarly, the range spread σ_r defines the desired value of points to be combined. Sampling points with value difference smaller than or equal to σ_r are included in the overall averaging. Larger ones are excluded.



(a) Input function f (b) BF: $\sigma_d = 10$, $\sigma_r = 0.4$ (c) BF: $\sigma_d = 20$, $\sigma_r = 0.4$

Figure 3.8: *An input signal with noise (a) is filtered twice by the bilateral filter with different parameters (b) and (c). Notice (c) is smoother as the region considered is double from that of (b);* $\sigma_d = 20$ *and* $\sigma_d = 10$ *respectively. Sharp discontinuities are preserved.*

Figure 3.8 shows a noisy input signal filtered twice by bilateral filtering. The rapid perturbation data in Figure 3.8(a) is attenuated in (b) while the sharp discontinuities remain. The large spatial domain chosen in (c) suppresses the high frequencies by involving distant points in the overall averaging value.

In discrete space, bilateral filtering is expressed by the sum of the product of the two Gaussian functions and the value of the neighboring point:

$$h(x) = k(x)^{-1} \sum_{u=0}^{N-1} f(u)c(u, x)s(f(u), f(x)) \tag{3.11}$$

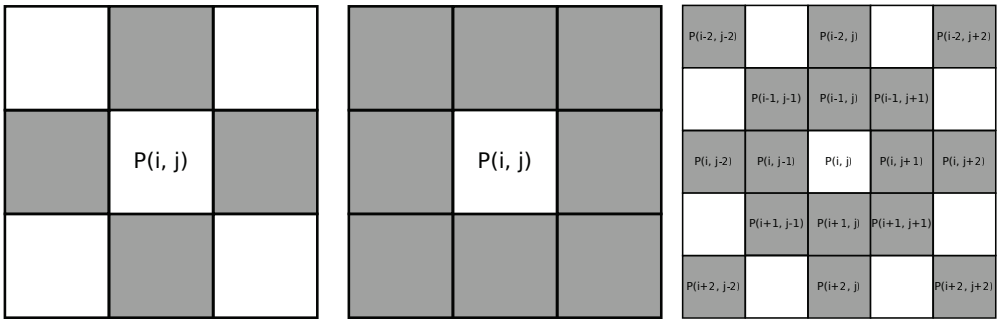
where N is the number of sampling points of input function f with

$$k(x) = \sum_{u=0}^{N-1} c(u,x)s(f(u), f(x))$$

3.4 Smoothing Images

Image acquisition is influenced by noise caused by different factors such as low-light illumination and electronics defects. The challenging goal is removing this noise while preserving the details and features defining the image. Many techniques and algorithms were devised and evolved over the years to address this issue. There is still no ideal solution for simultaneously smoothing and preserving details of the images. In this thesis, we will present the most well known techniques relative to our contributions.

A digital image is discretized into a two-dimensional grid of intensity value pixels defining the content of the image. Detail features, or edges, in an image define the characteristics of the subject of the image. They can be identified by the high difference in intensity values or gradients between neighboring pixels. Smoothing images results in modifying the pixels intensity values.



(a) (b) (c)

Figure 3.9: *Given the center pixel $P(i,j)$. (a) 4 connected neighbor pixels: considers only the non-diagonal pixels, (b) 8 connected neighbor pixels: considers all surrounding pixels, and (c) 16 connected neighbor pixels: including the 8 directly connected pixels, it also considers 8 pixels from the second ring of pixels away from $P(i,j)$.*

3.4.1 Statistical Methods

The intensity value of pixels can be attenuated using statistical techniques. Statistical quantities are computed over a defined neighborhood of the pixel being modified. Mean-filtering [Gonzalez 2001] is a method that replaces the value of each pixel by the mean of its surrounding neighbor pixels. Median-filtering replaces the value of each pixel by the median intensity value of its local neighborhood

(see [Tukey 1977, Narendra 1981, Huang 1979, Gallagher 1981]). These methods do smooth images, however they blur out the details of the characteristics of the images. Davis and Rosenfeld [Davis 1978] proposed the K-means method to preserve details. In a local neighborhood, the replacing mean value is computed on selective pixels that are bounded by precise criteria. It selects K neighbors pixels closest in pixel intensity value. The method gives image dependent neighborhoods, but they are not symmetric and require more computation.

3.4.2 Weighted Average

Smoothing images can also be achieved by associating weights to neighboring pixels before replacing the center pixel by their weighted average. The weights are determined from the local information of pixels; such as gradient [Wang 1981], second derivatives [Graham 1962] or local mean and variance information [Lee 1980, Lee 1981], and generally from a 4-8-16 connected neighbor pixels. Figure 3.9 highlights in grey the pixels considered to each of the pixel neighboring scheme. Weighting pixels was designed with the objective of preserving edges in the image by emphasizing pixels over others. However, most of the time they fail to do so. Okada [Okada 1985] weights the intensity value of neighboring pixels on a window kernel of 3 x 3 and 5 x 5 (see Figure 3.10) and proposes three algorithms:

- Algorithm 1: Given a pixel $P(i,j)$ in a 3x3 pixels area (Figure 3.10(a)), replaces its value by the following:

  $$P_{i,j} = (P_{i-1,j} + P_{i,j-1} + P_{i+1,j} + P_{i,j+1})/4$$

  if this condition is satisfied.

  $$\begin{aligned}(|P_{i-1,j} - P_{i,j}| < k) \wedge (|P_{i,j-1} - P_{i,j}| < k) \wedge \\ (|P_{i+1,j} - P_{i,j}| < k) \wedge (|P_{i,j+1} - P_{i,j}| < k)\end{aligned}$$

  where k is constant and determined empirically. It is chosen to exclude boundary between two regions as the difference between their intensity value is larger than k.

- Algorithm 2: In a 5x5 pixels area (Figure 3.10(b)), the value of pixel $P(i,j)$ is replaced by

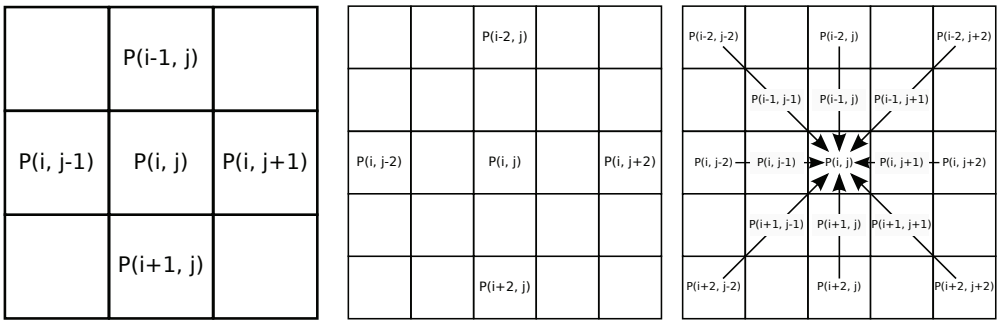
  $$P_{i,j} = (P_{i-2,j} + P_{i,j-2} + P_{i+2,j} + P_{i,j+2})/4$$

  if the condition

$$(|P_{i-2,j} - P_{i,j}| < k) \wedge (|P_{i,j-2} - P_{i,j}| < k) \wedge$$
$$(|P_{i+2,j} - P_{i,j}| < k) \wedge (|P_{i,j+2} - P_{i,j}| < k)$$

is satisfied.

- Algorithm 3: first applies algorithm 2 and then compute averages of eight pairs of pixels $\{P(i-2,j-2), P(i-1,j-1)\}$, $\{P(i-2,j), P(i-1,j)\}$,*etc.* as shown in Figure 3.10(c) by the arrows. Out of the eight pairs, it finds the pair with its average closest to $P(i,j)$. The value of $P(i,j)$ is then replaced by the mean value computed from the three pixels, the pair pixels, and $P(i,j)$.



(a) 3x3 boxes (b) 5x5 boxes alg2 (c) 5x5 boxes alg3

Figure 3.10: *Okada's [Okada 1985] different selection of pixels for the weighted averaging algorithms.*

Boult and Melter [Boult 1993] generalize Okada's concept by proposing G-neighbors. The idea is simple. One can chose any smoothing technique. However, its application is done only on pixels where their difference satisfies a predicate. This selected group of pixels is called G-neighbors. Two pixels with values A and B are considered in G-neighbors if $|A - B| < G$. The value of G has an impact on either tightening or loosening the requirement for pixels to be G-neighbors. A low G value will tighten up the requirement and a higher one will loosen it up. Figure 3.11 shows a grid of pixels with their intensity value. Pixel connected with dash lines are G-neighors with $G = 30$.

3.4.3 Convolution in 2D

Images can be filtered by convolving them with a 2D mask function. The 2D mask is centered at each pixel. Then, the value of each pixel is modified by computing a weighted sum of the product of the mask and local neighboring pixels of the centered

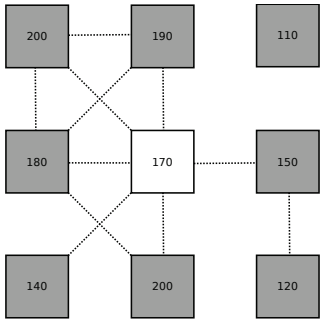


Figure 3.11: *The number inside each pixel represents the intensity value for that pixel. Pixels that are G-neighbors when G=30 are shown with dashed lines. (figure taken from G-Neighbor article)*

---

pixel. Convolution has a significant role in image processing. The concept of convolution in images is simply an addition of an extra dimension to its one-dimensional space presented in Section 3.3. The mathematical equation of a continuous 2D convolution is defined by the integrals over both dimensions of the product of the input data and filter function:

$$h(u,v) = (f \otimes g)(u,v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x,y)g(u-x, v-y)\, dxdy \tag{3.12}$$

where f is the 2D input data, g is the filter function, and $\otimes$ denotes the convolution operation. Note again that f is multiplied by a time-shifted g.

In discrete space, each output value of a convolution is the sum of the product of the two functions over a delimit defined range in both dimensions. The discrete convolution equation is:

$$h[m,n] = (f \otimes g)[m,n] = \sum_{i=-\omega}^{i=\omega} \sum_{j=-\omega}^{j=\omega} f[i,j]g[m-i, n-j] \tag{3.13}$$

where $-\omega$ and ω define the domain ranges of f.

Convolving an image with a filter function can either blur or denoise it. The Gaussian distribution is a well known smoothing operator. It filters out high frequencies and keeps low frequencies. The Gaussian distribution in two-dimension is simply the multiplication of two Gaussian functions each representing a dimension:

$$g(x,y) = g(x)g(y) = (\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}})(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{y^2}{2\sigma^2}}) = \frac{1}{\sigma^2 2\pi} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.14}$$

The σ parameter defines the area over the sampling pixels which will contribute

(a) Noisy Lena image (b) Gaussian blur $\sigma = 2.0$ (c) Gaussian blur $\sigma = 8.0$

Figure 3.12: *Smoothing a noisy Lena image(a) by convolving with the Gaussian function of* $\sigma = 2.0$ *(b) and* $\sigma = 8.0$*. As the* σ *value increases the image is blurrier and the edges fade away.*

to the overall weighted average of the central pixel. Figure 3.12 shows a noisy Lena image smoothed by convolving it with a 2D Gaussian function of σ values 2.0 and 8.0. The image is much more smoother as the value of σ increases. Moreover, the detail features such as the edges are blurred away which is a disadvantage of Gaussian smoothing. It does not differentiate between pixels that should be attenuated and those that shouldn't. It only favors nearby pixels over those farther away.

Two-dimensional Gaussian function is separable into two one-dimensional factors: $g(x, y) = g(x)g(y)$. This is computationally important, it reduces the cost of 2D convolution by separating it into two one-dimensional convolutions. Equation 3.13 becomes:

$$h[m, n] = (f \otimes g)[m, n] = \sum_{x=-\omega}^{x=\omega} g[m - x] \sum_{y=-\omega}^{y=\omega} f[x, y]g[n - y] \tag{3.15}$$

reducing its cost from $O(2\omega^2)$ to $O(2\omega)$ respectively.

The concept of convolution can be generalized to higher dimensions, simply by including the extra dimensions in the formulation. Convolution is very expensive to compute, and including extra dimensions increases the computational cost linearly. However, this process can be accelerated using the Fourier transforms.

3.4.4 Fourier Transform in 2D

Similarly as in one-dimensional, processing images in the frequency domain is simpler than in the spatial domain. In frequency space, periodic signal components of an image can be identified, then simple measurement can be applied to them that would be difficult to do in spatial domain. Convolution is a simple multiplication in frequency space. Images can be converted to and from the frequency domain by applying two-dimensional Fourier transforms.

Fourier transforms can be extended from one-dimensional to n-dimensional space. This is done by including the extra dimensions in both the *forward* and *inverse Fourier* equations (Equations 3.5 and 3.6) through a set of n nested integrals of one-dimensional Fourier transform:

$$F(u,v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x,y) e^{-i2\pi(ux+vy)}\, dxdy \tag{3.16}$$

where u and v are the frequency variables. $f(x,y)$ is the input 2D image and $F(u,v)$ is the *spectrum* of $f(x,y)$. The continuous *inverse Fourier* is:

$$f(x,y) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} F(u,v) e^{i2\pi(ux+vy)}\, dudv. \tag{3.17}$$

The discrete equations of the two-dimensional *forward Fourier* and *inverse Fourier* are respectively:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-i2\pi(\frac{ux}{M}+\frac{vy}{N})} \tag{3.18}$$

for $u = 0, 1, 2, ..., M-1$ and $v = 0, 1, 2, ..., N-1$ defining the dimensions of the 2D signal, and

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{i2\pi(\frac{ux}{M}+\frac{vy}{N})} \tag{3.19}$$

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

As mentioned earlier, discrete Fourier transform is ideal for digital computations. We use the Fast Fourier transform (FFT) [Brigham 1974, Brigham 1988] algorithm as it reduces significantly the conversion time to and from the frequency domain. Furthermore, it can be configured and extended to any number of dimension. Figure 3.13 displays the FFT of the Lena image.

Filtering images in the frequency domain can decrease the sharpness of the edges as high frequency components are also around the edges. Edges themselves could be represented by high frequency. Thus, the images loses some of its sharp details.

3.4.5 Bilateral Filter in 2D

Smoothing images while preserving their edges can be accomplished using the non-linear bilateral filter [Paris 2007, Tomasi 1998, Paris 2008, Paris 2009]. A filtered image is obtained by replacing the intensity value of each pixel with a Gaussian average value weighted by the geometric and photometric similarities between neighboring pixels within a spatial window [Tomasi 1998]. Bilateral filtering, presented in Section 3.3.3 on one-dimensional signal, can be extended to and applied on n-dimensional signals. This is achieved through a set of n nested summation of one-dimensional bilateral filter:

(a) Lena image (b) Fourier space of (a)

Figure 3.13: *(b) displays the frequency data of the Lena image(a) after converting it with FFT.*

$$h(x,y) = k(x,y)^{-1} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(u,v)c(p_{uv},p_{xy})s(f(u,v),f(x,y))dudv \qquad (3.20)$$

with the normalization

$$k(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} c(p_{uv},p_{xy})s(f(u,v),f(x,y))dudv.$$

The output and input functions are h and f respectively. (x,y) is the central pixel over which we compute the bilateral filter and (u,v) is a neighbor pixel of (x,y) in f. p_{uv} and p_{xy} are respectively the pixels position of (u,v) and (x,y) respectively. The closeness function $c(p_{uv},p_{xy})$ and the similarity function $s(f(u,v),f(x,y))$ are Gaussian functions of the Euclidean distance between their arguments:

$$c(p_{uv},p_{xy}) = \frac{1}{\sigma_d\sqrt{2\pi}} e^{-\frac{\|p_{uv}-p_{xy}\|}{2\sigma_d^2}}$$

and

$$s(f(u,v),f(x,y)) = \frac{1}{\sigma_r\sqrt{2\pi}} e^{-\frac{\|f(u,v)-f(x,y)\|}{2\sigma_r^2}}$$

The geometric spread σ_d defines the spatial window of the filter. A large σ_d will combine values of more distant pixels. Similarly, the photometric spread σ_r defines the desired intensity value of pixels to be combined. Pixels with value difference smaller than σ_r are included in the overall averaging. Larger ones are excluded.

The discrete two-dimensional function of bilateral filtering is

$$h(x,y) = k(x,y)^{-1} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n)c(p_{mn}, p_{xy})s(f(m,n), f(x,y)) \tag{3.21}$$

where M and N define the width and height of the input image f with

$$k(x,y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} c(p_{mn}, p_{xy})s(f(m,n), f(x,y))$$

(a) Noisy Lena image (b) BF: $\sigma_d = 2, \sigma_r = 0.35$ (c) BF: $\sigma_d = 10, \sigma_r = 0.35$

Figure 3.14: *(a) A noisy image of Lena. (b) The image of Lena is smoothed by the bilateral filtering. Most of the noise is filtered away and the edges are preserved. (c) Edges start to fade away as the geometric spread σ_d increases and farther away pixels are considered.*

Figure 3.14 shows a noisy image of Lena smoothed by the bilateral filter with different values of σ_d and σ_r. Note that with $\sigma_d = 2$ and $\sigma_r = 0.35$ the image is smoothed while the edges are preserved. Increasing the value of σ_d (Figure 3.14(c)) blurs away the edges as the filter covers more distant pixels.

Bilateral filter is a great mean for denoising images. However, tuning its parameters, σ_d and σ_r, might be a source of problems as their selection is dependent on a trial and error approach. Moreover, tuning their values to keep the edges and features of the image also keep some of the noise. If we push the parameter until the noise disappear, we start losing the features and details.

3.5 Smoothing Images using PDE

Partial differential equation (PDE) is an alternative technique to remove noise from images. This approach processes images in the temporal domain. It exploits the concept of heat diffusion where energy is dispersed from areas of high concentration to areas of lower concentration. In images, this is accomplished iteratively: at each

time step, the intensity value of each pixel is attenuated by a fraction of an average value computed over its local neighboring pixels. Mathematically, it is defined as

$$I(x, y, t + \Delta t) = I(x, y, t) + \Delta t(I_t) \tag{3.22}$$

where $I(x, y, t)$ is an image at time t. The original image is found at $t = 0$. Δt is the time step and $I(x, y, t+\Delta t)$ is the attenuated image after one time step. Finally, I_t is the average computed value. I_t can be computed for various functionalities, and it can be either linear or non-linear. Linear diffusion, or isotropic diffusion, carries diffusion equally in all direction. Non-linear diffusion, or anisotropic diffusion, controls the orientation of diffusion. Although the PDE technique of smoothing is presented here on two-dimensional images, it can be generalized to any number of dimensions. For notation clarification, we use I_x and I_{xx} to express the first and second derivatives of I respectively; *i.e.* $I_x = \frac{\partial I}{\partial x}$, $I_t = \frac{\partial I}{\partial t}$, $I_{xx} = \frac{\partial I_x}{\partial x}$, *etc.*

3.5.1 Linear Heat Diffusion

Linear heat equation (or linear heat diffusion) is a gradient descent PDE. It takes advantage of the gradient (I_x,I_y) information of the image to equilibrate the intensity values through out the image. More appropriately, it is defined as:

$$I_t = I_{xx} + I_{yy} = \Delta I \tag{3.23}$$

where Δ is the *Laplacian* operator. The second derivative of a pixel in each dimension is computed and summed up. A fraction of this value, depending on the choice of Δt, is added to the pixel at time t (Equation 3.22).

Linear heat diffusion is isotropic: diffusion is independent from orientation. In images, high intensity values are diffused in all directions. This blurs away the edges and characteristics of the image. Moreover, they are dislocated from their original position. Figure 3.15 shows an image smoothed by the linear heat diffusion. The sharpness of the edges have gradually diminished as the number of diffusion steps increases.

Applying heat diffusion several times converges to Gaussian smoothing. However, unlike Gaussian smoothing (executed in one step) heat diffusion iterative approach gives the user control over the smoothing. Diffusion can be aborted at any iteration step once the smoothing has reached a satisfying result.

3.5.2 Non-linear Heat Diffusion

In this section, we review the different non-linear heat diffusion schemes.

3.5.2.1 Geometric Heat Diffusion

Geometric diffusion diffuses only parallel to edges. This reduces the effect of blurring edges caused by isotropic diffusion. Laplace operator is flexible: it produces the same result regardless of the coordinates used to obtain ΔI. A coordinate for instance

(a) Noisy Lena image (b) $d = 15$ (c) $d = 30$

Figure 3.15: *A noisy image of Lena (a) smoothed by linear heat diffusion at diffusion steps: (b)* 15 *and (c)* 30*. Diffusion is spread uniformly in all direction. The intensity values are spread over all the image as more diffusion steps is applied.*

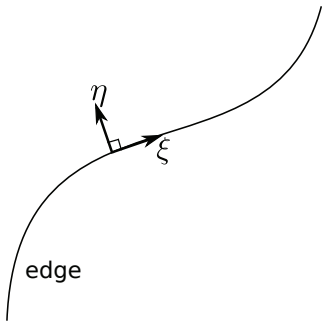


Figure 3.16: *$\xi\eta$ coordinate is a flexible coordinate system used as a typical xy coordinate.*

representing an edge and its orthogonal orientation can be used in the same way as a typical xy coordinate. Figure 3.16 illustrates this where ξ and η represent respectively the directions tangent to the edge and normal to the edge.

This generalizes Equation 3.23 into $I_t = I_{xx} + I_{yy} = I_{\xi\xi} + I_{\eta\eta} = \Delta I$. Hence, making $I_{\eta\eta} = 0$ restricts a diffusion that is tangent to the edge. Then Equation 3.23 becomes:

$$I_t = I_{\xi\xi} + I_{\eta\eta} - I_{\eta\eta} = \Delta I - I_{\eta\eta} \tag{3.24}$$

This can be further generalized by letting η be the gradient; the direction the image intensity increases. The gradient (∇I) is always orthogonal to the edge. Thus, $\eta = \frac{\nabla I}{\|\nabla I\|}$ and Equation 3.24 becomes:

Differential Terms	Difference Equations
$I_x(x,y)$	$I_x(x,y) = \frac{I(x+1,y)-I(x-1,y)}{2}$
$I_y(x,y)$	$I_y(x,y) = \frac{I(x,y+1)-I(x,y-1)}{2}$
$I_{xx}(x,y)$	$I_{xx}(x,y) = I(x+1,y) - 2I(x,y) + I(x-1,y)$
$I_{yy}(x,y)$	$I_{yy}(x,y) = I(x,y+1) - 2I(x,y) + I(x,y-1)$
$I_{xy}(x,y)$	$I_{xy}(x,y) = \frac{I(x+1,y+1)-I(x+1,y-1)+I(x-1,y-1)-I(x-1,y+1)}{4}$

Table 3.1: *Lists of spatial differences for first and second order derivatives.*

$$I_t = \Delta I - \left(\frac{\nabla I}{\|\nabla I\|}\right)^T (\nabla^2 I) \left(\frac{\nabla I}{\|\nabla I\|}\right) \tag{3.25}$$

where $\nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix}$ and $\nabla^2 I = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$ is the Hessian. By simple substitution, we get

$$I_t = I_{xx} + I_{yy} - \frac{\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix}}{I_x^2 + I_y^2}$$

which simplifies to the *geometric* heat equation:

$$I_t = \frac{I_x^2 I_{yy} - 2I_x I_y I_{xy} + I_y^2 I_{xx}}{I_x^2 + I_y^2} \tag{3.26}$$

$\frac{\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix}}{I_x^2+I_y^2}$ computes the contribution of the diffusion in the $\begin{bmatrix} I_x \\ I_y \end{bmatrix}$ direction. Excluding this value from ΔI makes the diffusion uniquely tangent to the edge.

Figure 3.17 illustrates geometric heat diffusion. The edges are well preserved even when the diffusion steps is increased. Only the interior regions are smoothed.

(a) Lena with noise (b) $d = 15$ (c) $d = 60$

Figure 3.17: *A noisy image of Lena(a) smoothed by geometric heat diffusion at diffusion steps: (b)* 15 *and (c)* 60*. Notice the edges are well preserved even as the number of diffusion steps increases. This can be seen in (c). The interior regions are blurred but the edges are preserved.*

At higher diffusion steps (Figure 3.17(c)), the image tends to give the impression of a water painting image.

3.5.2.2 Perona and Malik Diffusion

Scale-space heat diffusion smoothes images at different resolution levels by introducing a constant value C in Equation 3.23; *i.e.* $I_t = C\Delta I$. Different values of C define different scale levels at which the image will be smoothed. However, such a configuration has a cost, as it blurs away the edges of the image.

Motivated by the concept of scale-space [Witkin 1983] and its limitation, Perona and Malik [Perona 1990] proposed a scale-space anisotropic diffusion. They argue that C does not need to be constant. They proposed to replace it by a function that takes as parameter the magnitude of the gradient; *i.e.* $C(\|\nabla I\|)$. Injecting this function in Equation 3.23 gives:

$$\begin{aligned} I_t &= \frac{\partial}{\partial x}\left(\frac{\partial C(\|\nabla I\|)}{\partial I_x}\right) + \frac{\partial}{\partial y}\left(\frac{\partial C(\|\nabla I\|)}{\partial I_y}\right) \\ &= \frac{\partial}{\partial x}\left(\frac{\partial C(\|\nabla I\|)}{\partial \|\nabla I\|}\frac{\partial \|\nabla I\|}{\partial I_x}\right) + \frac{\partial}{\partial y}\left(\frac{\partial C(\|\nabla I\|)}{\partial \|\nabla I\|}\frac{\partial \|\nabla I\|}{\partial I_y}\right) \\ &= \frac{\partial}{\partial x}\left(\frac{\partial C(\|\nabla I\|)}{\partial \|\nabla I\|}I_x\right) + \frac{\partial}{\partial y}\left(\frac{\partial C(\|\nabla I\|)}{\partial \|\nabla I\|}I_y\right) \end{aligned}$$

By substituting the math notation, this gives the Perona-Malik diffusion PDE:

$$I_t = div\left(C'(\|\nabla I\|)\frac{\nabla I}{\|\nabla I\|}\right) \tag{3.27}$$

The diffusivity function $C(\|\nabla I\|)$ regulates how diffusion is processed. It favors diffusion along the edges rather than across them; *i.e.* diffusion is done inside the regions. They proposed monotonically decreasing functions and suggest the following two types:

- Type 1: $C(\|\nabla I\|) = \frac{K^2}{2} ln\left(1 + \left(\frac{\|\nabla I\|}{K}\right)^2\right)$ which after substituting and evaluating its derivative gives:

$$I_t = div\left(\frac{1}{1 + (\|\nabla I\|/K)^2}\nabla I\right) \tag{3.28}$$

- Type 2: with $C(\|\nabla I\|) = -\frac{K^2}{2} e^{-\left(\frac{\|\nabla I\|}{K}\right)^2}$ which similarly after substituting and evaluating its derivative gives:

$$I_t = div\left(e^{-\left(\frac{\|\nabla I\|}{K}\right)^2}\nabla I\right) \tag{3.29}$$

Notice that the constant K in both equations acts as the scale level or contrast parameter to which the image is smoothed. A gradient of magnitude lower than K converges the function to a unit value, allowing diffusion in the gradient direction. Inversely, a gradient of magnitude higher than K converges the function to zero, blocking the diffusion in this particular gradient direction. Thus, as $\|\nabla I\|$ increases the function value decreases and the opposite is also true. The value of K is quite a guess work to achieve an ideal smoothing with edge preservation. Yet, different K estimators were proposed by practical application found in [Hy 2006, Canny 1986, Black 1998].

Although the Perona-Malik diffusion preserves edges, it is ill-posed [Perona 1990, Weickert 1998], due to its strict binary diffusion condition; diffuse parallel to edges and none across. Noise around edges remains. In addition, these edges are enhanced temporarily before being blurred away as the number of diffusion steps increases.

Figure 3.18 shows images smoothed with type 1 and type 2 of Perona and Malik scheme. Although the edges are preserved, noise can be seen around them. Applying a couple of extra diffusion steps above the satisfied number of diffusion steps acts like a Gaussian smoothing blurring away the enhanced edges (Figure 3.19(a)). Moreover, increasing the threshold value K considers more distant pixels which also blurs away the edges (Figure 3.19(b)).

The Perona and Malik algorithm can be numerically expressed as:

$$I(x, y, t + \Delta t) = I(x, y, t) + \Delta t[f_N D_N I + f_E D_E I + f_W D_W I + f_S D_S I]_{(x,y,t)} \tag{3.30}$$

where

Equations	Gradient Descent PDEs
Linear heat equation	$I_t = I_{xx} + I_{yy}$
Geometric heat equation	$I_t = \frac{I_x^2 I_{yy} - 2I_x I_y I_{xy} + I_y^2 I_{xx}}{I_x^2 + I_y^2}$
Perona-Malik diffusion equation: Type 1	$I_t = div\left(\frac{1}{1+(\|\nabla I\|}\nabla I\right)$
Perona-Malik diffusion equation: Type 2	$I_t = div\left(e^{-\left(\frac{\|\nabla I\|}{K}\right)^2}\nabla I\right)$

Table 3.2: *The partial differential equations of the linear, geometric, Perona-Malik diffusion [Perona 1990].*

$$\begin{aligned} D_N I(x,y) &= I(x,y-1) - I(x,y) \\ D_S I(x,y) &= I(x,y+1) - I(x,y) \\ D_E I(x,y) &= I(x+1,y) - I(x,y) \\ D_W I(x,y) &= I(x-1,y) - I(x,y) \end{aligned}$$

are the intensity differences between the center pixel $I(x,y)$ and its four non-diagonal neighbor pixels and

$$\begin{aligned} f_N &= f(D_N I(x,y)) \\ f_S &= f(D_S I(x,y)) \\ f_E &= f(D_E I(x,y)) \\ f_W &= f(D_W I(x,y)) \end{aligned}$$

are the results of the tensor diffusion for each of $D_N I$, $D_S I$, $D_E I$, and $D_W I$.

3.5.3 Diffusion and Time Step Δt

Table 3.2 summarizes the PDE of the linear, geometric, and Perona-Malik diffusion scheme. Table 3.3 summarizes their numerical schemes. The time step Δt value is chosen in such a way that the diffusion is stable. Table 3.3 gives the limit values of

Diffusion Schemes	Numerical Schemes
Linear heat diffusion	$I(x,y,t+\Delta t) = I(x,y,t) + \Delta t(I_{xx} + I_{yy})$
Geometric heat diffusion	$I(x,y,t+\Delta t) = I(x,y,t) + \Delta t \left(\frac{I_x^2 I_{yy} - 2I_x I_y I_{xy} + I_y^2 I_{xx}}{I_x^2 + I_y^2}\right)$
Perona-Malik diffusion	$I(x,y,t+\Delta t) = I(x,y,t) + \Delta t\,(f(\|\nabla I\|)\nabla I)$

Table 3.3: *A list of equations for the various diffusion schemes.* $f(\|\nabla I\|)$ *is an increasing function.*

Diffusion Schemes	Time Step Condition
Linear heat diffusion	$0 \leq \Delta t \leq \frac{1}{4}$
Geometric heat diffusion	$0 \leq \Delta t \leq \frac{1}{2}$
Perona-Malik diffusion	$0 \leq \Delta t \leq \frac{1}{4}$

Table 3.4: *List of the time step condition values for the various diffusion schemes.*

Δt for the various diffusion schemes discussed.

3.5.4 Anisotropic Diffusion Tensor

Since its introduction, the Perona-Malik algorithm has inspired researchers to propose numerous nonlinear diffusion filters [Weickert 1998, Romeny 1994]. Most of them introduce new or modified edge-stopping functions to remedy the limits of the Perona-Malik algorithm and adapt to different applications. The proposed methods revolve around applying spatial regularization, temporal regularization, and manipulation of the gradient [Black 1998, Mayer 2007, Bajla 1993]. For instance, [Mayer 2007] computes a new edge-stopping function based on two variables; correlation and noise variance, from two input images. [Bajla 1993] computes a function based on a histogram of gradients. A histogram for each gradient in the image is computed. Gradient of frequency higher than a computed threshold is considered as an interior region encouraging smoothing. However, gradient of frequency less than the threshold defines a region of edges hereby reducing the smoothing.

The Perona-Malik algorithm and its derivatives use either scalar-diffusivity or matrix functions that adapt to the underlying image structure. Thus, only the magnitude of the gradient flux can be controlled rather than the direction in which to diffuse. This limits the efficiency in reducing noise especially near edges while

(a) Noisy Lena image (b) PM type 1 (c) PM type 2

Figure 3.18: *Smoothing with Perona-Malik diffusion: (a) A noisy Lena image. (b) The image of Lena smoothed by type 1 of Perona-Malik diffusion. (c) The image of Lena smoothed by type 2 of Perona-Malik diffusion. Both (b) and (c) were smoothed with* 9 *diffusion steps and* $K = 0.1$. *Edges are well preserved but one can notice noise around edges.*

(a) $d = 15$ $k = 0.1$ (b) $d = 9$ $k = 0.9$ (c) $d = 15$ $k = 0.9$

Figure 3.19: *Images are smoothed with Perona-Maliks' type* 1 *diffusion. Details of the image can be blurred away for multiple factors. (a) Applying more diffusion steps, (b) increasing the value K includes large gradients (i.e. edges) in the over all average, and (c) increasing both these values.*

preserving the detail structures. Diffusion tensors provide control on the mobility of diffusion. Through it, diffusion can be steered in the necessary orientation. Weickert [Weickert 1994, Weickert 1997, Weickert 1999b] proposed the following diffusion tensor J_ρ:

$$J_\rho(\nabla I_\sigma) = K_\rho \otimes J_0 \tag{3.31}$$

where $J_0 = \nabla I_\sigma \nabla {I_\sigma}^T$. Image I is first smoothed by the Gaussian K_σ with scale size σ; $I_\sigma = K_\sigma \otimes I$. The tensor product $(\nabla I_\sigma \nabla {I_\sigma}^T)$ makes the structure

favor orientation rather than direction. Thus, opposite directions do not cancel each other. The orientation can be averaged over a neighboring size of $O(\rho)$ by applying a component-wise convolution with a Gaussian K_ρ. J_ρ is a two-dimensional symmetric positive semidefinite matrix with an orthonormal basis of eigenvectors v_1 and v_2 along their corresponding eigenvalues $\mu_1 \geq \mu_2 \geq 0$. The eigenvalues measure the average of contrast in the eigenvector direction within the scale ρ. v_1 corresponds to the orientation with the highest fluctuation; *i.e.* parallel to the average gradient orientation. v_2 corresponds to the orientation with the lowest fluctuation. Thus, if we want to remove noise from images while keeping the edges, diffusion should be done along v_2. The orientation of the diffusion is controlled by λ_1 and λ_2 in the matrix:

$$G = \begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} v_1 & v_2 \end{bmatrix}^T \tag{3.32}$$

where v_1 and v_2 are the eigenvectors of the J_ρ. G is positioned in the classical diffusion equation $I_t = div(G\nabla I)$. Choosing $\lambda_1 > \lambda_2$ produces a diffusion favorable in the orientation of v_1. The case is true for $\lambda_2 > \lambda_1$ where diffusion is done in the orientation of v_2. If $\lambda_1 = \lambda_2$, this results in an isotropic diffusion.

3.5.4.1 Edge-Enhancing Diffusion and Coherence-Enhancing Diffusion

Weickert proposed two schemes, edge-enhancing diffusion (EED) and coherence-enhancing diffusion (CED), for the use of the diffusion tensor depending on the features of the image [Weickert 1994, Weickert 1997, Weickert 1999b, Weickert 1999a]. EED enhances and sharpens the edges by smoothing along them. Since, μ_1 represents the highest gradient orientation; *i.e.* an edge, λ_1 and λ_2 are chosen based on the Perona-Malik model and set as:

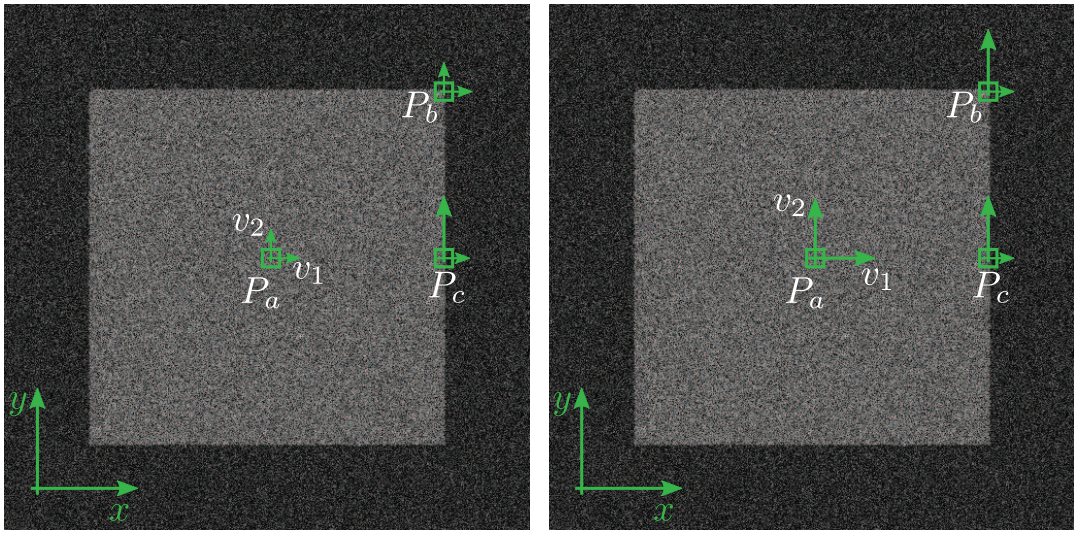
$$\begin{aligned} \lambda_1 &= g(\mu_1) \\ \lambda_2 &= 1 \end{aligned} \tag{3.33}$$

and

$$g(s) = \begin{cases} 1 & (s \leq 0) \\ 1 - exp\left(\frac{-C}{(s/\lambda)}\right) & (s > 0) \end{cases} \tag{3.34}$$

where C is a threshold and λ is the contrast parameter. The λ_i, for $i =< 1, 2 >$, configuration makes the diffusion parallel to v_2 if $\mu_1 \gg 0$. Otherwise, if $\mu_1 \leq 0$ it diffuses isotropically with $\lambda_1 = \lambda_2 = 1$. G is constructed based on the eigenvectors of the non-average diffusion tensor J_0.

CED enhances flow-like structures such as wood, fabrics, and fingerprints. It fills in gaps and completes interrupted lines. It averages gradient orientation over a large field. G is constructed based on the eigenvectors of J_ρ and its λ_i are:



(a) CED method (b) EED method

Figure 3.20: *The diffusion orientation of the CED and EED methods at specified regions; P_a, P_b, and P_c, of a noisy square image is illustrated by the respective arrows on location. $< v_1, v_2 >$ are the eigenvectors. The length of the arrows expresses the emphasis and amount of applied diffusion; longer arrow signifies more diffusion. (a) the behavior of CED: it is not equipped to smooth noise on regions free noise (P_a) as such very little diffusion is applied along v_1 and v_2. (b) the behavior of EED: it is not equipped to smooth noise at junctions (P_b); it always favors one direction over the other. This deforms details at junctions.*

$$\lambda_1 = \alpha$$
$$\lambda_2 = \begin{cases} \alpha & if\ \mu_1 = \mu_2 \\ \alpha + (1-\alpha) exp\left(\frac{C}{(\mu_1 - \mu_2)^2}\right) & otherwise \end{cases} \tag{3.35}$$

where $C > 0$ serves as a threshold and $(\mu_1 - \mu_2)$ measures the local coherence. A large difference signifies the presence of flow-like structure due to the high variance in one direction. If the local coherence $\gg C$ then λ_2 converges to 1 diffusing more along v_2 than v_1; v_2 is also called the *coherence* orientation in CED. Otherwise, if the local coherence $\ll$ C then λ_2 converges to α, where $\alpha \in (0, 1)$ and mainly a small value is chosen ($\alpha = 0.001$).

These methods smooth well noisy images made up only of edges and flow-like structures. However, they are handicapped when they are applied on noisy images containing edgeless regions and corners. Their limits are illustrated by smoothing the noisy square displayed in Figure 3.20 at two locations: P_a– inside the square and P_b– at the corner. At P_a, the region is free of edges and the eigenvectors $< v_1, v_2 >$ are parallel to the x and y-axis respectively with eigenvalues $\mu_1 \simeq \mu_2 \simeq 0$. Filtering

Smoothing Method	Regions		
	no edges $\mu_1 \simeq \mu_2 \simeq 0$	edge $\mu_1 \gg \mu_2$	corner $\mu_1 \simeq \mu_2 \gg 0$
EED	$\lambda_1 = \lambda_2 = 1$ (good)	$\lambda_1 \simeq 0$, $\lambda_2 = 1$ (good)	$\lambda_1 \simeq 0$, $\lambda_2 = 1$ (not good)
CED	$\lambda_1 = \lambda_2 = \alpha$ (not good)	$\lambda_1 = \alpha$, $\lambda_2 = 1$ (good)	$\lambda_1 = \lambda_2 = \alpha$ (good)

Table 3.5: *The behavior of the methods EED and CED at specified regions of an image. EED works well on regions with and without edges but not on corners; it deforms them as more diffusion is applied in v_2 (see P_b in Figure 3.20(b)). CED works well on regions with edges and corners but performs poorly on edge free regions; filtering is very slow (see P_a in Figure 3.20(a)).*

out noise at P_a requires an isotropic diffusion along the x and y-axis. Using CED, λ_2 converges to α resulting in an isotropic diffusion in the v_1 and v_2 orientation as $\lambda_1 = \lambda_2 = \alpha$. However, since α is chosen as a very small value; filtering will be very slow. Many iterations of diffusion will be needed before removing the noise which might blur details of the image. This constrains the use of CED on regions free of edges such as P_a. The CED diffusion orientation is illustrated with arrows in Figure 3.20(a) where the length signifies the emphasis and amount of diffusion; more diffusion is applied in the orientation of longer arrows.

At the corner P_b, the eigenvectors $< v_1, v_2 >$ are also parallel to the x and y-axis respectively however their eigenvalues are $\gg 0$; $\mu_1 \simeq \mu_2 \gg 0$. This indicates no smoothing is needed as it is surrounded by edges. Using EED, diffusion is applied only in the v_2 orientation as λ_1 converges to 0 (Figure 3.20(b)). This deforms the corners of the image. Table 3.5 summarizes the behavior of the EED and CED methods on regions without edges, with edges, and with corners.

The above diffusion tensors proposed by Weickert is very similar to ours. We propose a multi-scale method that smoothes images that contains all sorts of shapes; including edges and corners, and preserve them. Our technique, presented in Chapter 4, unifies the above methods in the following way: instead of explicitly forming the diffusion tensor using specific eigenvalues, we build the diffusion tensor based on a local histogram of gradient, in a continuous and consistent manner.

3.6 Smoothing Volumetric Data

Volumetric scans of objects are of high interest in medical imaging, engineering and analysing cultural heritage. They are produced using *tomographic reconstruction*, a technique that combine a large series of 2D scans captured from multiple views. Typically, penetrative radiation is used to obtain each 2D scan: X-Rays for CT scans, radio-frequency waves for MRI (magnetic resonance imaging), electron-

positron annihilation for PET scans, etc. Noise in the voxelised density data come from a variety of sources: the limited number of views, lack of captor sensitivity, high contrasts, the reconstruction algorithms, low-dose radiation, etc. Such data are constantly acquired with noise. Therefore, we want to reduce, or eliminate, noise as early as possible in the application. However, we want to remove the noise while preserving the sharp features of the volume object. This remains a challenging task. We explore next two main approaches for smoothing volumetric data: bilateral filter and the CED-EED methods.

3.6.1 Bilateral Filter on Volumetric Data

Bilateral filter is a very effective tool for smoothing noise in images. Therefore, it seems natural to choose it to filter out noise in volumes. However, we found little experiments on this subject. Fernandez et *al.* [Fernandez 2003] investigated bilateral [Tomasi 1998] and mean shift filtering [Comaniciu 2000, Comaniciu 1998, Comaniciu 2002] on 3D CT images. They extended the bilateral filter to three-dimensional space by adding an extra dimension to its two-dimension configuration and compared its application on 2D slices of the volume and on the complete 3D dataset. They've concluded that results are slightly better on the 3D dataset although its run-time is much more costly. Moreover, they found that the smoothing results of their mean shift implementation are better than the bilateral filter. Both algorithms were implemented on the CPU. Thus, their execution time are very expensive: 5 hours for the bilateral filter to process a volume of roughly 23 million voxels.

Jiang et *al.* [Jiang 2003] investigated the effectiveness of bilateral filter for denoising various biological electron microscopy subjects such as molecular complexes and segmented protein. They found that it effectively suppresses the noise without blurring the high resolution details. However, it is not without some potential problems. The values of its domain and range parameters are choices that depend on a trial and error approach. Thus, an inappropriate choice of these parameters may smooth the noise but also smooth the fine details of the model. To overcome the slow speed of the algorithm on large volumes, they parallelized the bilateral filter using the message passing interface (MPI) standard [Snir 1998, Gropp 1998]. Then, they segmented the volume into small pieces and processed them independently by different processors before gathering the results.

In Chapter 5, we compare our method with the bilateral filter. We have reached similar conclusions to that of [Jiang 2003]. For some values of the parameters, it keeps the sharp features but also keeps the noise and extending these parameters until the noise is filtered blurs away the features and details of the model. We implemented the bilateral filter in 3D on a GPU using CUDA [Sanders 2010]. Thus, its run-time is very fast. Moreover, our implementation is scalable to dataset of arbitrary size. We can process dataset of 27 million voxels in a manner of few minutes (see Chapter 5 for results).

3.6.2 EED and CED on Volumetric Data

The edge-enhancing diffusion (EED) and coherence-enhancing diffusion (CED) methods introduced in Section 3.5.4.1 were extended from their two-dimensional counterpart to three-dimensional space to be applied on volumes or 3D images. However, their configuration are again tailored to specified applications. They are mainly applied on 3D tomography reconstruction models consisting of homogeneous features such as spherical and tubular forms. For instance, [Achilleas 2001] use them on pleomorphic biological objects and [Meijering 2002] use them on blood vessels.

[Achilleas 2001] proposed a hybrid between the EED and CED methods, with numerous conditions to follow and several parameters to tune to achieve the best result. It uses $(\mu_1 - \mu_3)$; the difference between the first and third eigenvalues of J_ρ (Equation 3.31), as a switch condition. EED is applied when the difference value is lower than a chosen threshold otherwise CED is applied. The threshold is computed *ad hoc* from the variance calculated over only a noisy area. They configure the two methods as follow:

- for EED, λ_i are:

$$\begin{aligned} \lambda_1 &= \lambda_2 = g(\mu_1) \\ \lambda_3 &= 1 \end{aligned} \tag{3.36}$$

- for CED, λ_i are:

$$\begin{aligned} \lambda_1 &= \lambda_2 = \alpha \\ \lambda_3 &= \begin{cases} \alpha & if\ \mu_1 = \mu_3 \\ \alpha + (1-\alpha) exp\left(\frac{C}{(\mu_1 - \mu_3)^2}\right) & otherwise \end{cases} \end{aligned} \tag{3.37}$$

  where μ_1 and μ_3 are the highest and lowest variance respectively.

- The diffusion tensor G is:

$$G = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}^T \tag{3.38}$$

  where $< v_1, v_2, v_3 >$ are the eigenvectors of J_0 and J_ρ for EED and CED respectively.

As their two-dimensional counterparts, the three-dimensional CED and EED are tailored for smoothing 3D images that consist of only flow-like and edges structures. They are not equipped for smoothing models consisting of noisy surfaces and corners. Consider the two points, A on the surface and B on the corner of the noisy cube illustrated in Figure 3.21 at which smoothing will be exercise.

At point A, the eigenvectors $<v_1, v_2, v_3>$ are parallel to the x, y, and z-axis respectively with eigenvalues $\mu_1 \gg \mu_2 = \mu_3$. Smoothing out noise on the surface of

Smoothing Method	Regions		
	surface with no edges $\mu_1 \gg \mu_2 \simeq \mu_3$	edge $\mu_1 \simeq \mu_2 \gg \mu_3$	corner $\mu_1 \simeq \mu_2 \simeq \mu_3 \gg 0$
EED-CED discrete hybrid	$(\mu_1 - \mu_3) \gg 0$ $\Rightarrow$ CED is chosen and $\lambda_1 = \lambda_2 = \alpha$, $\lambda_3 = 1$. Diffusion is applied only along v_3 rather than along v_2 and v_3. (see pt. A in Figure 3.21)	$(\mu_1 - \mu_3) \gg 0$ $\Rightarrow$ CED is chosen and $\lambda_1 = \lambda_2 = \alpha$, $\lambda_3 = 1$. Diffusion is applied only along v_3, the edge, As we want. (see pt. C in Figure 3.21)	$(\mu_1 - \mu_3) \simeq 0$ $\Rightarrow$ EED is chosen $\lambda_1 \simeq \lambda_2 \simeq 0$, $\lambda_3 = 1$ Diffusion is applied along v_3 rather than none at all. (see pt. B in Figure 3.21)

Table 3.6: *The behavior of the discrete EED-CED hybrid at specified location of a volume cube (Figure 3.21). The hybrid chooses correctly only on the edge location. On the surface and corner, it chooses the CED method which diffuses in the v_3 orientation. This will create artifact in that orientation rather than removing the noise.*

point A requires diffusing along the yz-plane and none or very little in the x-axis orientation. This is not the case if the hybrid EED-CED method is applied. Since the local coherence, $(\mu_1 - \mu_3)$, is very large, CED will be applied with $\lambda_1 = \lambda_2 = \alpha$ and $\lambda_3 = 1$. Thus, diffusion will only be applied along the z-axis orientation where noise will be removed and create a horizontal artifact. For the sake of clarification, if EED was chosen a similar scenario would occur with $\lambda_1 = \lambda_2 = g(\mu_1)$ and $\lambda_3 = 1$. The diffusion orientation is illustrated with arrows in Figure 3.21 where the thickness and length signify the emphasis and amount of diffusion; more diffusion is applied in the orientation of thicker arrows.

At the corner B, the eigenvectors $< v_1, v_2, v_3 >$ are also parallel to the x, y, and z-axis respectively however unlike at point A the eigenvalues are equivalent; $\mu_1 \simeq \mu_2 \simeq \mu_3 \gg 0$. This indicates no smoothing is needed as it is surrounded by edges. However, in the hybrid method EED is applied since the local coherence is zero. Thus, diffusion is rather applied in the orientation of v_3 (Figure. 3.21). This deforms the model. Choosing CED is logically more correct as diffusion along each orientation is limited to a small quantity with $\lambda_1 = \lambda_2 = \lambda_3 = \alpha$. Table 3.6 summarizes the behavior of the discrete EED-CED hybrid of [Achilleas 2001] on noisy volumes consisting of corners, edges, and surfaces.

Applying the EED and CED methods separately on volumes such as the noisy cube have also their disadvantages. EED works well only on edges. CED works well only on edges and corners; both are not applicable on regions free of edges. Table 3.7 summarizes their behavior individually.

The λ_i parameters of EED and CED can be handcrafted freely to match the need of a particular application. For instance, [Meijering 2002] use only EED on 3D

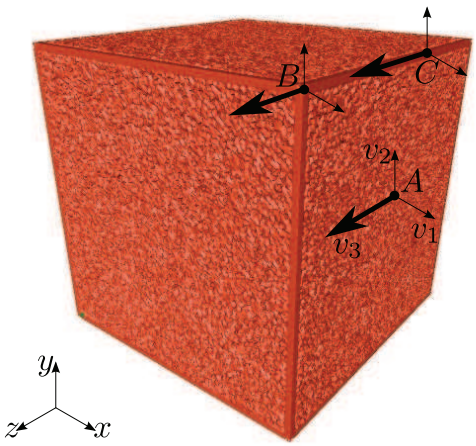


Figure 3.21: *The EED-CED diffusion orientation on a noisy surface (point A) and a corner (point B) is illustrated by the respective arrows on location.* $< v_1, v_2, v_3 >$ *are the eigenvectors of the EED-CED diffusion kernel. The thickness and length of the arrows expresses the emphasis and amount of applied diffusion; thicker arrow signifies more diffusion. According to the EED-CED condition, at both points, A and B, diffusion is carried in the* v_3 *orientation; i.e. z-axis.*

reconstructed blood vessels and configure their eigenvalues as follow:

$$\begin{aligned} \lambda_1 &= g(\mu_1) \\ \lambda_2 &= \lambda_3 = 1 \end{aligned} \tag{3.39}$$

Thus, diffusion is always applied along the v_2v_3-plane with minimal diffusion along v_1. As the hybrid EED-CED, it has its limits when applied to volumes with edges and corners. Table 3.7 summarizes its behavior on the surface, edge, and corner of the noisy cube (Figure 3.21). At point A, it behaves as we wish smoothing along the yz-plane as $\lambda \simeq 0$ and $\lambda_2 = \lambda_3 = 1$. However, on the edge (point C) it diffuses along v_2 in addition to diffusing along the edge (v_3). This blurs away the edge in the direction of v_2. The same scenario is repeated on the corner (point B); deforming the corner where diffusion is not required.

As we have seen, the EED and CED methods are not applicable on general models consisting of different shapes and structures. This is also true for the discrete hybrid proposed by [Achilleas 2001]. To deal with the intermediate geometries not covered by these methods, Mendrik et *al.* [Mendrik 2009] proposed a continuous hybrid of the EED-CED methods by explicitly forging the two methods. They use a linear combination of the methods respective eigenvalues λ of their diffusion tensor G:

$$\lambda_{h_i} = (1 - \varepsilon)\lambda_{c_i} + \varepsilon\lambda_{e_i} \tag{3.40}$$

Smoothing Method	Regions		
	surface with no edges $\mu_1 \gg \mu_2 \simeq \mu_3$	edge $\mu_1 \simeq \mu_2 \gg \mu_3$	corner $\mu_1 \simeq \mu_2 \simeq \mu_3 \gg 0$
EED in [Achilleas 2001]	$\lambda_1 \simeq \lambda_2 \simeq 0$ $\lambda_3 = 1$ (not good)	$\lambda_1 \simeq \lambda_2 \simeq 0$ $\lambda_3 = 1$ (good)	$\lambda_1 \simeq \lambda_2 \simeq 0$ $\lambda_3 = 1$ (not good)
EED in [Meijering 2002]	$\lambda_1 \simeq 0$ $\lambda_2 = \lambda_3 = 1$ (good)	$\lambda_1 \simeq 0$ $\lambda_2 = \lambda_3 = 1$ (not good)	$\lambda_1 \simeq 0$ $\lambda_2 = \lambda_3 = 1$ (not good)
CED	$\lambda_1 = \lambda_2 = \alpha$ $\lambda_3 = 1$ (not good)	$\lambda_1 = \lambda_2 = \alpha$ $\lambda_3 = 1$ (good)	$\lambda_1 = \lambda_2 = \lambda_3 = \alpha$ (good)

Table 3.7: *The behavior of the methods EEDs and CED individually at specified locations of a volume cube (Figure 3.21). The EED method of [Achilleas 2001] works well only on regions with edges. At other regions it introduces artifact. The EED method of [Meijering 2002] works well only on regions free of edges and corners. CED is applicable only on regions with edges and corners.*

---

where λ_{h_i} is the eigenvalue of the continuous hybrid diffusion tensor. λ_{c_i} and λ_{e_i} are the eigenvalues of CED (Equation 3.37) and EED (Equation 3.39) respectively. ε regulates the amount of value needed from each of λ_c and λ_e, where CED is used when $\varepsilon \to 0$ and EED is used when $\varepsilon \to 1$. The ε fraction is:

$$\varepsilon = exp^{\frac{\mu_2(\lambda_h^2(\xi-|\xi|)-2\mu_3)}{2\lambda_h^4}} \tag{3.41}$$

where λ_h is the contrast threshold and ξ is:

$$\xi = \left(\frac{\mu_1}{\alpha+\mu_2} - \frac{\mu_2}{\alpha+\mu_3}\right) \tag{3.42}$$

The continuous hybrid of [Mendrik 2009] generalizes the use of the EED-CED methods on objects of different shapes and structures. For instance, it works well on the different location of the noisy cube of Figure 3.21. However, to achieve the best result there are 8 parameters to tune including the conditions to follow in the EED-CED methods (see Section 3.5.4.1). This makes things tedious for the user.

The objective of [Mendrik 2009] is very similar to ours. However, we propose a different approach for smoothing volumes while preserving the sharp features. Our technique, presented in Chapter 5, unifies the above methods in the following way: instead of explicitly forming the diffusion tensor using specific eigenvalues, we build the diffusion tensor based on a local histogram of gradients, in a continuous and consistent manner. Moreover, we have only one parameter to tune — the threshold for the size of features to be preserved.

Further extension of EED-CED were proposed: Frangakis et al. [Achilleas 2001], combines the diffusion hybrid and the wavelet transform to reconstruct biomedical data while preserving edges. Schaap et al. [Schaap 2008] construct an importance map indicating the degree of significance of all voxels at any arbitrary scale by considering their total curvature, which is computed as the L2-norm of their Hessian matrix's eigenvalues. The result is interpolated with the smoothed volume by EED.

Chapter 4

Fast Multi-Scale Feature-Preserving Smoothing of Images

Contents

4.1 Introduction

Inspired by the Perona-Malik [Perona 1990] anisotropic diffusion algorithm, we propose a new multi-scale smoothing kernel that adapts to the local features of two-dimensional images. The result is a smooth 2D image where high frequency variations below a user-specified threshold is cancelled. Our algorithm consistently preserves sharp features such as edges and corners. It runs efficiently on parallel

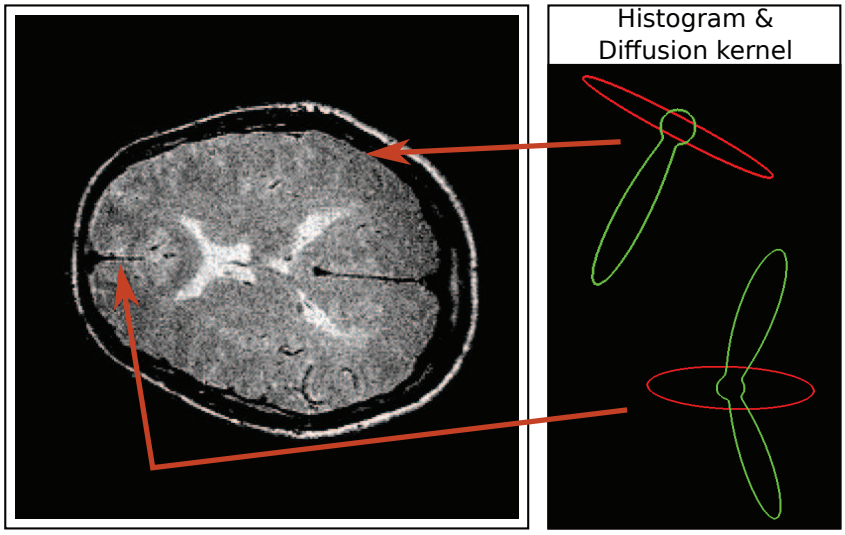


Figure 4.1: *Continuous local gradient histogram with their respective diffusion tensors at two different locations of a noisy MRI brain. The top figure shows a histogram (in green) with one dominant direction computed at the edge of the brain. Its diffusion tensor (in red) is parallel to the edge. The bottom figure shows the histogram computed between two hemispheres resulting in two dominant directions. To preserve the edges of the hemispheres diffusion should be done parallel to them. As such, the diffusion tensor is parallel to the edges.*

---

processors (such as GPUs) and is scalable to large datasets. Tuning is simple, with only one parameter—the threshold for the size of features to be preserved.

First, we compute *local continuous histograms* of gradients of the image. Then we compute diffusion tensors using these histograms. Finally, we apply *anisotropic diffusion* in the 2D data using the computed tensors. Since the diffusion at each pixel is guided by the local distribution of gradients, the smoothing respects local features. These operations are done in parallel, on the GPU, for entire blocks of pixels.

Figure 4.1 shows examples of continuous local histograms (in green) and their respective diffusion tensors (in red) that we have computed at two different locations on a noisy MRI brain. The top figure shows a histogram with one dominant direction computed at the boundary of the brain. Regions, covered by the specified feature size, having a single edge have uniform oriented gradients. Thus, a single lobe continuous histogram. Its diffusion tensor is orthogonal to the histogram lobe; *i.e.* we perform diffusion along the edge. The bottom figure shows a histogram with two dominant directions where we compute over a region with two different gradient distribution. Once more, the diffusion tensor is orthogonal to both lobes of the histogram performing the diffusion along the edges. Hence, removing the noise and preserving the edges.

In this chapter, we present a detail explanation of the theory behind our algorithm and our implementation. Then, we analyze our results and compare with other smoothing methods.

Introduction

Inspirés par l'algorithme de diffusion anisotrope de Perona-Malik [Perona 1990], nous proposons un nouveau noyau de lissage multi-échelle qui s'adapte aux caractéristiques locales des images 2D. Le résultat est une image lisse où les variations hautes fréquences en-dessous d'une valeur spécifiée par l'utilisateur sont supprimées. Notre algorithme préserve régulierement les lignes caractéristiques comme les arrêts et les coins. Notre algorithme tourne efficacement sur processeur parallèle (comme les GPUs) et peut être appliqué à des données plus larges. Le réglage est relativement facile, avec un seul paramètre que l'on appelle *taille caractéristique*—la taille minimale des caracteristiques que l'on souhaite préserver.

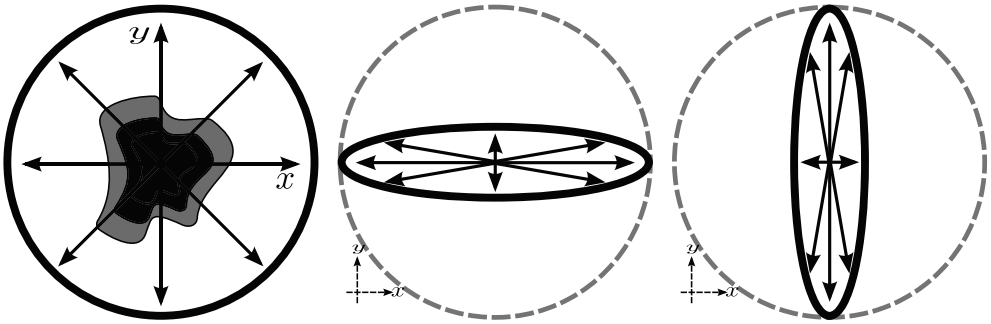
Dans un premier temps, nous calculons *l'histogramme continu local* des gradients de l'image. Ensuite, nous calculons les tenseurs de diffusion en utilisant les histogrammes calculés. Finalement, nous appliquons la *diffusion anistrope* sur les donn'ees 2D en utilisant les tenseurs ainsi obtenus. Le lissage respecte les lignes caractéristiques puisque la diffusion à chaque pixel est guidée par la distribution locale des gradients. Le calcul des tenseurs et la diffusion sont faits en parallèle sur le GPU par blocs entiers de pixels.

Figure 4.1 montre des exemples d'histogrammes continus locaux (en vert) et leur tenseur respectif de diffusion (en rouge) que l'on a calculés à deux différentes positions d'une image IRM bruitée d'un cerveau. La figure du haut montre un histogramme dominé par une direction calculée au bord du cerveau. Les regions d'une taille caracteristique donnée et qui ne rencontrent qu'une seule arrête, ont une orientation de gradient uniforme. Par conséquent, elles présentent un histogramme continu avec seul lobe. Leur tenseur de diffusion est orthogonal à la direction du lobe de l'histogramme afin que la diffusion soit appliquée le long de l'arrête. La figure en bas montre un histogramme avec deux directions dominantes qui a été calculé à partir d'une région qui abrite une distribution de deux différents gradients. Encore une fois, le tenseur de diffusion est orthogonal aux deux lobes du histogramme. Cela enlève le bruit et préserve les arrêtes.

Dans ce chapitre, nous présenterons une expliquation détaillée de la théorie de notre algorithme ainsi que notre implementation. Ensuite, nous analyserons les resultats et nous les comparerons avec d'autres méthodes de filtrage.

4.2 Theoretical Background

We review two tools that are key to our method: anisotropic diffusion and local continuous histogram.



(a) $C_1 = C_2$ (b) $C_1 \gg C_2$ (c) $C_1 \ll C_2$

Figure 4.2: *(a) Isotropic diffusion is performed as $C_1 = C_2$. (b) $C_1 \gg C_2$: Diffusion is applied more in the x-axis than the y-axis. This is illustrated by the transformation of the circle into a cigar oriented in the x-axis. (c) $C_1 \ll C_2$: Diffusion is applied more in the y-axis.*

4.2.1 Anisotropic Diffusion

The general form of anisotropic diffusion introduced by Perona and Malik [Perona 1990] for smoothing images is

$$\begin{aligned} I_t &= div(M\nabla I) \\ &= M\Delta I + \nabla M \nabla I \end{aligned} \tag{4.1}$$

where I is an image of intensity or density values and M is the diffusion tensor, a symmetric matrix. If M consists of equal constant values C, then Equation 4.1 reduces to an isotropic heat equation:

$$\begin{aligned} I_t &= M\Delta I \\ &= tr\left(\begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix}\begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}\right) \\ &= tr(MH(I)) \end{aligned} \tag{4.2}$$

where $C_1 = C_2$ are the scales of the diffusion and $H(I)$ is the Hessian matrix of image I. This is illustrated by the circle in Figure 4.2(a). Diffusion is processed equally in all directions demonstrated by the outward directed arrows. Moreover, diffusion is applied proportionally to C_1 and C_2. Assigning different value to C_1 and C_2 where $C_1 \neq C_2$ results in sort of anisotropic diffusion; it alters the balance of diffusion in the direction of the largest of C_1 and C_2. If $C_1 > C_2$, more diffusion is applied along the x-axis; *i.e.* I_{xx}, than the y-axis; *i.e.* I_{yy}. From Figure 4.2(b), when $C_1 \gg C_2$ the circle is transformed into a cigar shape oriented in the x-axis. It's the inverse for $C_2 \gg C_1$; shown in Figure 4.2(c).

The above configuration limits our control of diffusion to only two orientations;

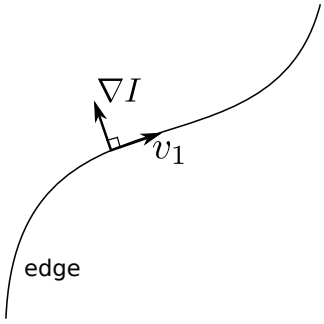


Figure 4.3: *An orthonormal 2D basis formed over an edge. It consists of the gradient vector ∇I and the vector v_1. ∇I is orthogonal to the edge. v_1 is parallel to the edge and orthogonal to ∇I.*

---

the x-axis and y-axis. We can remove this limitation by expressing Equation 4.2 using the eigen decomposition of M. Since M is symmetric, it can be written as $M = V^T DV$ where D is a diagonal matrix with eigenvalues λ_i and V contains the eigenvectors v_i of M:

$$\begin{aligned} I_t &= tr(V^T DVH(I)) \\ &= tr(DVH(I)V^T) \\ &= \textstyle\sum_i \lambda_i v_i^T H(I) v_i. \end{aligned} \tag{4.3}$$

In this form, we achieve anisotropic diffusion. It uses the Hessian matrix as a quadratic form to measure the directional second derivatives along vectors v_i, and applies diffusion proportionally to the eigenvalues λ_i. Through this expression, we have the freedom to steer the diffusion process in any direction by simply choosing the suitable v_i and λ_i. For instance, geometric diffusion corresponds to diffusing only orthogonally to the gradient ∇I, using the diffusion tensor:

$$M_g = V^T \begin{bmatrix} \varepsilon & 0 \\ 0 & 1 \end{bmatrix} V \text{ with } V = \left[\frac{\nabla I}{\|\nabla I\|}, v_1 \right] \tag{4.4}$$

where v_1 is a vector that completes ∇I into an orthonormal 2D basis (see Figure 4.3). Choosing a very small ε value reduces the diffusion in the ∇I direction to almost nothing and diffuses only in the v_1 direction. Strictly speaking, geometric diffusion corresponds to $\varepsilon = 0$. In practice, it is better to have $0 < \varepsilon \ll 1$ to keep M_g invertible. An invertible tensor M signifies that applying a geometric diffusion into an image is equivalent to applying a Gaussian filter of covariance M to this image:

$$I(v, t+1) = \int e^{-u^T M^{-1} u} I(v+u, t) du \tag{4.5}$$

where v is the central pixel (x, y) and u is a neighboring pixel of v. We will use

this Gaussian property to derive a feature preserving diffusion tensor.

For clarity, the Gaussian $e^{-u^T M^{-1} u}$ simplifies to its classic form $e^{-\|u^2\|}$ and $e^{-\frac{\|u^2\|}{\sigma^2}}$ if M is respectively an identity matrix and a diagonal matrix with σ^2. This is achieved by expanding:

$$e^{-u^T M^{-1} u} = e^{-\begin{bmatrix} x & y \end{bmatrix} M^{-1} \begin{bmatrix} x \\ y \end{bmatrix}}$$

and replacing M by the identity matrix and diagonal σ^2 matrix.

- For an identity matrix M, we get:

$$\begin{aligned} e^{-u^T M^{-1} u} &= e^{-\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}} \\ &= e^{-(x^2+y^2)} = e^{-\|u^2\|} \end{aligned} \tag{4.6}$$

- and for a diagonal matrix of σ^2:

$$\begin{aligned} e^{-u^T M^{-1} u} &= e^{-\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \end{bmatrix}} \\ &= e^{-\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \frac{1}{\sigma^2} & 0 \\ 0 & \frac{1}{\sigma^2} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}} \\ &= e^{-\frac{(x^2+y^2)}{\sigma^2}} = e^{-\frac{\|u^2\|}{\sigma^2}}. \end{aligned} \tag{4.7}$$

We invert the diffusion tensor M to simplify the calculation. It is reversed to its original state at the end of the calculation.

4.2.2 Local Continuous Histogram

Figure 4.4: *A continuous smooth histogram based on a Gaussian distribution.*

Local continuous histograms are extensions of discrete histograms. Each value in the data contributes to neighboring bins—proportionally to a weight function such as a Gaussian—in addition to the histogram bin that it falls into. Local continuous

histograms allow robust estimation of histogram properties such as modes and extrema. They work well for edge-aware smoothing operations of images, as pointed out by Kass and Solomon [Kass 2010].

The following definitions hold for histograms both in 2D and 3D. Let F be a 2D or 3D dataset, and $f(x)$ be the data value at point x. The local continuous histogram at bin b is defined as a function of point x over the entire dataset by:

$$\begin{aligned} h_b(x) &= \int_F K_b(f(y))g_\sigma(x-y)dy \\ \text{with} \quad & g_\sigma(x) = \frac{1}{\sigma}e^{-\frac{\|x\|^2}{\sigma^2}} \end{aligned} \tag{4.8}$$

the histogram kernel K_b represents how much the value $f(y)$ contributes to the histogram bin at b. It is usually a Gaussian. The sharper this Gaussian, the more local the histogram is. The other Gaussian g_σ is a spatial kernel that rules the influence of neighboring points to the local histogram at x. Using a pulse function (of the size of the bin) for K_b and a spatial pulse function for g_σ results in a local discrete histogram. Figure 4.4 illustrates a continuous smooth histogram based on a Gaussian distribution.

Using Equation 4.8, we can compute local continuous histograms h_x for all values x at once in a very efficient way, using only two Fourier transforms. First, we rewrite Equation 4.8 as a convolution over x:

$$h_b(x) = (g_\sigma \otimes_x (K_b \circ f))(x)$$

Then, from the convolution theorem, and noting the Fourier transform with $\mathcal{F}$, we have:

$$\mathcal{F}(h_b) = \mathcal{F}(g_\sigma) \times_{\omega_x} \mathcal{F}(K_b \circ f)$$

where the product is performed over the spatial spectral variable ω_x. The spectrum of the Gaussian g_σ is also a Gaussian, $g_{\frac{1}{\sigma}}$, therefore:

$$h_b = \mathcal{F}^{-1}(g_{\frac{1}{\sigma}} \times_{\omega_x} \mathcal{F}(K_b \circ f)) \tag{4.9}$$

4.3 Feature Preserving Smoothing in 2D Images

4.3.1 Objectives

We apply anisotropic diffusion within the image. We compute adaptive diffusion tensors at each pixel, such that noise is smoothed up to a specified scale and features larger than this scale are preserved.

In Figure 4.5, we provide an intuition of how this diffusion tensor is constructed. It shows an image of a square with a wave-like structure on its top and bottom sides. On the wave-like structure, when the user specifies a fine scale, we would like

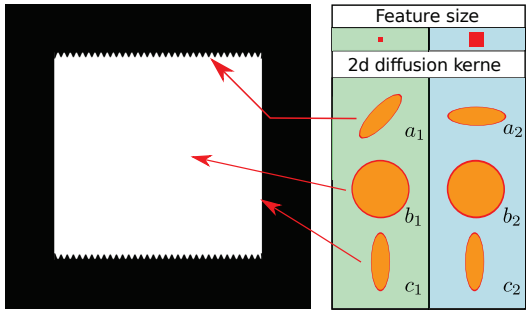


Figure 4.5: *We adapt the diffusion tensors to local variation by respecting the specified scale. For example, in the wave-like structure, a choice of fine scale smoothing requires diffusion along tensor a_1 while coarse smoothing requires tensor a_2 to smooth the thread.*

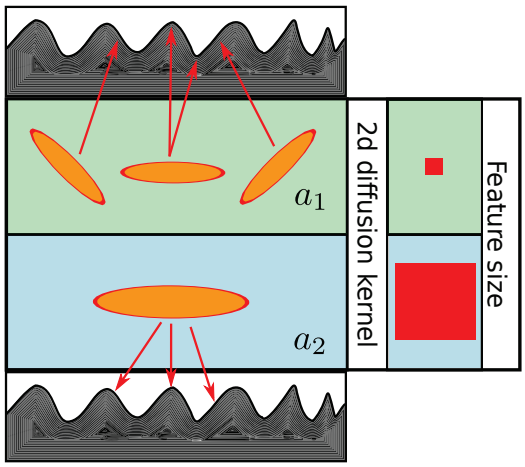


Figure 4.6: *Close-up view of the wave structure of Figure 4.5. For a small feature size, we produce diffusion tensors that adapts locally to the structure and diffuse parallel to it a_1 to preserve the structure. For a large feature size, we would like to filter out the thread as such we produce a diffusion tensor a_2 parallel to the whole structure rather than locally.*

to preserve the wave and must diffuse locally parallel to each of the wave (using kernel a_1). At a larger scale, however, we would like to smooth the wave to produce a flat line, which requires a different kernel a_2. These are better visualized at the close-up view in Figure 4.6. At other locations in the image, the smoothing kernel may be the same across multiple scales such as b_1 and b_2 where the result is a circle.

We construct the diffusion tensor at each pixel using the local distribution of gradients, filtered at the user-specified scale. Filtered gradient distribution are computed by building local continuous histograms in the space of directions. To account

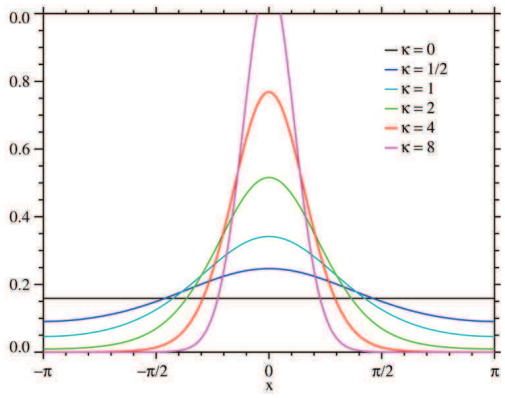


Figure 4.7: *The Von Mises distribution. As k increases the distribution becomes sharper and more local. Courtesy from wikipedia.*

---

for the feature size, we compute gradients on a blurred version of the image, obtained using an isotropic Gaussian of variance equal to half the requested feature size. We then use the filtered gradient histogram to build a diffusion kernel, at each pixel, that preserves the requested feature size.

4.3.2 Scale-space Local Gradient Distributions

Let s be the specified feature size and f the image data. The continuous histogram of gradients has value $h_\omega(x)$ at point x in direction ω, defined by:

$$h_\omega(x) = \int_{\|y-x\|<2s} g_s(x-y) K_\omega(\nabla f(y)) dy \tag{4.10}$$

where K_w is the normalized Von Mises kernel [Jammalamadaka 2001] defined by

$$K_\omega(\omega') = \frac{k}{2\pi} e^{k\omega\cdot\omega'} \tag{4.11}$$

and $\omega \cdot \omega'$ is the dot product between directions ω and ω'. Larger values of k correspond to sharper kernels K_ω (see Figure 4.7). A continuous gradient histogram of each pixel is discretized and constructed as follow:

- We create n directional bins. They are oriented in a circular style and are equiangular. The greater the number of bins the smoother is the histogram. Figure 4.8(a) illustrates a histogram with n directional bins.

- We compute a weight for each gradient of the image; to estimate its contribution, relative to each of the directional bin. This is accomplished via the Von Mises kernel (Equation 4.11).

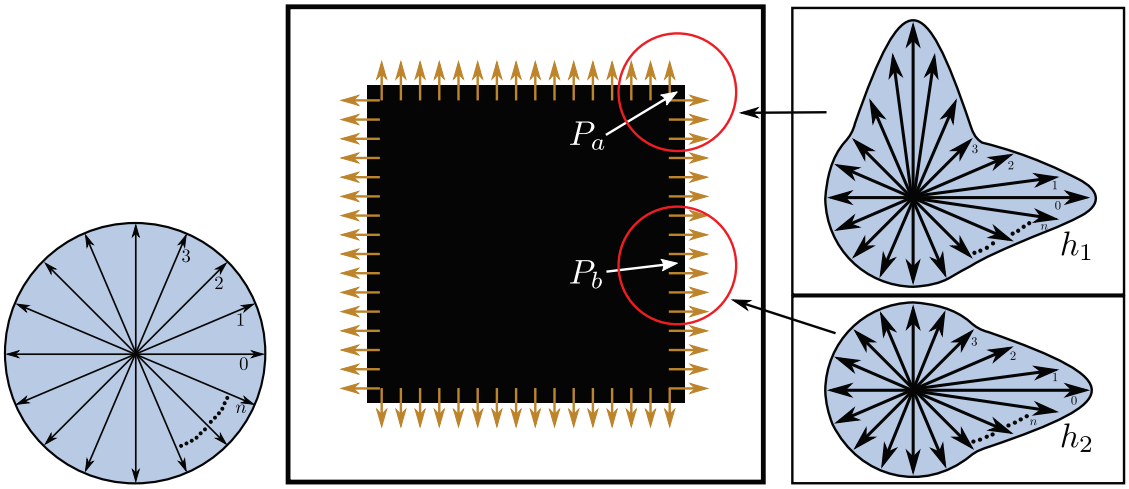
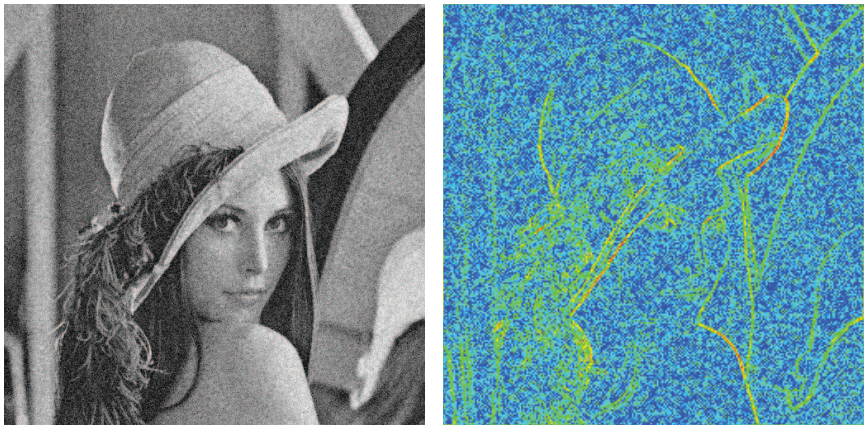


Figure 4.8: *(a) Continuous histogram with n equiangular directional bins oriented in a circular fashion. (b) Shows an image of two concentric square; a black one and a white one. It also displays two examples of local continuous gradient histograms—h_1 taken at a corner pixel P_a and h_2 at a side pixel P_b. The yellow arrows represent the gradients at the boundaries of the black square and the circles represent the smooth scale—they have the same feature size s. At P_a, histogram h_1 is produced. It has two distinct lobes directed upward and to the right as s encircles an equal distribution of gradient oriented upward and to the right. At P_b, s encircles only gradients directed to the right producing a one lobe histogram h_2 directed to the right.*

- Finally, we apply a spatial Gaussian g_s weight of variance s over the image. This excludes all pixels that falls outside the scope of the user-defined feature size s from the central pixel.

Figure 4.8(b) shows two examples of continuous gradient histograms taken at two locations; P_a—a corner and P_b—a side, of the concentric black and white squares image. The yellow arrows express the gradients at the boundaries of the black square and the red circles represent the feature size s. The circle centered at P_a encircles an equal number of gradients upwards and rightwards. This produces histogram h_1 with two distinct lobes smoothly directed upwards and to the right relative to the gradient distribution. The second circle positioned at P_b encircles only gradients directed to the right. This however produces a single lobe histogram, h_2, also directed to the right. Note, only gradients that falls inside the user-defined feature size are considered in the building of a histogram. Gradients outside the feature size scope are disregarded.

We use two Fourier transform to compute the local continuous gradient histogram. Furthermore, note that we compute the continuous gradient histogram on



(a) (b)

Figure 4.9: *(a) Noisy Lena image. (b) Gradient display of (a).*

the gradients of a blurred image of f rather than the gradients of f itself. Blurring an image sharpens the gradients by filtering out small scale variation of the data. We apply a 2D Gaussian $g_{\frac{s}{2}}$ of variance $\frac{s}{2}$ on the image data f using a convolution or equivalently two Fourier transform:

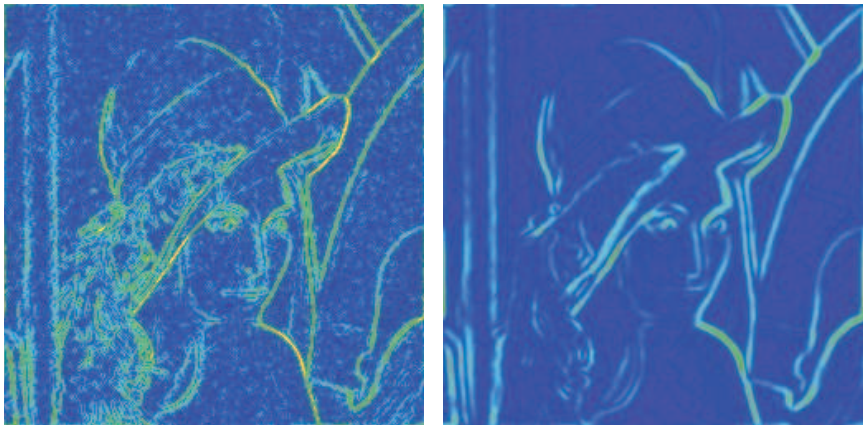
$$\begin{aligned} f^b &= g_{\frac{s}{2}} \otimes f \\ \text{or} & \\ f^b &= \mathcal{F}^{-1}(g_{\frac{2}{s}} \times \mathcal{F}(f)) \end{aligned} \tag{4.12}$$

where f^b is the resulted blurred image. Thus, $\nabla f^b(y)$ replaces $\nabla f(y)$ in Equation 4.10.

We use the Gaussian g twice in the process. First to filter out small scale variations of the data (Equation 4.12), then to spread the contribution of each gradient to the histogram of nearby pixels (Equation 4.10). The variance is half the scale of the requested feature size in Equation 4.12 and equal to the feature size in Equation 4.10. Figure 4.9 shows a noisy Lena image (a) and its gradient distribution (b). The small scale variation is represented by the bright spots dispersed over the image. Figure 4.10(a) and Figure 4.10(b) show the gradients blurred at feature size $s = 5$ and $s = 15$. Note that the small variations diminished and gradients of the image have sharpened.

4.3.3 Computation of Adaptive Diffusion Tensors

Having the local distribution of gradients $\omega \to h_\omega(x)$ at a scale s for all point x in the image, we define the diffusion tensors as:



(a) (b)

Figure 4.10: *The gradients of noisy Lena image (Figure 4.9) blurred at feature size* 5 *and* 15. *Note that the dispersed gradient in Figure 4.9(b) have been filtered out.*

$$M(x) = \left(\frac{1}{\mathcal{H}(x)} \int_{\Omega} h_{\omega}(x) M_g^{-1}(\omega) d\omega \right)^{-1} \quad \text{with} \quad \mathcal{H}(x) = \int_{\Omega} h_{\omega}(x) d\omega \tag{4.13}$$

where $M_g(\omega)$ is the geometric diffusion tensor defined in the direction ω (see section 4.2.1); *i.e.* :

$$M_g(\omega) = V^T \begin{bmatrix} \varepsilon & 0 \\ 0 & 1 \end{bmatrix} V \text{ with } V = [\omega, v_1] \tag{4.14}$$

where we use $\varepsilon = \frac{1}{1000}$. We compute a geometric diffusion tensor $M_g(\omega)$ for each directional bin and then sum them up. However, its contribution to the overall diffusion tensor $M(x)$ is weighted by $h_{\omega}(x)$ (Equation 4.10). $M(x)$ is then normalized by $\mathcal{H}(x)$—the sum of the local distribution of gradients in each of the histogram directional bins. Note that the matrix inversion involved is always well defined since we sum up positive definite matrices.

Equation 4.13 is justified as diffusing using the geometric diffusion tensor is equivalent to smoothing the data with a Gaussian kernel g defined by its covariance matrix M_g:

$$g(x) = e^{-x^t M_g^{-1} x}$$

To account for multiple directional constraint, Gaussian kernels corresponding to different directions are multiplied. Consider two main directions ω_1 and ω_2 orthog-

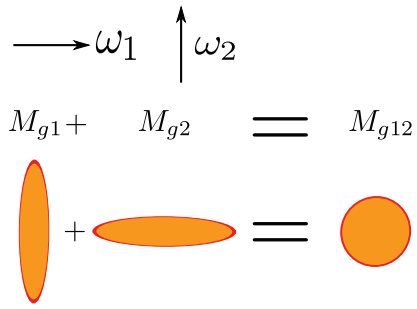


Figure 4.11: *M_{g1} and M_{g2} are the diffusion tensors of direction ω_1 and ω_2 respectively. They are orthogonal to each other. Their geometric mean is a circle.*

onal to one another. Multiplying the Gaussian kernels g_1 and g_2 is an appropriate solution since each Gaussian will cancel out the parts of the support that is not wanted in the other Gaussian. For a proper normalization, we actually need a geometric mean, which in the example above is:

$$g_{12} = (g_1 g_2)^{\frac{1}{2}} = e^{-x^t \frac{M_{g1}^{-1} + M_{g2}^{-1}}{2} x}$$

The geometric mean of Gaussians results in an harmonic mean of their covariance matrices as in Equation 4.13. We provide a visual description of the example above in Figure 4.11. We show the diffusion tensors M_{g1} and M_{g2} of directions ω_1 and ω_2 respectively. They are orthogonal to one another and their geometric mean M_{g12} is a circle. Note that the result is not the intersection of M_{g1} and M_{g2}.

To illustrate the procedure for computing adaptive diffusion tensors, consider the edge in Figure 4.12 along its gradients represented by the yellow arrows. We would like to build a continuous gradient histogram and its diffusion tensor at point P with the feature size s presented by the red circle. Figure 4.13 provides a visual aid. For the sake of simplicity, we create a histogram with only eight directional bins—a_1. a_2 shows the eight directional bins separated and spread. b_2 displays the contribution of the gradients within s to each of the directional bins in a_2. The length of the arrow is used as an estimator of the amount of gradients contributing to the respective directional bin. Note that no gradient contribute to the four rightmost directional bins as they are directed away from the gradients. The continuous gradient histogram b_1 is the product of the gathered contributed bins and adding it to a_1. c_2 displays the diffusion tensors relative to each of the directional bins. They are all orthogonal to their respective directional bins and scaled to their contributed gradients. Thus, they adapt to the gradients distribution and orientation. The geometric mean of all eight diffusion tensors is displayed by c_1. It is orthogonal to the continuous gradient histogram b_1.

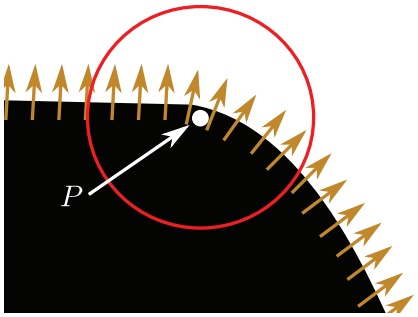


Figure 4.12: *An edge along with its gradient represented by the yellow arrows. The red circle represents a feature scale of size s.*

4.3.4 Examples

Figures 4.1 and 4.14 provide examples of continuous gradient histogram (in green) with their respective diffusion tensors at different positions (in red). Regions with a single edge produce continuous histograms with one dominant orientation. Its diffusion tensor is flat and parallel to the edge. Regions with junctions (two or more edges) produce continuous histogram with two or more dominant orientations. In this situation, the diffusion tensor has a circle-like shape as diffusion is done in all direction with very low eigenvalues. Thus, diffusion has little effect which preserves the junctions.

4.3.5 Diffusion

Following the computation of the diffusion tensor of each pixel, we apply anisotropic diffusion iteratively in the orientation of the respective tensors. Since the diffusion at each pixel is guided by the local distribution of gradients, the smoothing respects and preserves the local features of the image. We compute the eigenvalues and eigenvectors of each diffusion tensor. Then, we diffuse in the orientation of the eigenvectors proportionally to their respective eigenvalues. Mathematically, we define it as

$$I_t = \sum_{i=0}^{2} \frac{(V_i^T H(I) V_i)\lambda_i^2}{V_i^T V_i} \tag{4.15}$$

where I is an image of intensity or density values with $V_i = \begin{bmatrix} v_{x_i} \\ v_{y_i} \end{bmatrix}$ and λ_i are the

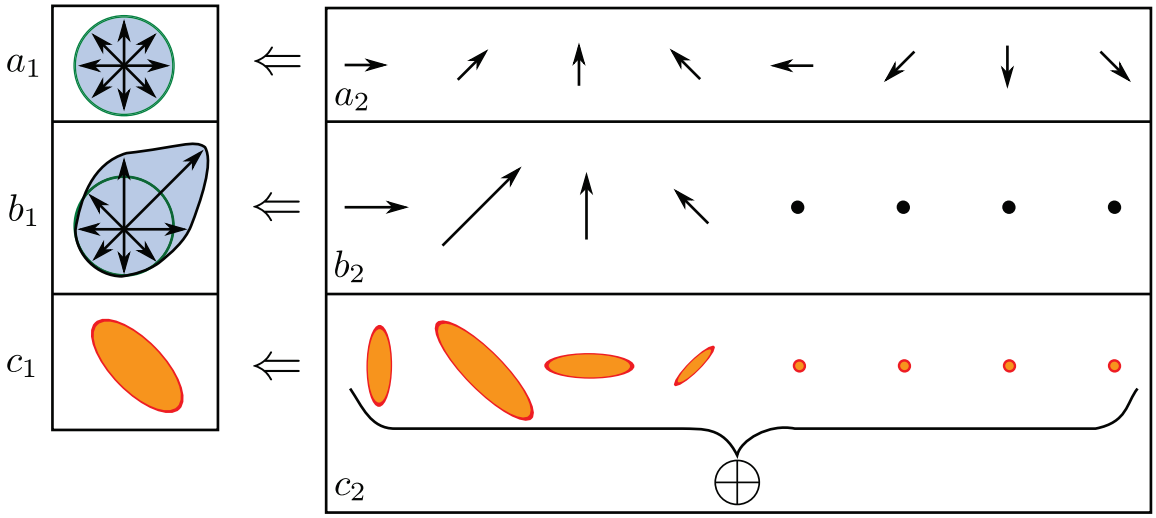


Figure 4.13: *Visual procedure for computing the continuous gradient histogram and the diffusion tensor of Figure 4.12 at point P. For simplicity, we create a histogram, a_1, of eight directional bins. They are separated and spread in a_2. b_2 illustrates the contribution of the distributed gradients within the red circle; i.e. the feature size, to each of the directional bins in a_2. The contributed amount is estimated by the length of the arrow. The four rightmost directional bins have zero contribution as they are directed away from the gradients. b_1 displays the overall continuous histogram of b_2. The diffusion tensors relative to their directional bins are shown in c_2. They are orthogonal to their respective directional bin and scale to their contributed gradient. As such, the four rightmost diffusion tensors are scaled to nearly zero. c_1 is the diffusion tensor resulted from the geometric mean of the eight diffusion tensors in c_2. Note that in our code, we create a histogram of 256 directional bins scattered uniformly on the unit circle.*

---

two eigenvectors and eigenvalues respectively. $H(I) = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$ is the Hessian matrix of image I. The Hessian matrix is used as a quadratic form to measure the directional second derivatives along the vectors V_i. I_t is iteratively added to I by a time step $\Delta t = 0.2$:

$$I(x, y, t + \Delta t) = I(x, y, t) + \Delta t(I_t). \tag{4.16}$$

4.4 Implementation

We compute the diffusion tensors and perform anisotropic diffusion almost entirely on GPU using NVIDIA'S CUDA model [Sanders 2010]. Computing diffusion tensors and performing anisotropic diffusion can be categorized as *embarrassingly parallel*

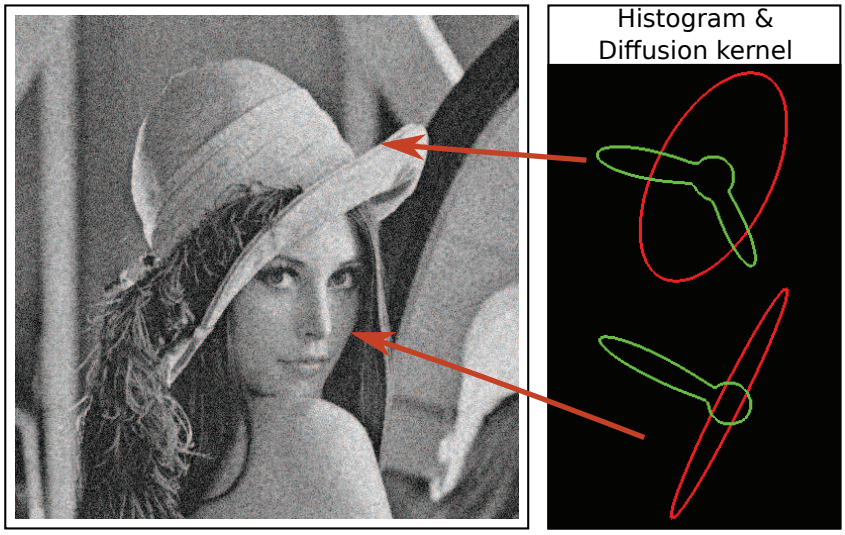


Figure 4.14: *Continuous local gradient histogram with their respective diffusion tensors at distinct locations. The green colored loop represents the histogram and the red represents the diffusion tensor. The top figure shows a histogram computed over a region containing a junction resulting with two dominant directions. Its respective diffusion tensor results in a elliptic shape. The bottom figure shows a histogram computed over a region containing an edge resulting with one dominant direction. Its respective diffusion tensor is flat and orthogonal to the dominant direction.*

---

problems since we process blocks of pixels at a single time step. See Figure 4.15 for the pseudo code, and Figure 4.16 for an overview of the graphics pipeline.

- First, we blur the image using a convolution with g_s, as in (Equation 4.12), using two Fast Fourier Transforms (FFT). We transform f into the Fourier domain (FFT-forward), multiply the spectrum by the spectrum of g_s, and finally transform back into the primal using the second FFT (FFT-inverse). We use the FFT standard function provided by NVIDIA's CUDA software development kit.

- For each histogram direction ω_i, and for all pixels x at once, we compute the directional histogram $h_{\omega_i}(x)$ using (Equation 4.9). This involves two FFT calls and a product in Fourier space.

- As we compute histogram values, we accumulate the inverse of the geometric diffusion matrix M_g for each pixel in a buffer using Equation 4.13. This operation is also done in one time step.

- Finally, we invert the matrices for all pixels and use them for smoothing the volume using diffusion.

for all pixels x **do**
  init tensor $M(x)$ to 0
compute $g_s \otimes f$ using 2 FFT + product
for all histogram directions ω_i **do**
  for all pixels x **do**
    compute $K_{\omega_i}(\nabla(g_s \otimes f))$
  convolve with g_s using 2 FFT + product
  for all pixels x **do**
    accumulate $h_{\omega_i}(x)M_g(\omega_i)^{-1}$ into $M(x)^{-1}$
for all pixels x **do**
  invert $M(x)^{-1}$
  diffuse at x according to $M(x)$

Figure 4.15: *Pseudocode for the computation of diffusion tensors in the 2D image. All operations are done on the GPU, using CUDA.*

Algorithms 4.1 and 4.2 sketch in details the algorithms for computing the diffusion tensors of 2D images and performing anisotropic diffusion. In this implementation, directional gradient histograms are never stored explicitly, their values are used directly to compute the integral in Equation 4.13. We discretize the directions into 256 directions scattered uniformly on the unit circle.

The GPU memory footprint of our algorithm is proportional to the number of pixels in the dataset. The largest GPU memory cost needed at once during our algorithm execution is $52 \times n^2$ bytes; where n is the number of pixels in the dataset. It is composed of:

- intensity or density value and cumulated histogram weight (2 floats),
- cumulated diffusion tensor (3 floats),
- Cuda-FFT plan (8 floats per pixel for precomputed cosines)

This memory footprint can be halved at the expense of a little more computation cost, if using an explicit computation of cosines in the FFT algorithm.

4.5 Results and Comparison

We tested our algorithm on several different images:

- Lena(Figure 4.17). This famous image was taken by a camera. It consists of curves, edges, junctions, and small details such as the feathers on the rear of the hat. To test our algorithm, we added to the picture random noise of high frequency.

Algorithm 4.1: *Computing 2D Diffusion Tensors for images on GPU*

input : f, 2D noisy data. s, feature size.
output: M, vector of diffusion tensors

`// Initialize tensors` $M(x)$
forall *pixels* x **in** f **do**
  $M(x) \leftarrow 0$

`// Initialize continous histogram`
$\Omega \leftarrow$ **new** `histogram()`

`// Blurring` f `using Gaussian` $g_{\frac{s}{2}}$ `and CUDA FFT (using Eq. 4.12).`
$F \leftarrow$ `CudaFFT(`f, *FFT-forward*`)`
$F \leftarrow F \times g_{\frac{2}{s}}$
$f^b \leftarrow$ `CudaFFT(`F, *FFT-inverse*`)`

`// Computing directional histogram`
forall *histogram directions* ω_i **in** Ω **do**
  forall *pixels* x **in** f **do**
    $h_{\omega_i}(x) \leftarrow K_{\omega_i}(\nabla f^b)$

  `// convolve wih` g_s `using 2 CUDA FFT (using Eq. 4.9).`
  $H_{\omega_i} \leftarrow$ `CudaFFT(`h_{ω_i}, *FFT-forward*`)`
  $H_{\omega_i} \leftarrow H_{\omega_i} \times g_{\frac{1}{s}}$
  $h_{\omega_i} \leftarrow$ `CudaFFT(`H_{ω_i}, *FFT-inverse*`)`

  `// Accumulate the diffusion tensors for each pixel using`
  `// Eq. 4.13`
  forall *pixels* x **do**
    $V \leftarrow \begin{bmatrix} \omega_i & v_i \end{bmatrix}$
    $\varepsilon \leftarrow \frac{1}{1000}$
    $M_g(\omega_i) \leftarrow V^T \begin{bmatrix} \varepsilon & 0 \\ 0 & 1 \end{bmatrix} V$
    `// Computing geometric mean`
    $M(x)^{-1} \mathrel{+}= h_{\omega_i}(x) M_g(\omega_i)^{-1}$
    $\mathcal{H}(x) \mathrel{+}= h_{\omega_i}(x)$

`// Normalize the tensors` $M(x)^{-1}$ `and inverse to` $M(x)$
forall *pixels* x **do**
  $M(x)^{-1} \leftarrow \frac{M(x)^{-1}}{\mathcal{H}(x)}$
  $M(x) \leftarrow$ `invert(`$M(x)^{-1}$`)`
`return` M

- An MRI image of a knee (Figures 4.18 and 4.19). A knee is made from a mixture of skins (on the surface), muscle tissues, bones, cartilages, veins, and

Algorithm 4.2: *Computing 2D Diffusion on GPU*

input : f, 2D noisy data. M, vector of diffusion tensors
output: f_s, smooth 2D data at feature size s.

$f_s \leftarrow$ **new** 2D-data()
$\nabla t \leftarrow 0.2$
for n *diffusion steps* **do**
  forall *pixels* x **do**
    $[V_1, V_2] \leftarrow$ `eigenvectors` $(M(x))$
    $[\lambda_1, \lambda_2] \leftarrow$ `eigenvalues` $(M(x))$
    $f_t \leftarrow \sum_{i=0}^{2} \frac{(V_i^T H(f(x)) V_i)\lambda_i^2}{V_i^T V_i}$
    $f_s(x) \leftarrow f(x) + \nabla t f_t$
  $f \leftarrow f_s$
`return` f_s

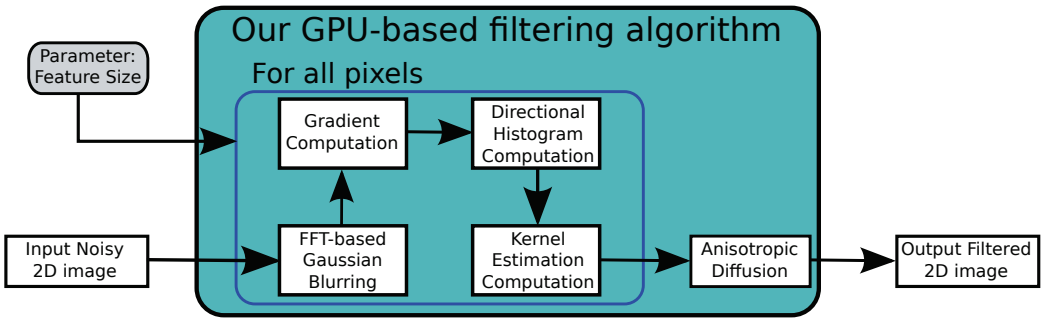


Figure 4.16: *GPU computation pipeline. All computations are done on the GPU. The input are the original volume and the feature size.*

ligament tissues with different density and thickness. This produces different permeability to magnetic resonance machines. Thus, it results in a large range of contrast. The image is made up of density values.

- An MRI image of a brain (Figure 4.20). The dataset is made of soft tissues and bones.

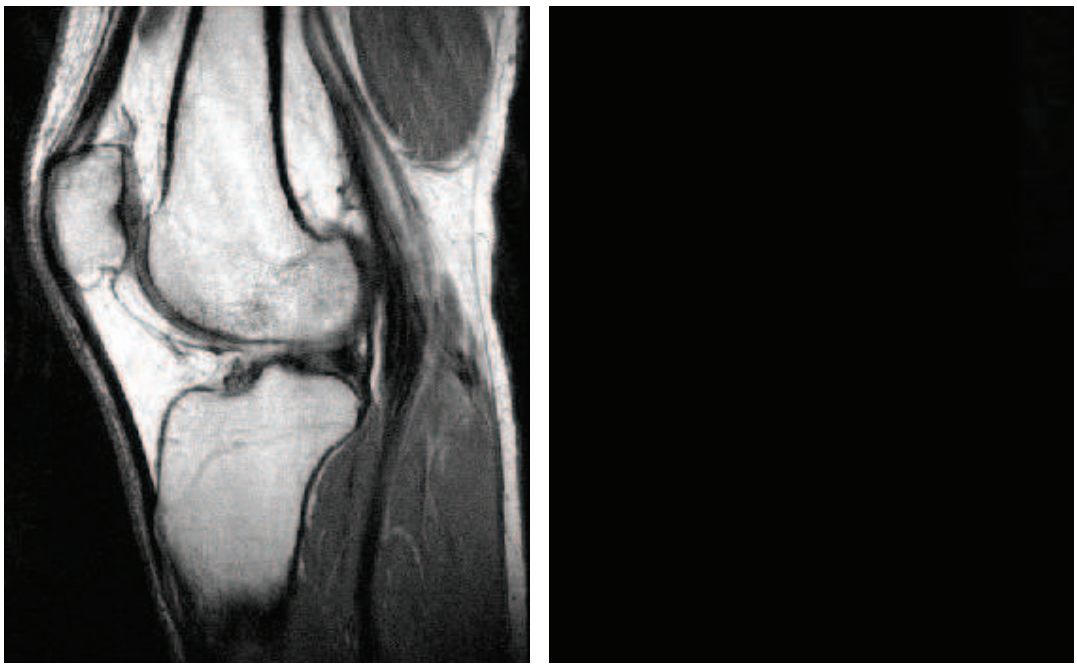
- A scanned text (Figure 4.21). A text consists of mainly edges. Texts, written or printed, deteriorate over time introducing cracks and noise in the text. To test our algorithm on text, we added random noise to it.

(a) (b)

Figure 4.17: *Image of Lena (*512^2 *pixels): (a) random noise is added to the image and (b) the image after smoothing by our algorithm at feature size* $s = 5$ *and* 40 *diffusion steps (processing time* $= 0.26$ *sec). Note how our algorithm reduces the noise while preserving the sharp and detail features such as the junctions, curve of the hat and the feathers on the hat.*

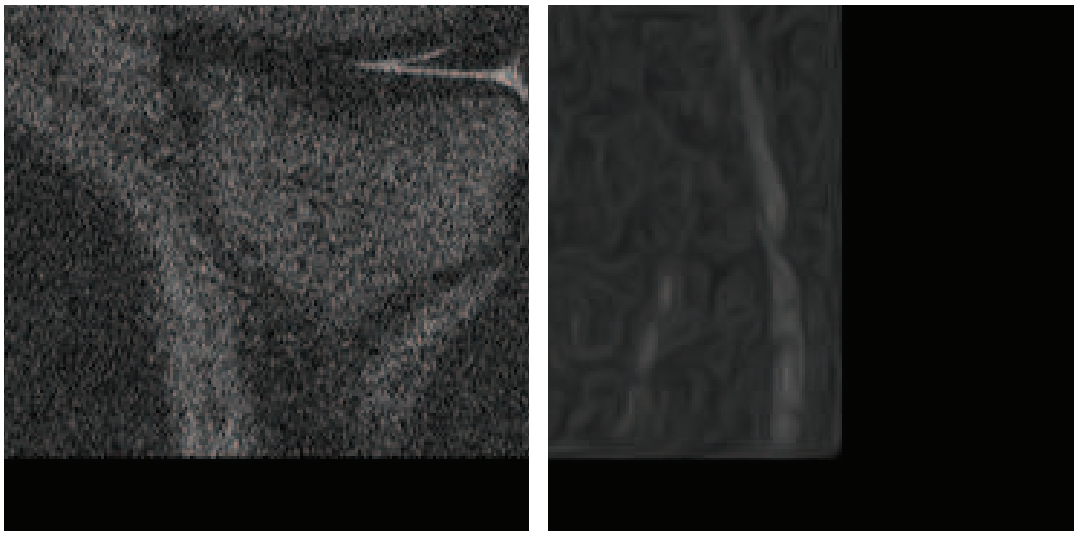
4.5.1 Noise and Preservation of Sharp Features

Noise is clearly visible on all five images. See images (a) of Figures 4.17, 4.18, 4.19, 4.20, and 4.21. Our algorithm smoothes out this noise while preserving the sharp features. Figures 4.17(b), 4.18(b), 4.19(b), 4.20(b), and 4.21 show the images smoothed by our algorithm. The reduction of the noise is clearly visible as well as the preservation and enhancement of the sharp and detail features. In the image of Lena the edges and feathers are well preserved (see Figure 4.28 for close up views). Our algorithm also closes cracks and makes it easier to identify and separate different components in the image. For example, in the smooth MRI knee (Figure 4.18(b)) the cracks on the skin are closed, the bones and muscle tissues are sharper and clearer. Similar results are achieved in Figure 4.19(b). The sand like noise is reduced and reveals the knee components without losing its main components and sharp details. In the brain image, the soft tissue has been reconstructed from its noisy dataset making it easier to identify the different parts of the brain. The noise in the scanned text has significantly reduced while preserving the edges of the letters. Moreover, the cracks within the letters are closed and the text is sharper.



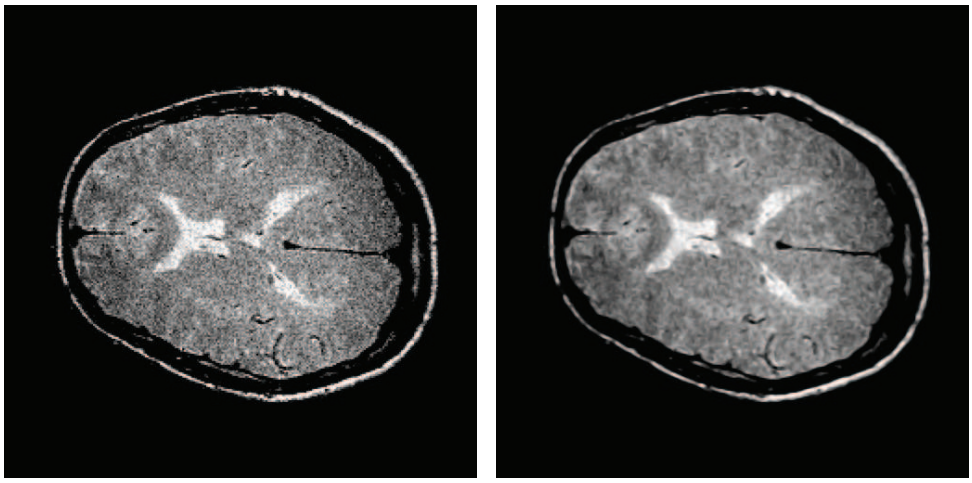
(a) (b)

Figure 4.18: *MRI of a knee (*512^2 *pixels): (a) a noisy MRI of a knee and (b) the image after smoothing by our algorithm at feature size* $s = 2$ *and* 20 *diffusion steps (processing time* $= 0.26$ *sec). Note how the crack on the skin are closed. Moreover, the bones and muscle tissues are sharpen and clearer. This filtered image has not been accepted medically.*



(a) (b)

Figure 4.19: *MRI of a knee (*512^2 *pixels): (a) original MRI, it is full of sand like noise. (b) The image after smoothing by our algorithm. We processed (a) at feature size* $s = 10$ *and* 40 *diffusion steps. This filtered image has not been accepted medically.*



(a) (b)

Figure 4.20: *MRI of a brain (*512^2 *pixels): (a) original MRI brain and (b) the brain image after smoothing with our algorithm. It is easier to identify the soft tissue of the brain after reconstruction. We processed (a) at feature size* $s = 2$ *and* 24 *diffusion steps. This filtered image has not been accepted medically.*

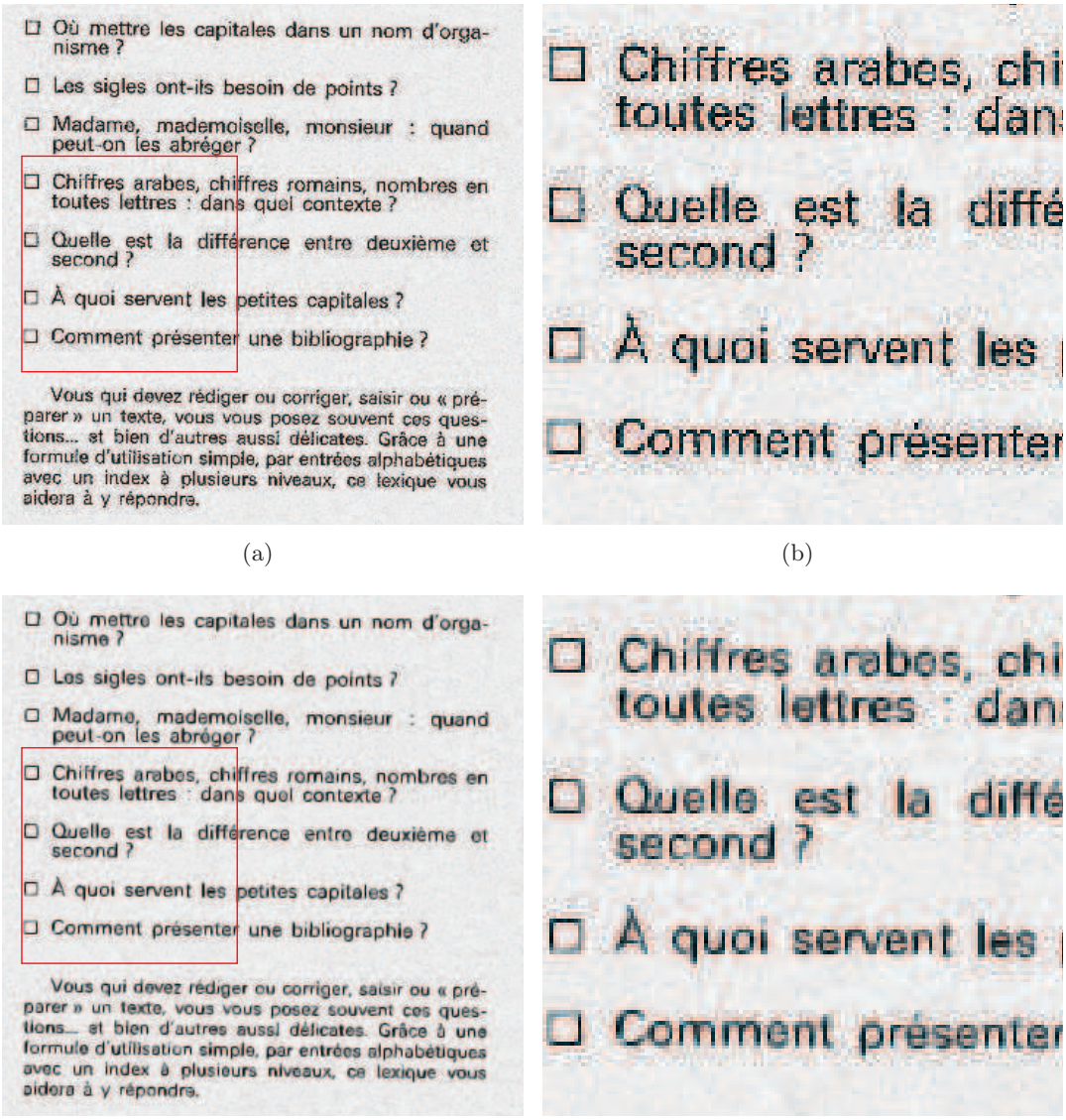
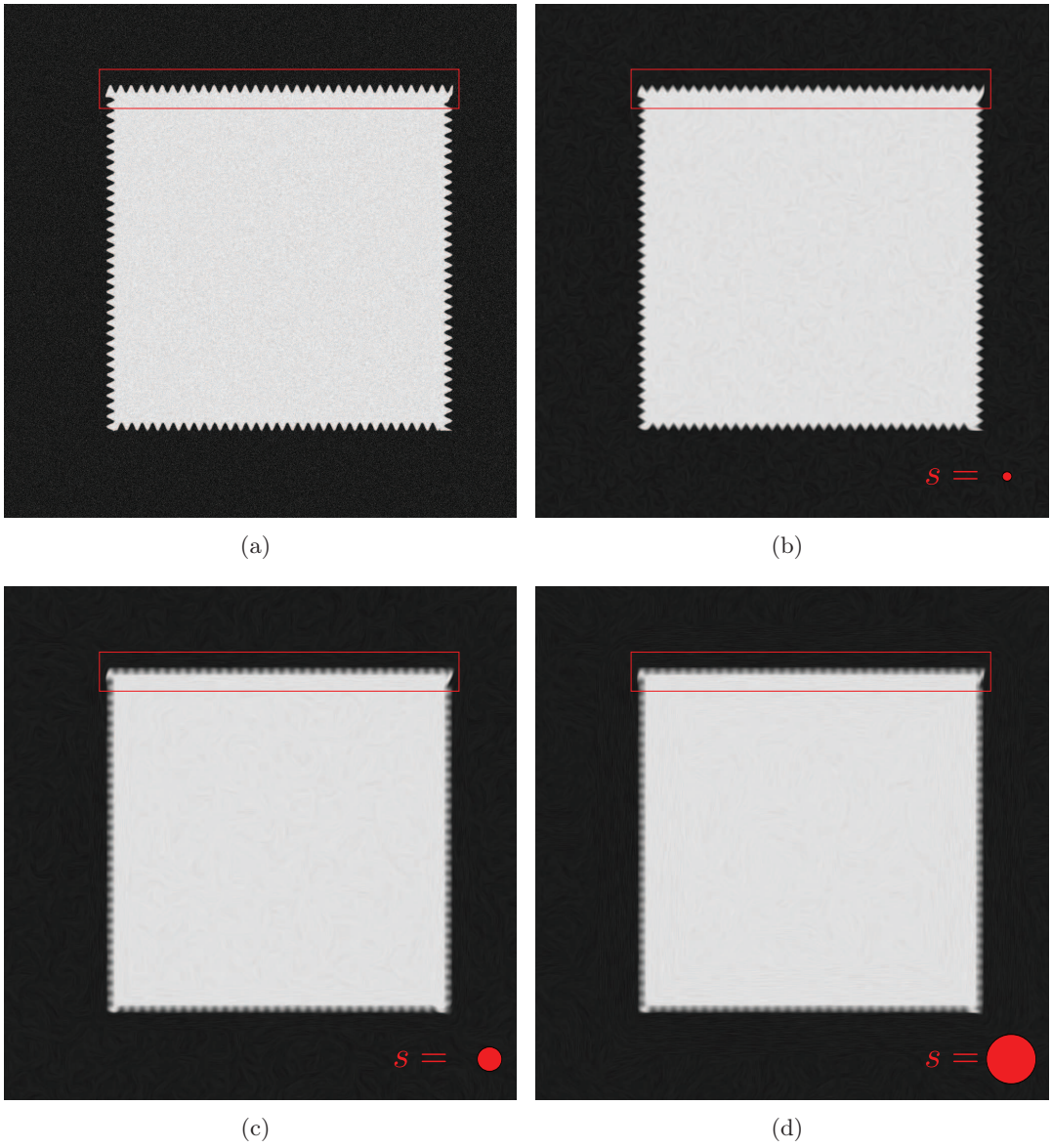


Figure 4.21: *Scanned text* (1024^2 *pixels): (a) original text scanned with noise added, (b) the text after smoothing with our algorithm. We processed (a) at feature size* $s = 5$ *and* 24 *diffusion steps (processing time* $= 0.67$ *sec). Note how our algorithm smoothes the noise while preserving the letters. Moreover, it fills in the cracks and enhances the visibility of the letters.*



(a) (b)

(c) (d)

Figure 4.22: *A square encircle by a wave feature (*1024^2 *pixels) at various feature sizes: (a) original data, (b), (c) and (d): filtered data with increasing feature size of* 20, 50, *and* 100 *respectively. With a feature size of 20, our algorithm removes the noise but preserves the waves feature. When we increase the feature size to* 100, *our algorithm blurs the waves, but keeps the contour of the square structure.*

4.5.2 Feature Size

The key parameter of our algorithm is the size of the feature we are interested in. Filtering removes details smaller than this feature size, while preserving surface

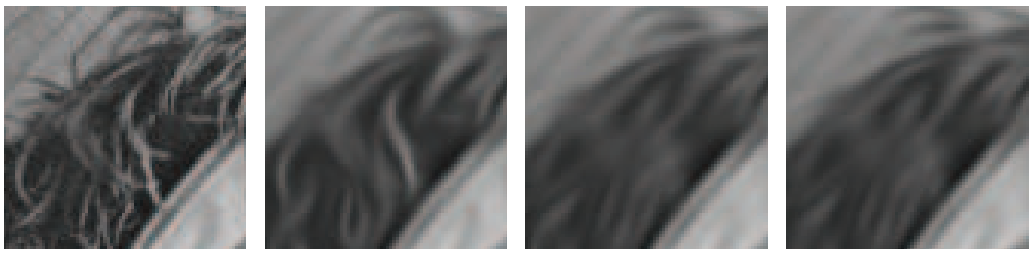
(a) (b)

(c) (d)

Figure 4.23: *Image of Lena (*512^2 *pixels) at various feature sizes: (a) original data, (b), (c) and (d): filtered data with increasing feature size of* 20, 50, *and* 100 *respectively. Note how our algorithm smoothes out details of the feathers on hat as the feature size increases while the preserving the edges greater than the specified feature size.*

details that are larger.

In Figures 4.22 and 4.23, we show the effect of increasing the feature size on a wave structure and on the detail features of the image of Lena. For a small value of the parameter, our algorithm smoothes while preserving the wave structure on the



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

Figure 4.24: *Close-up views of the feathers on the hat Lena from its respective (a-d) images of Figure 4.23 at feature size* 20, 50, *and* 100. *As we increase the feature size, the details of the feathers are removed.*

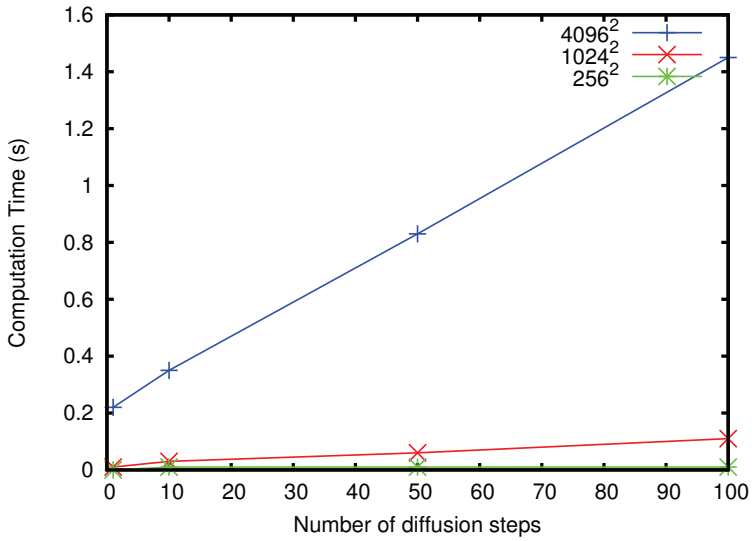


Figure 4.25: *Computation time on the GPU (in seconds) as a function of the number of diffusion steps, for different size of the dataset.*

square for a large number of diffusion steps. The same is true on the features of Lena most notably the feathers on the hat. As we increase the feature size, our algorithm smoothes out these features. Figure 4.24 shows close-up views of the feathers as the feature size increases. On the image of Lena, note that the edges larger in size than the feature size are preserved such as the edges on the hat, face, and background structures.

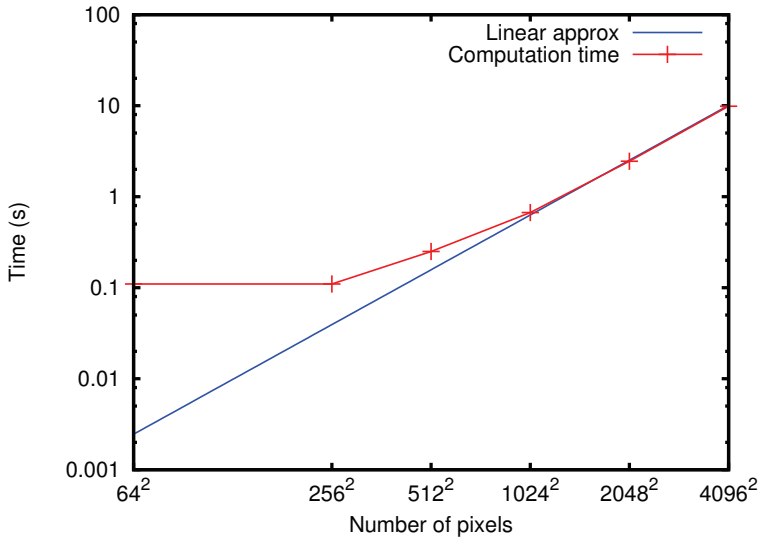


Figure 4.26: *Logscale plot of the processing time (in seconds) of the diffusion tensors, as a function of the number of pixels. The computation time increases linearly with the increase of the number of pixels.*

4.5.3 Computation Time

Computation times for our algorithm depend on two parameters: the number of pixels in the dataset and the number of diffusion steps. It does not depend on the feature size.

The number of diffusion steps has a minimal impact on computation time. Figure 4.25 displays the computation time for 1, 10, 50, and 100 diffusion steps, of different dataset sizes. The time measurements include a constant initializing time, which explains the afine—rather than linear—behavior of the curve. After this first step, computation time increases linearly with the number of steps, as expected. Note how fast it is. It takes only *1.5 sec.* to execute 100 diffusion steps on an image made of 4096^2 pixels ($\sim$ 17 million pixels).

4.5.4 Scalability

The computation time of the diffusion tensors increases linearly with the total number of pixels n^2. Figure 4.26 shows the computation time as a function of the total number of pixels. It takes $\sim$ 10 *sec.* to compute the diffusion tensors of a dataset of 4096^2 pixels.

The memory footprint of our algorithm is equal to $52n^2$. On a graphics card with 1 GB of memory, roughly 354 MB are reserved by the card itself for its own use, leaving 669 MB available for our algorithm. This means that the maximum size for a dataset to be processed in a single chunk is roughly 4096^2 corresponding

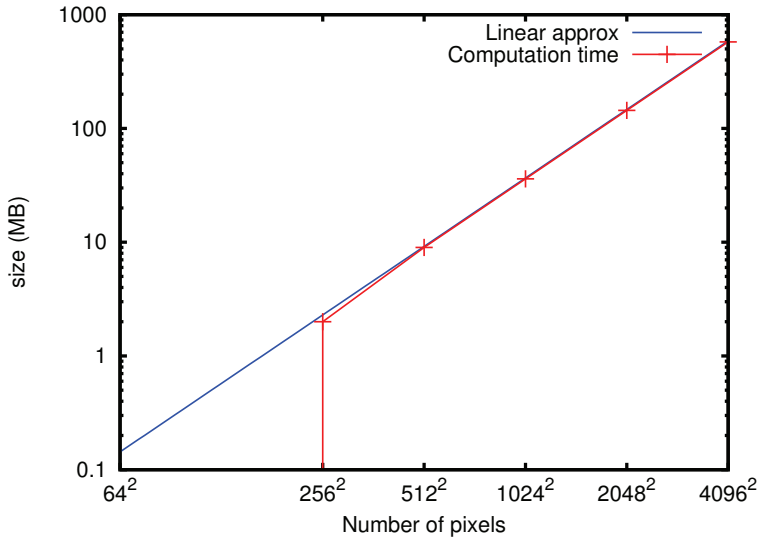


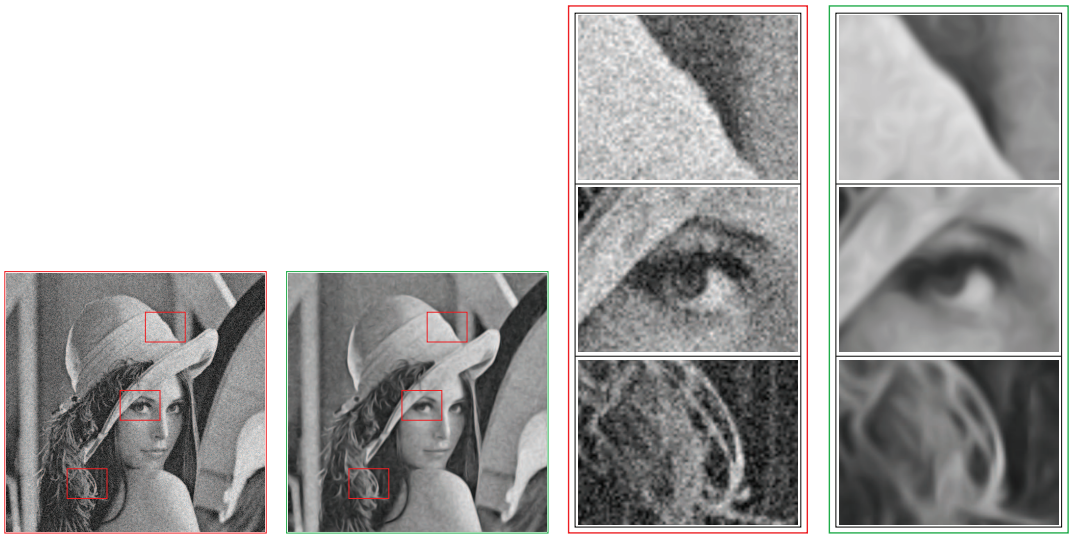
Figure 4.27: *Logscale plot of the memory size (in MB), as a function of the number of pixels. Datasets of size below* 256^2 *pixels take very little memory space. Above this size, the memory space requirement increases linearly with the increase of the number of pixels.*

---

to 576 MB of memory.

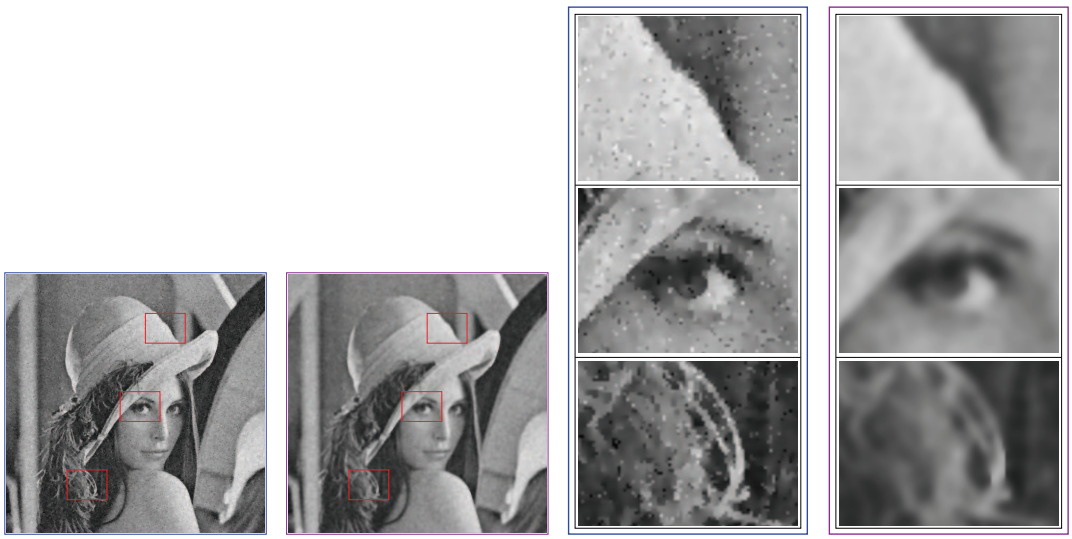
Datasets that are larger than this limit can be segmented into smaller pieces, which are processed separately by the GPU, before we combine the results together. For a continuous stitching, we keep an overlap between the data pieces. Using this division and stitching method, we can process datasets of arbitrary size.

Figure 4.27 shows a plot of the memory size requirement in *MB* as a function of the number of pixels. Our algorithm requires very little memory space for datasets of size below 256^2 pixels. Above this size, the memory requirement of our algorithm increases linearly with the increase of the number of pixels in the dataset.



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

Figure 4.28: *Image of Lena (512^2 pixels): (a) high frequency noise is added to the image, (b) the image after filtering with our algorithm ($s = 5$, diffusion steps $= 40$). (c) and (d) close-up on specific details from (a) and (b).*



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

Figure 4.29: *Image of Lena (512^2 pixels) using the Perona-Malik algorithm [Perona 1990], with different parameter values: (a) $k = 0.1$, diffusion steps $=$ 10, (b) $k = 0.3$, diffusion steps $=$ 10. (c) and (d) close-up on specific details from (a) and (b).*

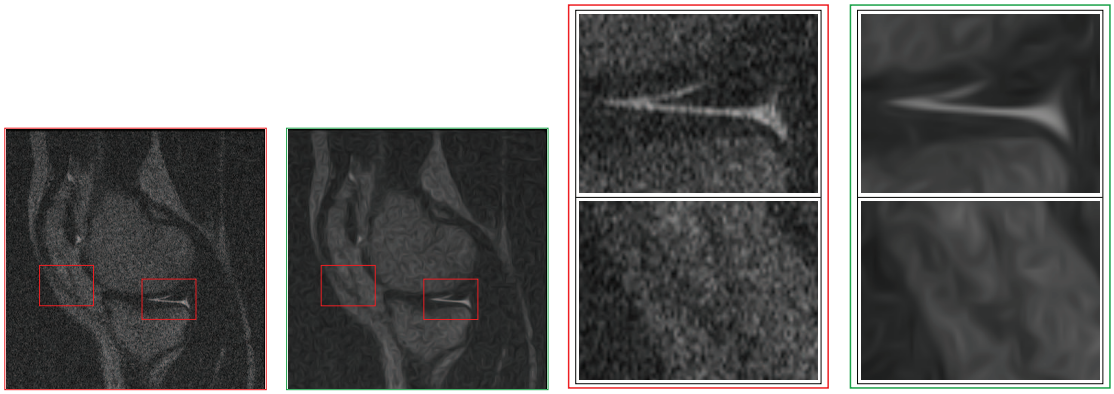
(a) (b) (c) (d)

Figure 4.30: *Image of Lena (512^2 pixels), using bilateral filtering with different parameter values: (a) $\sigma_d = 2$, $\sigma_r = 1.5$, (b) $\sigma_d = 2$, $\sigma_r = 0.48$. (c) and (d) close-up on specific details from (a) and (b).*

4.5.5 Comparison with Existing Algorithms

For comparison, we implemented the Perona-Malik [Perona 1990] and bilateral filtering [Tomasi 1998] algorithms. We tested these algorithms on the noisy image of Lena (Figure 4.17(a)) and the MRI of a knee (Figure 4.19(a)). Figures 4.29 and 4.30 show the results of these algorithms on the image of Lena dataset compare with Figure 4.28 for our own algorithm on the same dataset. Both algorithms exhibit the same behavior: for some values of the parameters, they keep the edges and features, but they also keep some of the noise. This is seen in the close-up views of Figures 4.29(c) and 4.30(c) where grain like and crystallize noise remain. If we push the parameters until the point where the noise disappears, we start losing features and details. This is better illustrated in the close-up view of the feathers on the hat of Lena (see Figures 4.29(d) and 4.30(d)) where they are blurred way compare to our own algorithm shown in Figure 4.28(d). We observe similar results on the MRI of the knee shown in Figures 4.32 and 4.33 compare with Figure 4.31 for our own algorithm. For each algorithm, we display the best results we could find.

Bilateral filtering was also implemented on the GPU, resulting in computation times similar to those of our algorithm. Please refer to Table 4.1 for computation time (in seconds), for several values of the parameters and image sizes. It is faster than our algorithm when σ_d is reasonably small. However, as σ_d increases so does



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

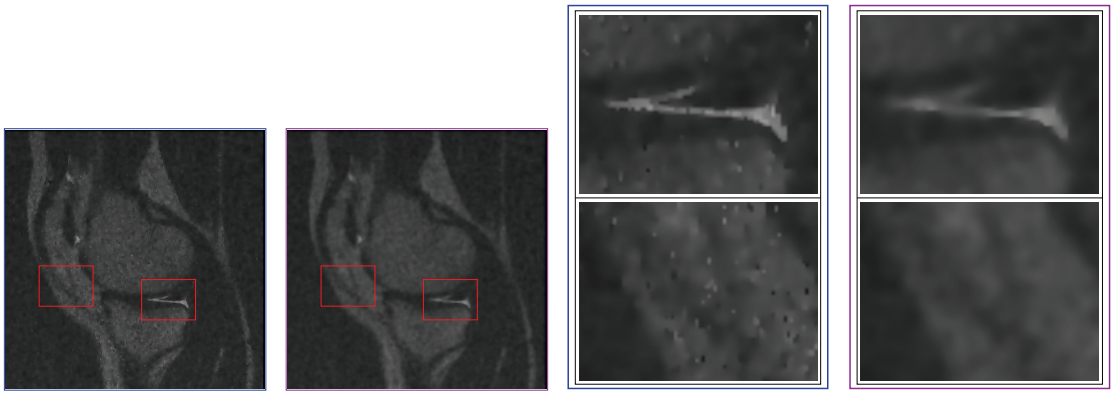
Figure 4.31: *MRI of a knee* (512^2 *pixels): (a) original image, (b) the image after filtering with our algorithm* ($s = 10$, *diffusion steps* $= 40$). *(c) and (d) close-up on specific details from (a) and (b).*

---

its computation time which can become slower than ours.

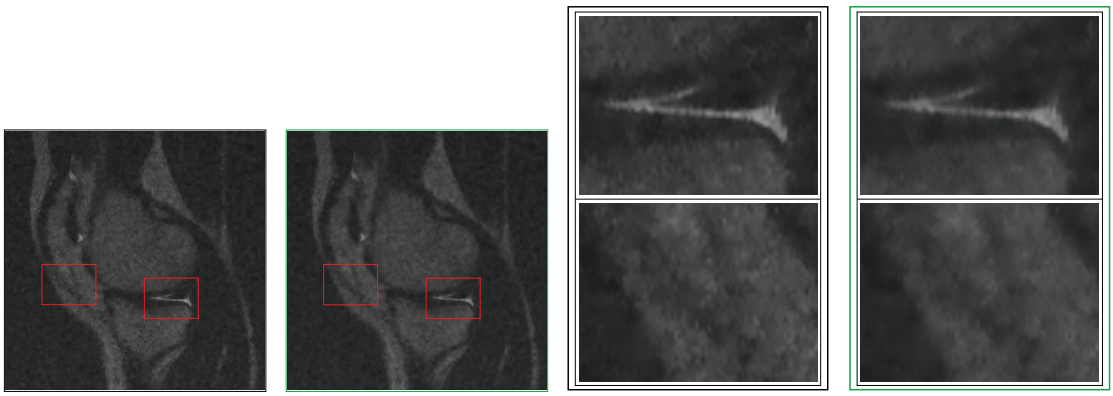
We did not implement the EED and CED methods for comparison because of the lack of time. Nevertheless, based on the thorough analysis we provided in Chapter 3, we can give a theoretical comparison. Our method adapts the diffusion tensors to any shape and diffuses accordingly to preserve the sharp features. We reconsider the noise square of Figure 3.20 to illustrate that our method behaves as expected (shown in Figure 4.34) at P_a, P_b, and P_c and compare with that of the behavior of EED and CED.

At P_a, our diffusion tensor is a circle; *i.e.* diffusion is carried isotropically as is required to smooth out the noise. At P_b, no or very little diffusion is applied because of the presence of a junction of edges. Finally at P_c, we diffuse only along the y-axis because our diffusion tensor is an ellipsoid parallel to the y-axis. Unlike our method, EED and CED behave differently and does not filter the noise. EED works well on regions with and without edges but not on corners (see Figure 3.20(b)) and CED works well on regions with edges and corners but performs poorly on edge free regions; filtering is very slow (see Figure 3.20(a)).



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

Figure 4.32: *MRI of a knee (*512^2 *pixels), using Perona-Malik, with different parameter values: (a)* $k = 0.08$*, diffusion steps* $= 10$*, (b)* $k = 0.15$*, diffusion steps* $= 10$*. (c) and (d) close-up on specific details from (a) and (b).*



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

Figure 4.33: *MRI of a knee (*512^2 *pixels), using bilateral filter, with different parameter values: (a)* $\sigma_d = 2$*,* $\sigma_r = 0.11$*, (b)* $\sigma_d = 2$*,* $\sigma_r = 0.15$*. (c) and (d) close-up on specific details from (a) and (b).*

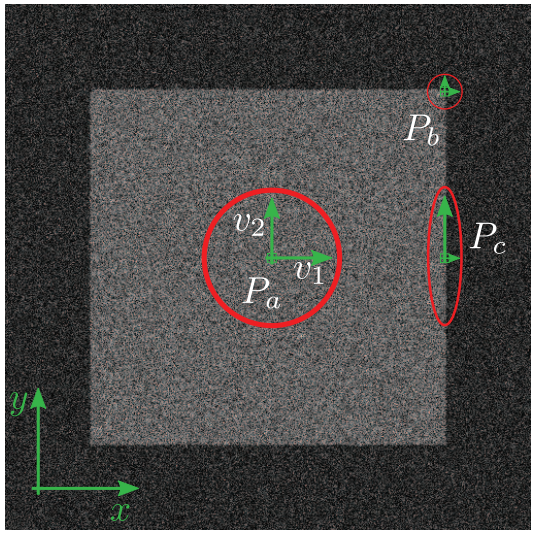


Figure 4.34: *The diffusion orientation of our method at specified regions; P_a, P_b, and P_c, of a noisy square image is illustrated by the respective arrows on location. $< v_1, v_2 >$ are the eigenvectors. The length of the arrows expresses the emphasis and amount of applied diffusion; longer arrow signifies more diffusion. The red circle and ellipse represent the diffusion tensors. Our method adapts well to the different regions as expected compare to the EED and CED methods (see Figure 3.20).*

Parameter Values	**Image Size**		
	512^2	1024^2	4096^2
$\sigma_d = 2$, $\sigma_r = 0.11$	0.11 s	0.16 s	1.02 s
$\sigma_d = 4$, $\sigma_r = 0.15$	0.14 s	0.27 s	1.97 s
$\sigma_d = 8$, $\sigma_r = 2$	0.28 s	0.71 s	11.27 s
$\sigma_d = 12$, $\sigma_r = 2$	0.49 s	1.62 s	$\sim$ 65 s

Table 4.1: *Computation time for 2D bilateral filtering (in seconds), on the GPU, as a function of image size and parameter values.*

4.6 Analysis and Limitations

As shown along this chapter, our technique works well on various types of noisy images. Our diffusion tensor adapts effectively and efficiently to the shapes of the image content and filters the noise below the specified feature size while preserving the sharp details. In addition to, we do so very fast as we take advantage of the massively parallel processing power of the GPU. Its single-instruction-multiple-data (SIMD) architecture [Flynn 1972] allows our filtering method to process blocks of pixels simultaneously at a time step. Compared to a CPU implementation, the GPU has the upper hand by roughly a factor of 1000. For instance, filtering an image

of size 1024^2 pixels using our method on CPU would take $\sim 428 secs.$ compared to $0.8 secs.$ on the GPU.

During our testing, we witnessed a single major limitation in our two-dimensional smoothing method. Choosing a large feature size to smooth out large details in a very noisy image create a smoke or flow like artifact in the image; specially in homogeneous regions. Figure 4.35 shows such artifact on the Lena image after smoothing its noisy image (Figure 4.17(a)) at feature size 100. The explanation behind such a behavior is simple. For a large feature size value, we compute a diffusion tensor that account for the majority of gradients in the area covered by the feature size. In a dense noisy image, the method may identify patterns in the noise and misinterpret them for sharp details and attempts to preserve them. As we increased the diffusion steps on the image, we noticed a reduction of this artifact but at the cost of maybe smoothing real details on the image. From another point of view, we can use this artifact as an artistic effect on an image. Note that such a phenomenon is created or observed by the CED method [Weickert 1994, Weickert 1997, Weickert 1999b, Weickert 1999a].

In the next chapter, we present our three-dimensional method for filtering volumes where the method performs better. Moreover, we do not find any of the behaviors observed and described above.

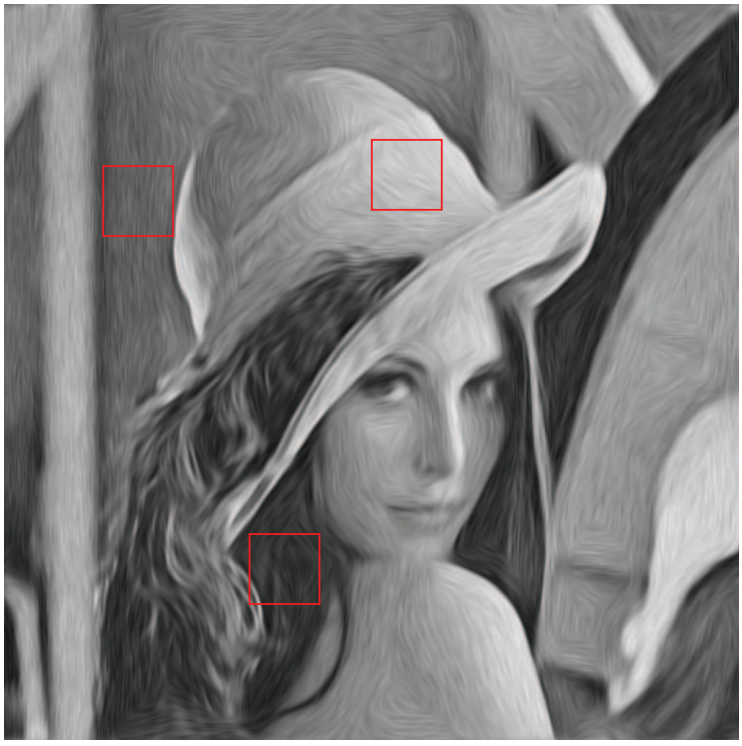


Figure 4.35: *Smoke like artifact on the Lena image* (512^2 *pixels) after smoothing its noisy image (Figure 4.17(a)) with our method at feature size* 100.

Chapter 5

Fast Multi-Scale Feature-Preserving Smoothing of Volumetric Data

Contents

5.1 Introduction

In this chapter, we extend our multi-scale smoothing algorithm presented in Chapter 4 from 2D images to 3D images (volumes). We address the problem of smoothing noisy tomographic reconstructions to improve the quality of level set surfaces that are extracted from the 3D datasets. As with 2D images, our algorithm adapts the

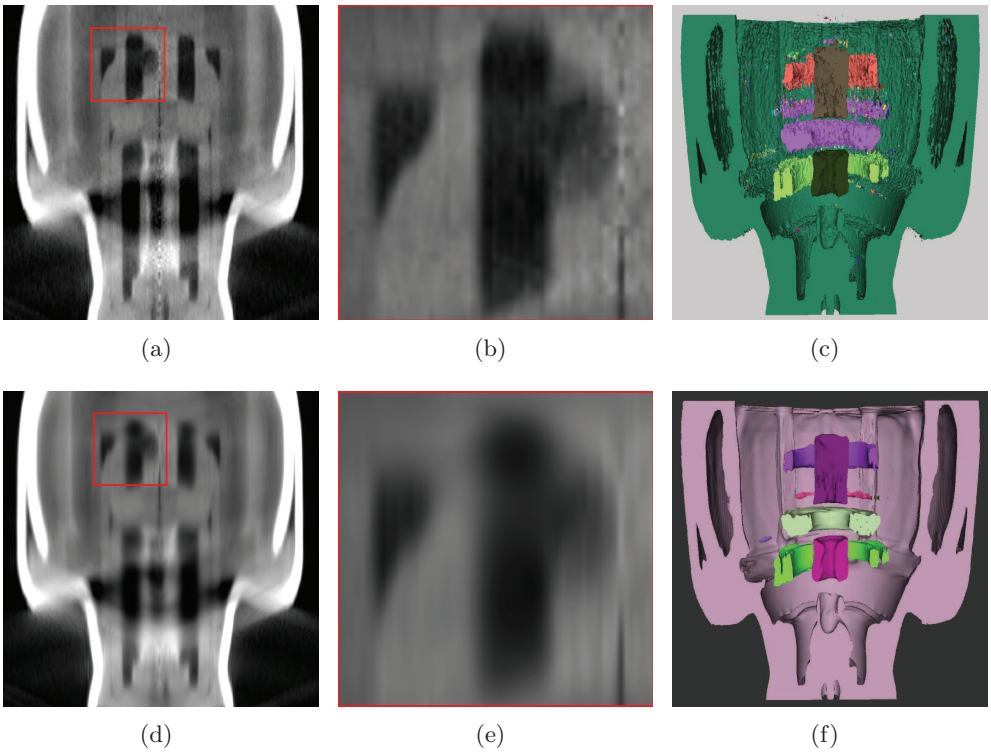


Figure 5.1: *Slices of (a) the original CT reconstruction of the inhaler model, (b) close-up view of an area of (a) squared in red and (c) the extracted surface of (a). (d) CT slice of the inhaler model after smoothing by our method, (e) close-up view of the smooth CT (d) square in red, and (f) extracted surface from smoothed data. We raised the gamma level in the CT images to better visualize the details of the volumes.*

---

smoothing kernel to local features. The result is a smooth volume density field where we cancel high frequency variations below a user-specified threshold. Our algorithm consistently preserves sharp features such as edges and corners. It runs efficiently on parallel processors (such as GPUs) and is scalable to large datasets. We use the feature size s parameter to tune the level of features to preserve.

Our algorithm works in the following way: First, we compute 3D *local continuous histograms* of gradients of the volume density field. Then we compute diffusion tensors using these histograms. Finally, we apply *anisotropic diffusion* in the 3D data using the computed tensors. Since the diffusion at each voxel is guided by the local distribution of gradients, the smoothing respects local features. These operations are done in parallel, on the GPU, for entire blocks of voxels. We demonstrate that using dual contouring to extract level-set surfaces from our filtered data results in high-quality surfaces with sharp feature (such as edges and corners).

Introduction

Dans ce chapitre, nous prolongerons notre algorithme de lissage multi-échelle présenté dans le Chaptire 4 de l'espace 2D à l'espace 3D pour lisser des volumes 3D. On s'intéresse au problème de lissage de données tomographiques bruitées afin d'améliorer la qualité des surfaces extraites de celles-ci. Comme dans les images, notre algorithme adapte le noyau de lissage aux caractéristiques locales. Le résultat est un champs de densités lisse où les variations de hautes fréquences en-dessous d'une valeur spécifiée sont annulées. Notre algorithme tourne efficacement sur processeur parallèle (comme les GPUs) et peut être appliqué à des données plus larges. Le réglage est relativement facile, avec un seul paramètre que l'on appelle *taille caractéristique*—la taille minimale des caracteristiques que l'on souhaite préserver.

Notre algorithme fonctionne de la façon suivante: dans un premier temps, nous calculons les *histogrammes continus locaux* 3D des gradients du volume de champs de densités. Ensuite, nous calculons les tenseurs de diffusion en utilisant ces histogrammes. Finalement, nous appliquons la *diffusion anisotrope* sur les données 3D en accord avec leurs tenseurs obtenus. Le lissage respecte les caractéristiques locales puisque la diffusion à chaque voxel est guidée par la distribution locale des gradients. Nous démontrons des résultats de surfaces de haute qualité extraits des données 3D lissées qui néanmoins préservent arrêtes et coins.

5.2 Volumetric Data and Noise

Volumetric scans of objects are ubiquitous in medical imaging, engineering and analyzing cultural heritage. Computer tomography is a class of techniques that produce such volumetric representations of objects by combining a large series of 2D scans, captured from multiple views. Typically, penetrative radiation is used to obtain each 2D scan: X-Rays for CT (computer tomography) scans, radio-frequency waves for MRI (magnetic resonance imaging), electron-positron annihilation for PET (positron emission tomography) scans and so on. The process used to amalgamate multiple 2D scans to produce a single voxelized density field is called *tomographic reconstruction*.

Unfortunately, the voxelised density data is typically polluted by noise from a variety of sources: the limited number of views [Herman 2009], lack of captor sensitivity, high contrasts, non-monochromaticity of the X-Ray source, imperfect stability of the X-Ray source, pixel defects on the captor and non-uniformity of the absorption process across wavelengths. The constraint that data acquisition should be noiseless is unrealistic. We want to reduce, or eliminate, noise as early as possible in the application pipeline.

The scanned volume is typically explored as 2D slices (images) of the density field, as 3D volumetric data or after extracting iso-surfaces (level sets) from the density field. The effect of noise is particularly frustrating when 3D iso-contour (level-set) surfaces are extracted from the volume density field. Typical algorithms for extracting surfaces, such as marching cubes [Lorensen 1987] and dual contour-

ing [Ju 2002] are sensitive to noise. The noise ultimately manifests itself as crude surfaces plagued by artifacts such as disconnected components, bridges and multiple holes in the surface.

Three broad strategies are typically adopted for smoothing noise: 2D smoothing of the sliced views of the density field; surface smoothing of the surfaces extracted as level-sets of the density data; and smoothing of the volume density before surface extraction. 2D smoothing is efficient, but not applicable when 3D models are required from the data. Surface smoothing is popular, but with the disadvantage that certain artifacts such as topological inconsistencies are extremely challenging, if not impossible, to eliminate. Methods potentially useful to contour surface of tomographic data include bilateral filtering based approaches [Jones 2003, Vialaneix 2011], spectral mesh processing [Bruno 2009], and anisotropic surface-based diffusion [Tasdizen 2002, Clarenz 2000]. All these methods aim at preserving the topology of the original surface, and therefore do not entirely tackle our problem.

When the goal is to extract 3D models, filtering the volumetric data before level-set extraction alleviates such inconsistencies. Avoiding to blur sharp features and to decimate fine elements remains a challenge.

5.3 Theoretical Background

In Chapter 4, we introduced and took advantage of two tools: anisotropic diffusion (Section 4.2.1) and local continuous histogram (Section 4.2.2). In this chapter, we make use of the same tools but in three-dimension. Here, we present their extension from two-dimensional to three-dimensional.

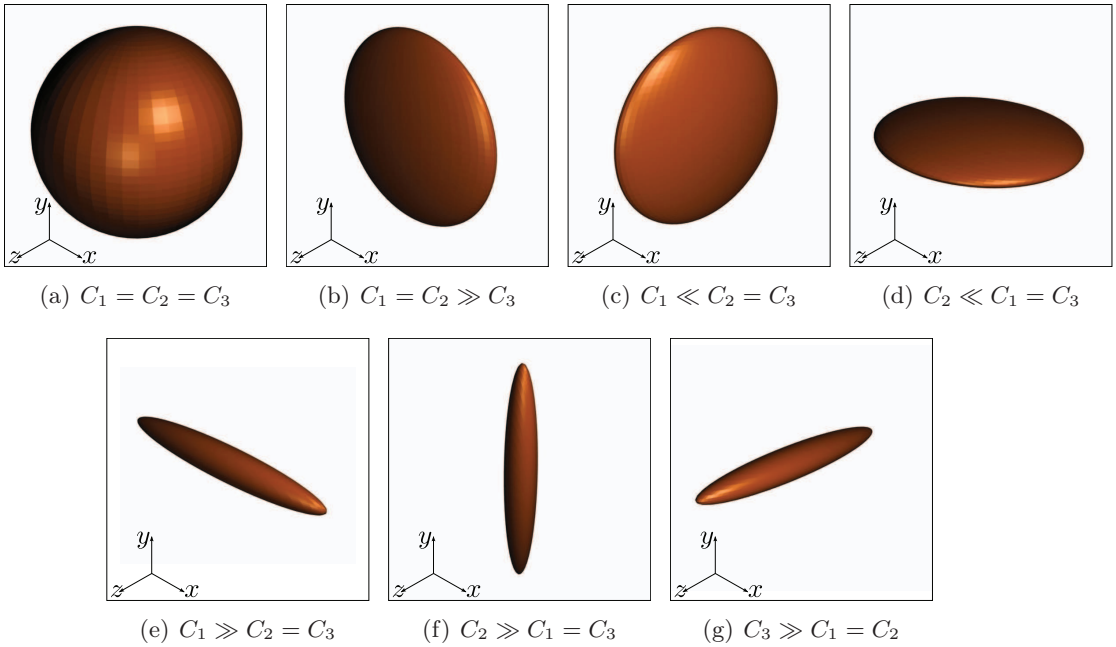
5.3.1 Anisotropic Diffusion

Smoothing using diffusion is a well-studied technique, introduced by Perona and Malik for image smoothing [Perona 1990]. The general form of anisotropic diffusion of a density field f is represented variationally as:

$$\begin{aligned} f_t &= div(M\nabla f) \\ &= M\Delta f + \nabla M \nabla f \end{aligned} \tag{5.1}$$

where M is a symmetric matrix called the diffusion tensor. If M consist of equal constant values C, then Equation 5.1 reduces to an isotropic heat equation:

$$\begin{aligned} f_t &= M\Delta f \\ &= tr\left(\begin{bmatrix} C_1 & 0 & 0 \\ 0 & C_2 & 0 \\ 0 & 0 & C_3 \end{bmatrix} \begin{bmatrix} f_{xx} & f_{xy} & f_{xz} \\ f_{xy} & f_{yy} & f_{yz} \\ f_{xz} & f_{yz} & f_{zz} \end{bmatrix} \right) \\ &= tr(MH(f)) \end{aligned} \tag{5.2}$$



(a) $C_1 = C_2 = C_3$ (b) $C_1 = C_2 \gg C_3$ (c) $C_1 \ll C_2 = C_3$ (d) $C_2 \ll C_1 = C_3$

(e) $C_1 \gg C_2 = C_3$ (f) $C_2 \gg C_1 = C_3$ (g) $C_3 \gg C_1 = C_2$

Figure 5.2: *(a) Isotropic diffusion is performed. Illustrated by the transformation of the sphere to a plane: diffusion is applied more in (b) the x and y-axis than the z-axis, (c) the y and z-axis than the x-axis, and (d) the x and z-axis than the y-axis. The sphere is transformed into a cigar as diffusion is applied more in the (e) x-axis, (f) y-axis, and (g) z-axis.*

---

where $C_1 = C_2 = C_3$ are the scales of the diffusion which form a sphere (Figure 5.2(a)) and $H(f)$ is the Hessian matrix of dataset f. Diffusion is processed equally in all direction of the sphere. Assigning different value to C_1, C_2, and C_3 results in sort of anisotropic diffusion where diffusion is applied proportionally to C_1, C_2, and C_3. If $C_3 \ll C_1 = C_2$, the sphere is transformed into a plane of dimension proportional to C_1 and C_2 (see Figure 5.2(b)). Thus, more diffusion is applied in the x and y-axis orientations than the z-axis. Figures 5.2(c) and 5.2(d) show the cases when $C_1 \ll C_2 = C_3$ and $C_2 \ll C_1 = C_3$ respectively. If $C_1 \gg C_2 = C_3$, the sphere is transformed into a cigar shape oriented in the x-axis signifying that diffusion is applied more in the x-axis than the y and z-axis (see Figure 5.2(e)). Figures 5.2(f) and 5.2(g) show the cases when $C_2 \gg C_1 = C_3$ and $C_3 \gg C_1 = C_2$ respectively.

The above configuration limits our control of diffusion to only three orientation the x-axis, y-axis, and z-axis. We can remove this limitation by expressing Equation 5.2 using the eigen decomposition of M. Since M is symmetric, it can be written as $M = V^T DV$ where D is a diagonal matrix with eigenvalues λ_i and V contains the eigenvectors v_i of M:

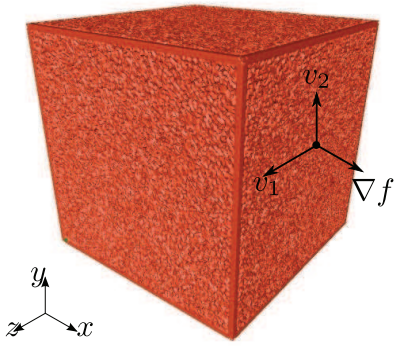


Figure 5.3: *An orthonormal 3D basis formed over a surface. It consists of the gradient vector ∇f and the vectors v_1 and v_2. ∇f is orthogonal to the surface. v_1 and v_2 are parallel to the surface but orthogonal to each other and to ∇I.*

$$\begin{aligned} f_t &= tr(V^T DVH(f)) \\ &= tr(DVH(f)V^T) \\ &= \textstyle\sum_i \lambda_i v_i^T H(f) v_i. \end{aligned} \tag{5.3}$$

In this form, we achieve anisotropic diffusion. It uses the Hessian matrix as a quadratic form to measure the directional second derivatives along vectors v_i, and applies diffusion proportionally to the eigenvalues λ_i. Through this expression, we have the freedom to steer the diffusion process in any orientation by simply choosing the suitable v_i and λ_i. For instance, geometric diffusion corresponds to diffusing only orthogonally to the gradient ∇f, using the diffusion tensor:

$$M_g = V^T \begin{bmatrix} \varepsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} V \text{ with } V = \left[\frac{\nabla f}{\|\nabla f\|}, v_1, v_2 \right] \tag{5.4}$$

where v_1 and v_2 are two vectors that complete ∇f into an orthonormal 3D basis (see Figure 5.3). Choosing a very small ε value reduces the diffusion in the ∇f direction to almost nothing and diffuses only in the directions v_1 and v_2. strictly speaking, geometric diffusion corresponds to $\varepsilon = 0$. In practice, it is better to have $0 < \varepsilon \ll 1$ to keep M_g invertible. An invertible tensor M signifies that applying a geometric diffusion into an image is equivalent to applying a Gaussian filter of covariance M to this volume:

$$f(v, t+1) = \int e^{-u^T M^{-1} u} f(v+u, t) du \tag{5.5}$$

where v is the central voxel (x, y, z) and u is a neighboring voxel of v. We will use this Gaussian property to derive a feature preserving diffusion tensor.

5.3.2 Local Continuous Histogram

Local continuous histograms are extensions of discrete histograms. Each value in the data contributes to neighboring bins—proportionally to a weight function such as a Gaussian—in addition to the histogram bin that it falls into. Local continuous histograms allow robust estimation of histogram properties such as modes and extrema. They work well for edge-aware smoothing operations of images, as pointed out by Kass and Solomon [Kass 2010].

Let F be a 3D dataset, and $f(x)$ be the data value at point x. The local continuous histogram at bin b is defined as a function of point x over the entire dataset by:

$$\begin{aligned} h_b(x) &= \int_F K_b(f(y))g_\sigma(x-y)dy \\ \text{with} \quad & g_\sigma(x) = \frac{1}{\sigma}e^{-\frac{\|x\|^2}{\sigma^2}} \end{aligned} \tag{5.6}$$

the histogram kernel K_b represents how much the value $f(y)$ contributes to the histogram bin at b. It is usually a Gaussian. The sharper this Gaussian, the more local the histogram is. The other Gaussian g_σ is a spatial kernel. Figure 4.4 illustrates a continuous smooth histogram based on a Gaussian distribution.

Using Equation 5.6, we can compute local continuous histograms h_x for all values x at once in a very efficient way, using only two Fourier transforms. First, we rewrite Equation 5.6 as a convolution over x:

$$h_b(x) = (g_\sigma \otimes_x (K_b \circ f))(x)$$

Then, from the convolution theorem, and noting the Fourier transform with $\mathcal{F}$, we have:

$$\mathcal{F}(h_b) = \mathcal{F}(g_\sigma) \times_{\omega_x} \mathcal{F}(K_b \circ f)$$

where the product is performed over the spatial spectral variable ω_x. The spectrum of the Gaussian g_σ is also a Gaussian, $g_{\frac{1}{\sigma}}$, therefore:

$$h_b = \mathcal{F}^{-1}(g_{\frac{1}{\sigma}} \times_{\omega_x} \mathcal{F}(K_b \circ f)) \tag{5.7}$$

5.4 Feature Preserving Smoothing in Volumes

5.4.1 Objective

We apply anisotropic diffusion within the volume. We compute adaptive diffusion tensors at each voxel, such that noise is smoothed up to a specified scale and features larger than this scale are preserved, when contour surfaces are extracted.

In Figure 5.4, we provide an intuition of how this diffusion tensor is constructed.

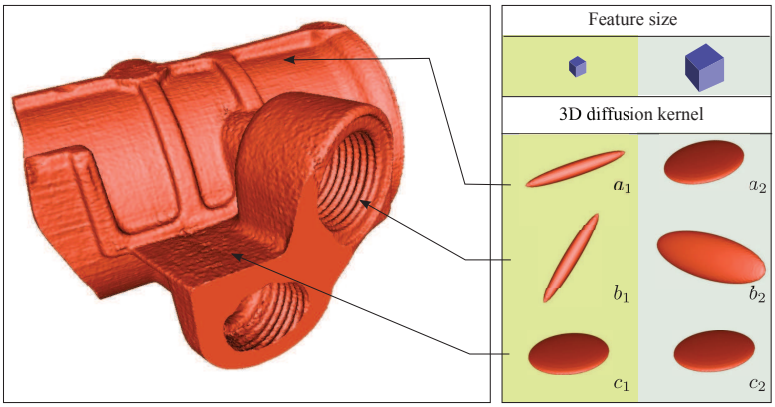


Figure 5.4: *We adapt the diffusion tensors to local variation by respecting the specified scale. For example, in the internal thread, choice of fine scale smoothing requires diffusion along tensor b_1 while coarse smoothing requires tensor b_2 to smooth the thread.*

In the internal thread, when the user specifies a fine scale, we would like to preserve the thread and must diffuse parallel to it (using kernel b_1). At a larger scale, however, we would like to smooth the thread to produce a clean cylinder, which requires a different kernel b_2. At some locations in the volume, the smoothing kernel may be the same across multiple scales such as c_1 and c_2 where the result is flat.

We construct the diffusion tensor at each voxel using the local distribution of gradients, filtered at the user-specified scale. For computing filtered gradient distributions, we build local continuous histograms in the space of directions. To account for the feature size, we compute gradients on a blurred version of the volume, obtained using an isotropic Gaussian of variance equal to half the requested feature size. We then use the filtered gradient histogram to build a diffusion kernel, at each voxel, that preserves the requested feature size.

5.4.2 Scale-space Local Gradient Distributions

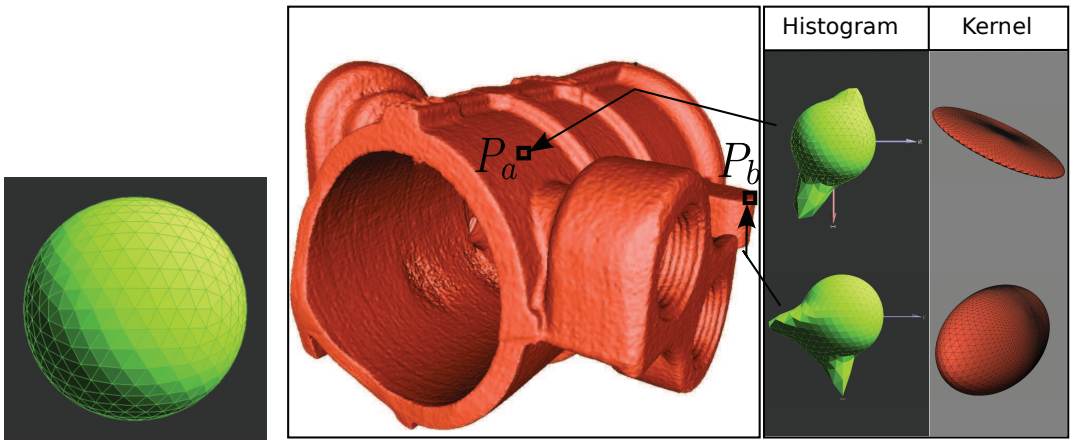
Let s be the specified feature size and f the volumetric data. The continuous histogram of gradients has value $h_\omega(x)$ at point x in direction ω, defined by:

$$h_\omega(x) = \int_{\|y-x\|<2s} g_s(x-y) K_\omega(\nabla f(y)) dy \tag{5.8}$$

where K_w is the normalized Von Mises kernel [Jammalamadaka 2001] defined by

$$K_\omega(\omega') = \frac{k}{2\pi(e^k - e^{-k})} e^{k\omega\cdot\omega'} \tag{5.9}$$

and $\omega \cdot \omega'$ is the dot product between directions ω and ω'. Larger values of k



(a) (b)

Figure 5.5: *(a) A sphere histogram with n directional bins. (b) Continuous local gradient histogram with their respective diffusion tensors at distinct locations. The top figures show a histogram computed over a continuous region resulting with one dominant direction. Its respective diffusion tensor is flat and orthogonal to the dominant direction. The bottom figures show a histogram computed over a region containing an edge resulting with two dominant directions. Its respective diffusion tensor results in a cylinder shape orthogonal to the dominant directions and parallel to the edge.*

correspond to sharper kernels K_ω. Figure 4.7 shows the kernel distribution with various values of k. A continuous gradient histogram of each voxel is discretized and constructed as follow:

- We create n directional bins. They are discretized and scattered uniformly on the unit sphere, using a fixed subdivision of the icosahedron. The greater the number of bins the smoother is the histogram. Figure 5.5(a) illustrates a histogram with n directional bins.
- We compute a weight for each gradient of the volume; to estimate its contribution, relative to each of the directional bin. This is accomplished via the Von Mises kernel in Equation 5.9.
- Finally, we apply a 3D spatial Gaussian g_s weight of variance s over the volume. This excludes all voxels that falls outside the scope of the user-defined feature size s; *i.e.* the radius s, from the central voxel.

Figure 5.5 shows two examples of continuous gradient histograms and their associated computed kernel taken at two locations; P_a on the surface and P_b on an edge, in the model of Figure 5.4. At P_a all gradients are orthogonal to the surface. This produces a single lobe histogram oriented orthogonally to the surface. At P_b,

there are an equal number of gradients directed downwards and leftwards. This produces a histogram with two distinct lobes smoothly directed downwards and to the left relative to the gradients distribution. Note, we disregard gradients outside the user-defined feature size scope the construction of the histogram.

We use two Fourier transform to compute the local continuous gradient histogram. Furthermore, note that we compute the continuous gradient histogram on the gradients of a blurred volume of f rather than the gradients of f itself. Blurring the volume sharpens the gradients by filtering out small scale variation of the data. We apply a 3D Gaussian $g_{\frac{s}{2}}$ of variance $\frac{s}{2}$ on the volume data f using a convolution or equivalently two Fourier transform:

$$\begin{aligned} f^b &= g_{\frac{s}{2}} \otimes f \\ &\text{or} \\ f^b &= \mathcal{F}^{-1}(g_{\frac{2}{s}} \times \mathcal{F}(f)) \end{aligned} \tag{5.10}$$

where f^b is the result blurred volume. Thus, $\nabla f^b(y)$ replaces $\nabla f(y)$ in Equation 5.8.

We use the Gaussian g twice in the process. First to filter out small scale variations of the data (Equation 5.10), then to spread the contribution of each gradient to the histogram of nearby voxels (Equation 5.8). The variance is half the scale of the requested feature size in Equation 5.10 and equal to the feature size in Equation 5.8. We refer the reader to Figures 4.9 and 4.10 for a visual illustration of the Gaussian effect on gradients. Although the effect is shown on 2D images, it has the same effect on volumes.

5.4.3 Computation of Adaptive Diffusion Tensors

Having the local distribution of gradients $\omega \to h_\omega(x)$ at a scale s for all voxel x in the volume, we define the diffusion tensors as:

$$M(x) = \left(\frac{1}{\mathcal{H}(x)} \int_\Omega h_\omega(x) M_g^{-1}(\omega) d\omega \right)^{-1} \tag{5.11}$$
$$\text{with} \quad \mathcal{H}(x) = \int_\Omega h_\omega(x) d\omega$$

where $M_g(\omega)$ is the geometric diffusion tensor defined in the direction ω (see section 5.3.1); *i.e*:

$$M_g(\omega) = V^T \begin{bmatrix} \varepsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} V \text{ with } V = [\omega, v_1, v_2] \tag{5.12}$$

where we use $\varepsilon = \frac{1}{1000}$. We compute a geometric diffusion tensor $M_g(\omega)$ for

each directional bin and then sum them up. However, its contribution to the overall diffusion tensor $M(x)$ is weighted by $h_\omega(x)$. $M(x)$ is then normalized by $\mathcal{H}(x)$—the sum of the local distribution of gradients in each of the histogram directional bins. Note that the matrix inversion involved is always well defined since we sum up positive definite matrices.

Equation 5.11 is justified as diffusing using the geometric diffusion tensor is equivalent to smoothing the data with a Gaussian kernel g defined by its covariance matrix M_g

$$g(x) = e^{-x^t M_g^{-1} x}$$

To account for multiple directional constraint, Gaussian kernels corresponding to different directions are multiplied. Consider two main directions ω_1 and ω_2 orthogonal to one another. Multiplying the Gaussian kernels g_1 and g_2 is an appropriate solution since each Gaussian will cancel out the parts of the support that is not wanted in the other Gaussian. For a proper normalization, we actually need a geometric mean, which in the example above is:

$$g_{12} = (g_1 g_2)^{\frac{1}{2}} = e^{-x^t \frac{M_{g1}^{-1} + M_{g2}^{-1}}{2} x}$$

The geometric mean of Gaussians results in an harmonic mean of their covariance matrices as in Equation 5.11. We provide a visual description of the example above in Figure 4.11. We show the diffusion tensors M_{g1} and M_{g2} of directions ω_1 and ω_2 respectively. They are orthogonal to one another and their geometric mean M_{g12} is a circle. Note that the result is not the intersection of M_{g1} and M_{g2}.

In Section 4.3.3, we provided a visual illustration (Figures 4.12 and 4.13) of the procedure for computing adaptive tensors. Although we presented it in 2D, the concept is the same in 3D.

5.4.4 Diffusion

Following the computation of the diffusion tensor of each voxel, we apply anisotropic diffusion iteratively in the orientation of the respective tensors. Since the diffusion at each voxel is guided by the local distribution of gradients, the smoothing respects and preserves the local features of the volume. We compute the eigenvalues and eigenvectors of each diffusion tensor. Then, we diffuse in the orientation of the eigenvectors proportionally to their respective eigenvalues. Mathematically, we define it as

$$f_t = \sum_{i=0}^{3} \frac{(V_i^T H(f) V_i)\lambda_i^2}{V_i^T V_i} \tag{5.13}$$

where f is a volume of density value with $V_i = \begin{bmatrix} v_{x_i} \\ v_{y_i} \\ v_{z_i} \end{bmatrix}$ and λ_i are the three

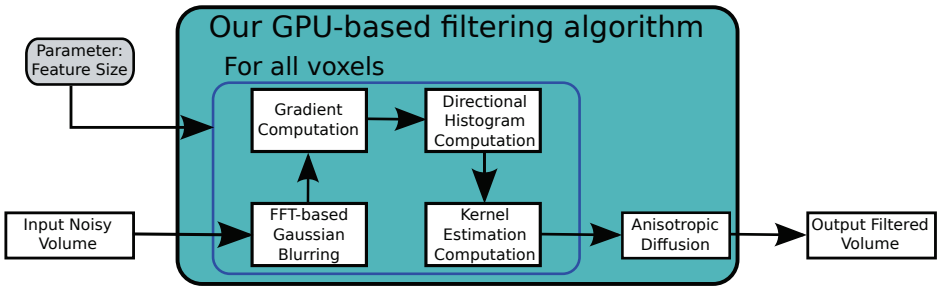


Figure 5.6: *GPU computation pipeline. All computations are done on the GPU. The input are the original volume and the feature size.*

---

eigenvectors and eigenvalues respectively. $H(f) = \begin{bmatrix} f_{xx} & f_{xy} & f_{xz} \\ f_{xy} & f_{yy} & f_{yz} \\ f_{xz} & f_{yz} & f_{zz} \end{bmatrix}$ is the Hessian matrix of volume f. The Hessian matrix is used as a quadratic form to measure the directional second derivatives along the vectors V_i. f_t is iteratively added to f by a time step $\Delta t = 0.025$:

$$f(x, y, z, t + \Delta t) = f(x, y, z, t) + \Delta t(f_t). \tag{5.14}$$

5.5 Implementation

We compute the diffusion tensor almost entirely on GPU using NVIDIA's CUDA model [Sanders 2010]. See Figure 4.15 for the pseudo code, and Figure 5.6 for an overview of the graphics pipeline.

- First, we blur the volume using a convolution with g_s, as in Equation 5.10, using two Fast Fourier Transforms (FFT). We transform f into the Fourier domain (FFT-forward), multiply the spectrum by the spectrum of g_s, and finally transform back into the primal using the second FFT (FFT-inverse). FFT is a standard function provided by NVIDIA's CUDA software development kit.

- For each histogram direction ω_i, and for all voxels x at once, we compute the directional histogram $h_{\omega_i}(x)$ using Equation 5.7. This involves two FFT calls and a product in Fourier space.

- As we compute histogram values, we accumulate the inverse of the geometric diffusion matrix M_g for each voxel in a buffer using Equation 5.11.

- Finally, we invert the matrices for all voxels and use them for smoothing the volume using diffusion.

for all voxels x **do**
  init tensor $M(x)$ to 0
compute $g_s \otimes f$ using 2 FFT + product
for all histogram directions ω_i **do**
  for all voxels x **do**
    compute $K_{\omega_i}(\nabla(g_s \otimes f))$
  convolve with g_s using 2 FFT + product
  for all voxels x **do**
    accumulate $h_{\omega_i}(x)M_g(\omega_i)^{-1}$ into $M(x)^{-1}$
for all voxels x **do**
  invert $M(x)^{-1}$
  diffuse at x according to $M(x)$

Figure 5.7: *Pseudocode for the computation of diffusion tensors in the volume. All operations are done on the GPU, using CUDA.*

Algorithms 5.1 and 5.2 sketch in details the algorithms for computing the diffusion tensors of volumes and performing anisotropic diffusion. In this implementation, directional gradient histograms are never stored explicitly, their values are used directly to compute the integral in Equation 5.11. We discretize the directions into 642 directions scattered uniformly on the unit sphere, using a fixed subdivision of the icosahedron.

5.6 Results and Comparison

We tested our algorithm on several data sets. These datasets all correspond to difficult cases for tomography:

- An inhaler (Figure 5.8 and 5.9). This object is an assembly of plastic and metallic parts, having different permeability to X-ray, resulting in a large range of contrast.

- A mechanical part (Figure 5.10), made from aluminum. The thickness of this object changes with its orientation, resulting again in contrast issues for tomography. This object also contains large smooth areas (the outer cylinder) as well as sharp continuous features (such as the internal threads and the letters at the top).

- An archaeological artifact (Figure 5.11 and 5.12). This object is a wooden box which can not be opened because it is too fragile. The Museum uses tomography to observe the interior of the box in a non-destructive way. The dataset is made of softer materials with a large range of permeability to X-rays: tree leaves, wood and sea shells.

Algorithm 5.1: *Computing 3D Diffusion Tensors for volumes in GPU*

input : f, 3D noisy data. s, feature size.
output: M, vector of diffusion tensors

// Initialize tensors $M(x)$
forall *voxels* x **in** f **do**
  $M(x) \leftarrow 0$

// Initialize continous histogram
$\Omega \leftarrow$ **new** `histogram()`

// Blurring f using Gaussian $g_{\frac{s}{2}}$ and CUDA FFT (using Eq. 5.10).
$F \leftarrow$ CudaFFT (f, FFT-forward)
$F \leftarrow F \times g_{\frac{2}{s}}$
$f^b \leftarrow$ CudaFFT (F, FFT-inverse)

// Computing directional histogram
forall *histogram directions* ω_i **do**
  forall *voxels* x **in** f **do**
    $h_{\omega_i}(x) \leftarrow K_{\omega_i}(\nabla f^b)$

  // convolve wih g_s using 2 CUDA FFT (using Eq. 5.7).
  $H_{\omega_i} \leftarrow$ CudaFFT (h_{ω_i}, FFT-forward)
  $H_{\omega_i} \leftarrow H_{\omega_i} \times g_{\frac{1}{s}}$
  $h_{\omega_i} \leftarrow$ CudaFFT (H_{ω_i}, FFT-inverse)

  // Accumulate the diffusion tensors for each voxel using
  // Eq. 5.11.
  forall *voxels* x **do**
    $V \leftarrow \begin{bmatrix} \omega_i & v_1 & v_2 \end{bmatrix}$
    $\varepsilon \leftarrow \frac{1}{1000}$
    $M_g(\omega_i) \leftarrow V^T \begin{bmatrix} \varepsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} V$
    // Computing geometric mean
    $M(x)^{-1}$ += $h_{\omega_i}(x) M_g(\omega_i)^{-1}$
    $\mathcal{H}(x)$ += $h_{\omega_i}(x)$

// Normalize the tensors $M(x)^{-1}$ and inverse to $M(x)$
forall *voxels* x **do**
  $M(x)^{-1} \leftarrow \frac{M(x)^{-1}}{\mathcal{H}(x)}$
  $M(x) \leftarrow$ `invert`$(M(x)^{-1})$

`return` M

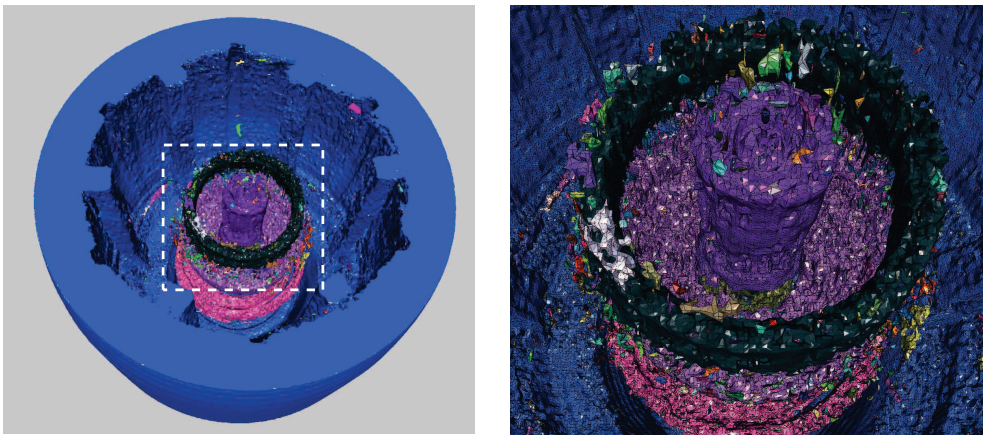
5.6.1 Noise and Connected Components

Surface extraction, using dual contouring on the original datasets results in a large number of separated connected components (several thousands in our experiments), some of them having a high topological genus (many holes), and with visible surface noise. See Figures 5.8, 5.10(a) and 5.11. In these pictures, each color corresponds

Algorithm 5.2: *Computing 3D Diffusion on GPU*

input : f, 3D noisy data. M, vector of diffusion tensors
output: f_s, smooth 3D data at feature size s.

```
1  f_s ← new 3D-data()
2  ∇t ← 0.025
3  for n diffusion steps do
4      forall voxels x do
5          [V_1, V_2, V_3] ← eigenvectors (M(x))
6          [λ_1, λ_2, λ_3] ← eigenvalues (M(x))
7          f_t ← Σ_{i=0}^{3} ((V_i^T H(f(x)) V_i) λ_i^2) / (V_i^T V_i)
8          f_s(x) ← f(x) + ∇t f_t
9      f ← f_s
10 return f_s
```



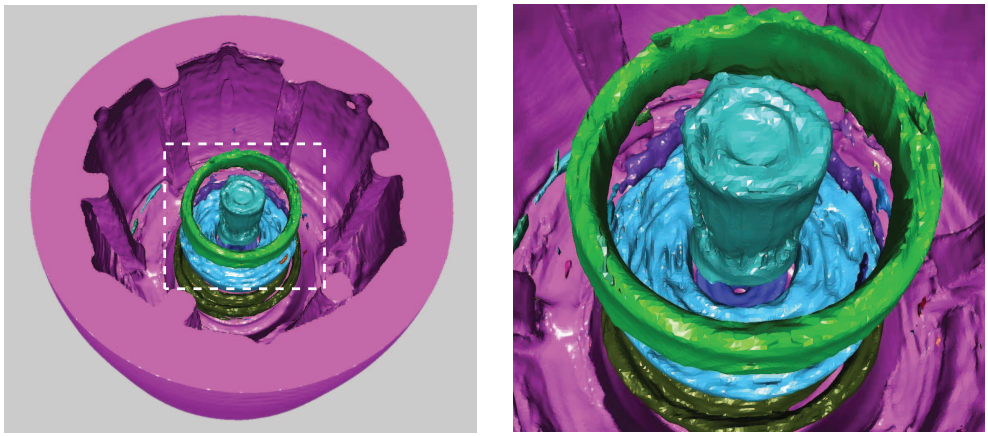
(a) (b)

Figure 5.8: *Inhaler model (256^3 voxels): (a) contour surface extracted from the raw data and (b) close up. Note the large number of disconnected components, holes and surface noise. This example has 4978 different connected components.*

to a different connected component.

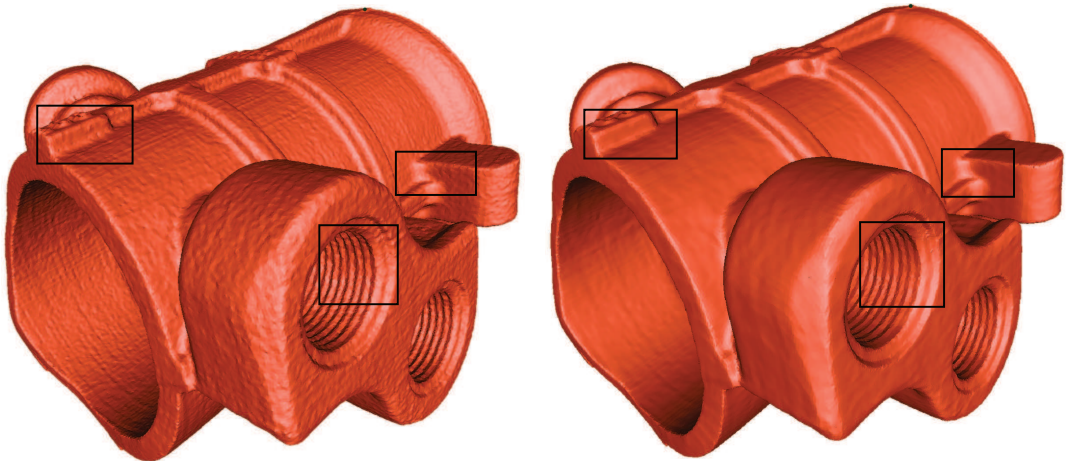
Our algorithm smoothes the volumetric datasets so that surface extraction results in a smaller number of connected components, with less holes and smoother surfaces. See Figures 5.9, 5.10(b) and 5.12 for comparison. The reduction in surface noise is clearly visible. Numerically, the number of separate connected components was reduced from 4978 to 218 for the inhaler dataset, and from 1406 to 468 for the shells in a box dataset.

Figures 5.12 and 5.15 show the volumetric dataset after filtering. The noise reduction is also visible on this volumetric data. Our algorithm makes it easier to



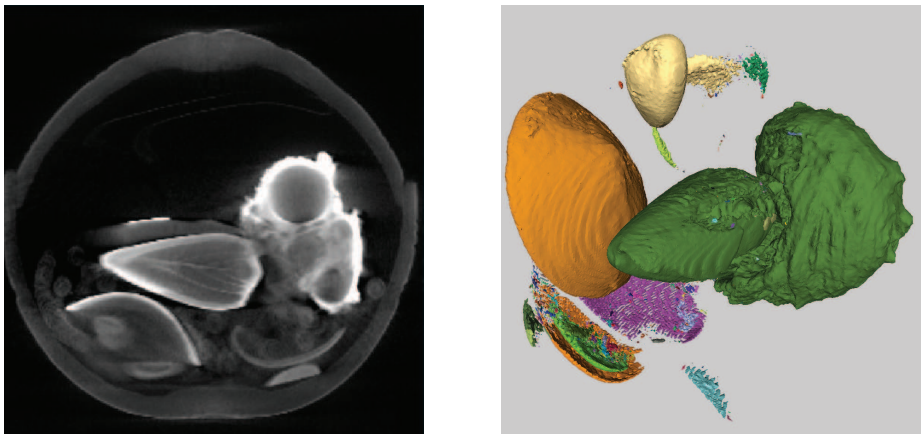
(a) (b)

Figure 5.9: *Inhaler model (*256^3 *voxels): (a) contour surface extracted after volume smoothing with our algorithm and (b) close up. Our algorithm reduced the surface noise as well as the number of connected components and holes. Computation time for this example was 79.8 s, and the number of connected components was reduced to 218.*



(a) (b)

Figure 5.10: *Mechanical part (*300^3 *voxels): (a) surface extracted from original data and (b) surface extracted after smoothing by our algorithm. Note how our algorithm removes the surface noise while preserving the sharp features such as the internal threads or the letters on top of the model. See Figure 5.14 for close-ups on the internal thread region.*



(a) (b)

Figure 5.11: *Shells in a box (256^3 voxels): (a) slice from the original volume data and (b) contour surfaces extracted. Note the large number of connected components (1406 in this example) and the surface noise.*

identify the different components of the objects. It also provides a better separation between different elements. For example, in the shells dataset, the filtered dataset separates clearly between the two shells on the right, unlike the unfiltered dataset.

5.6.2 Preservation of Sharp Features

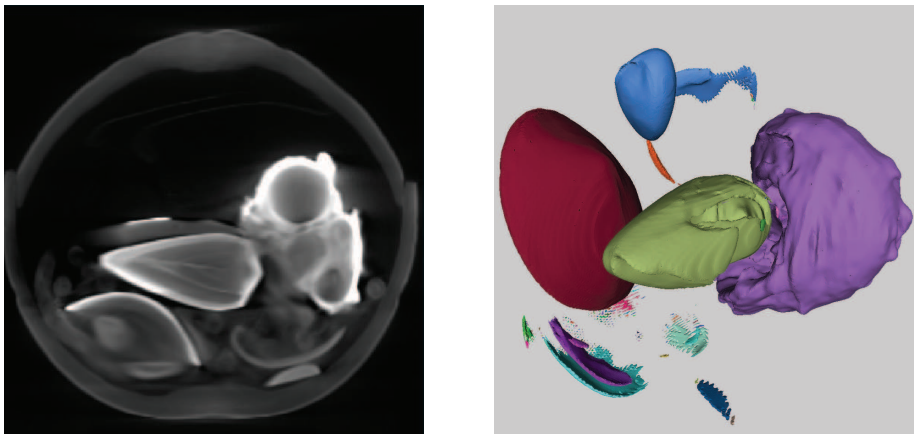
Figure 5.10 compares the surface extracted from the mechanical part dataset after filtering with the surface extracted from the unprocessed dataset.

Figure 5.13 shows close-up details of the extracted surface: the internal thread, the letters engraved at the top of the model, and a combination of sharp and smooth edges at the end of the model. Notice how our method preserves the sharp edges, including the thread and the letters, while smoothing out the surface noise.

5.6.3 Feature Size

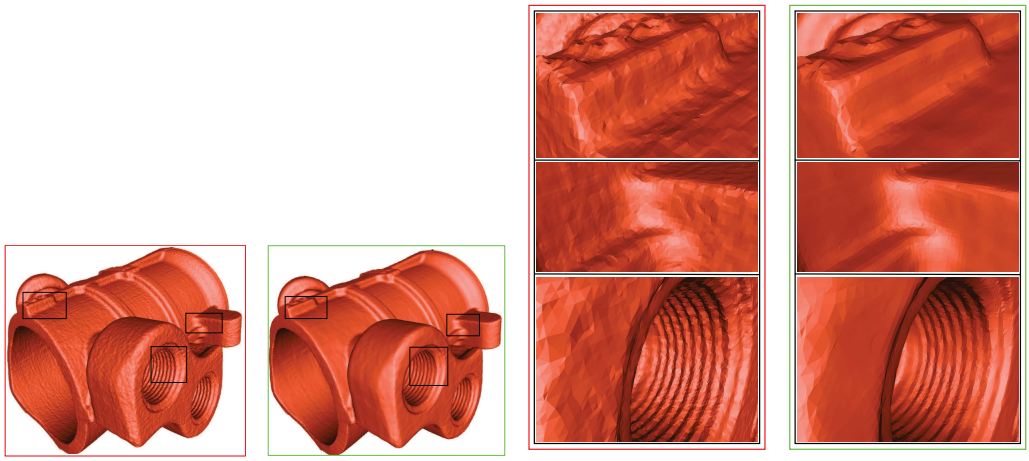
The key parameter of our algorithm is the size of the feature we are interested in. Filtering removes surface details smaller than this feature size, while preserving surface details that are larger.

Figure 5.14 shows the effect of increasing the feature size on the internal thread of the mechanical part dataset. For a small value of the parameter, our algorithm preserves the thread while removing the noise. As we increase the feature size, our algorithm smoothes the threads. Note that the edges of the cylinder are preserved, since they are larger in size.



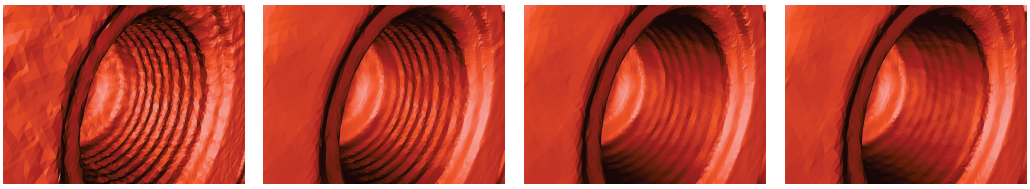
(a) (b)

Figure 5.12: *Shells in a box (256^3 voxels), after filtering with our algorithm: (a) a slice from the volumetric data and (b) contour surfaces extracted. Note how subtle structures, such as the thin internal walls of the central shell, are well preserved in the volume data. Our algorithm removed the surface noise as well as many erroneous surface components, and provided the (correct) separation between the right two shells. The number of connected components was reduced to 468. Compare with Figure 5.11 for the unfiltered data.*



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

Figure 5.13: *Mechanical part (300^3 voxels): (a) contour surface extracted from the raw data, (b) contour surface extracted from the data after filtering with our algorithm ($s = 20$, diffusion steps $= 40$). (c) and (d) close-up on specific details from (a) and (b).*



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

Figure 5.14: *Internal thread from the mechanical part at various feature sizes (Figure 5.10): (a) original data, (b) (c) and (d): filtered data with increasing feature size of 20, 50 and 100 respectively. With a feature size of 20, our algorithm removes the noise but preserves the internal threads. When we increase the feature size to 100, our algorithm removes the threads, but keeps the contours of the cylinder. The filtering takes 79.8 s to compute.*

(a) (b) (c)

Figure 5.15: *Slices from the mechanical part dataset (see (a)): (b) original data and (c) data filtered by our algorithm, $s = 100$. Note how our algorithm removed the noise inside the part volume, as well as the threads. Compare with Figure 5.14 for the extracted surfaces.*

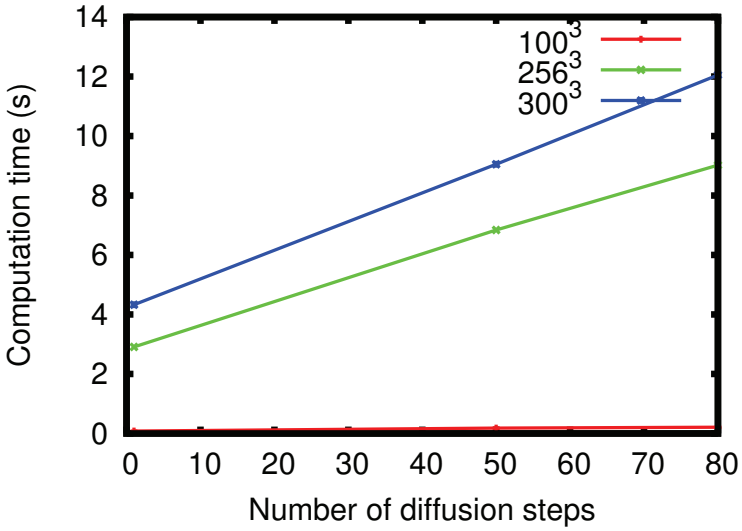


Figure 5.16: *Computation time on the GPU (in seconds) as a function of the number of diffusion steps, for different size of the dataset.*

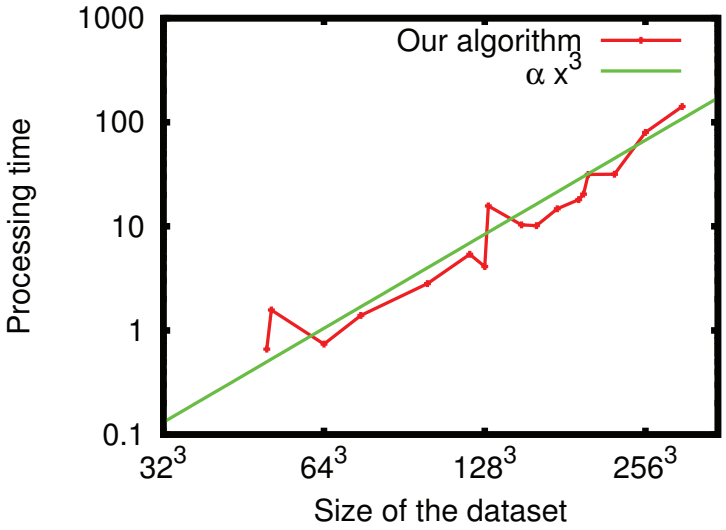


Figure 5.17: *Logscale plot of the processing time (in seconds), as a function of the number of voxels (measured as the edge length of the dataset). The total number of voxels is the cube of this edge length, and the computation time increases linearly with it.*

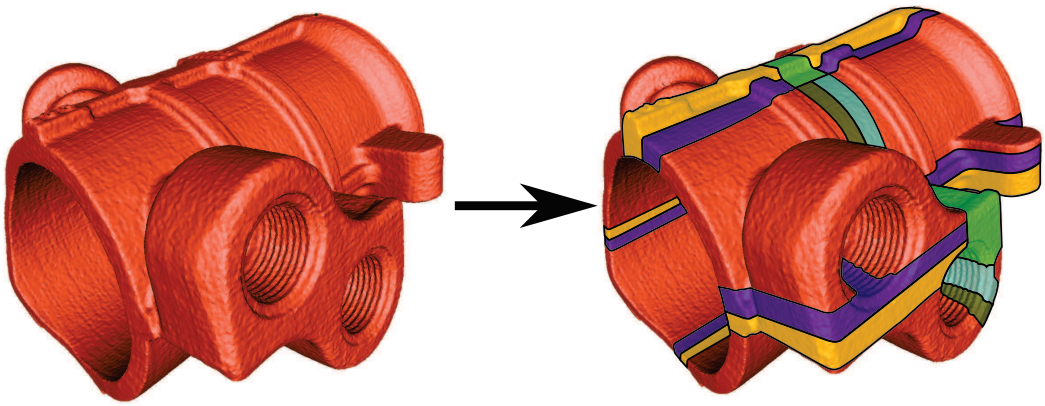


Figure 5.18: *Mechanical part (300^3 voxels) divided into eight pieces. Each piece is processed separately by the GPU, before we combine the results together. The different colored areas highlight the overlap required data from adjacent pieces for a smooth stitching after filtering.*

5.6.4 Computation Time

Computation times for our algorithm depend on two parameters: the number of voxels in the dataset and the number of diffusion steps. It does not depend on the feature size.

The number of diffusion steps has an impact on computation time. Figure 5.16 displays the computation time for 1, 50 and 80 diffusion steps, for different dataset sizes. The time measurements include a constant initializing time, which explains the afine—rather than linear—behavior of the curve. After this first step, computation time increases linearly with the number of steps, as expected.

5.6.5 Scalability

The computation time increases with the total number of voxels n^3. Figure 5.17 shows the computation time as a function of the total number of voxels. We observe sharp variations in computation time, related to data alignment issues. For example, computing on a 128^3 voxels dataset is faster than for 120^3 or 130^3 voxels.

The memory footprint of our algorithm is equal to $64n^3$. On a graphics card with 1 GB of memory, roughly 354 MB are reserved by the card itself for its own use, leaving 669 MB available for our algorithm. This means that the maximum size for a dataset to be processed in a single chunk is roughly 400^3, corresponding to 550 MB of memory.

We segment datasets that are larger than this limit into smaller pieces, which are processed separately by the GPU, before we combine the results together. For a continuous stitching, we keep an overlap between the data pieces. Figure 5.18 shows the mechanical tool divided into eight pieces. In addition to, it highlights

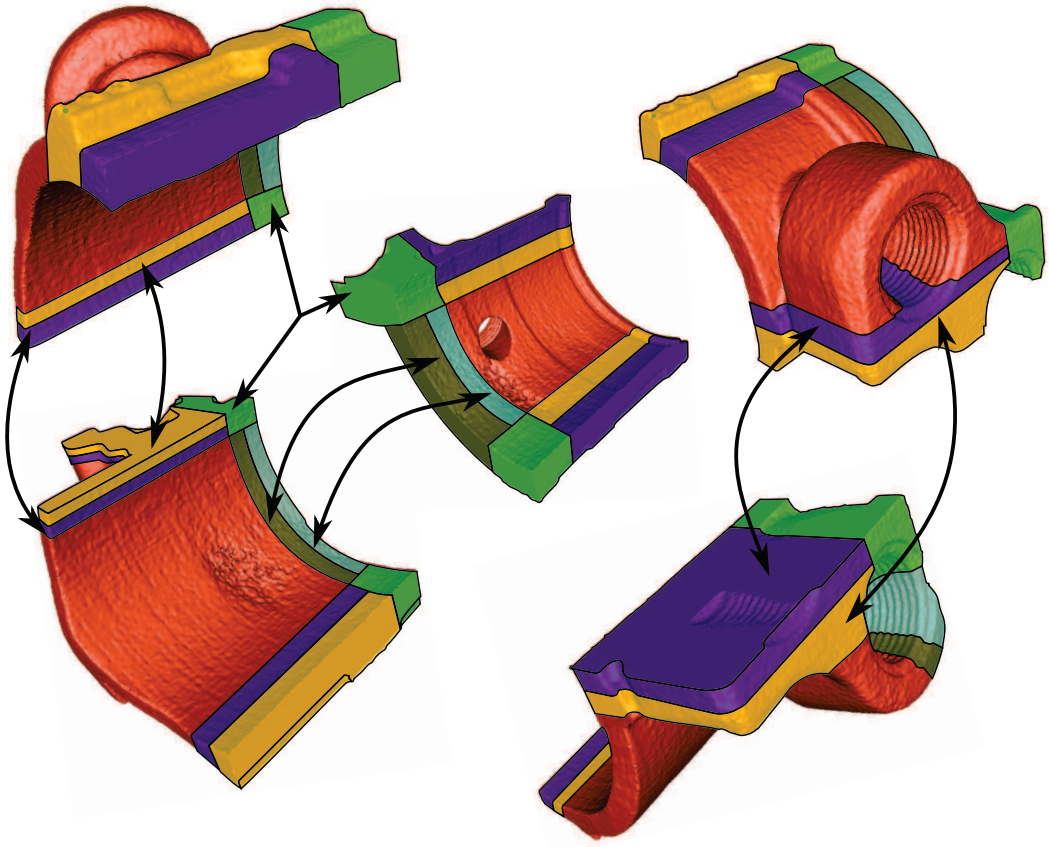
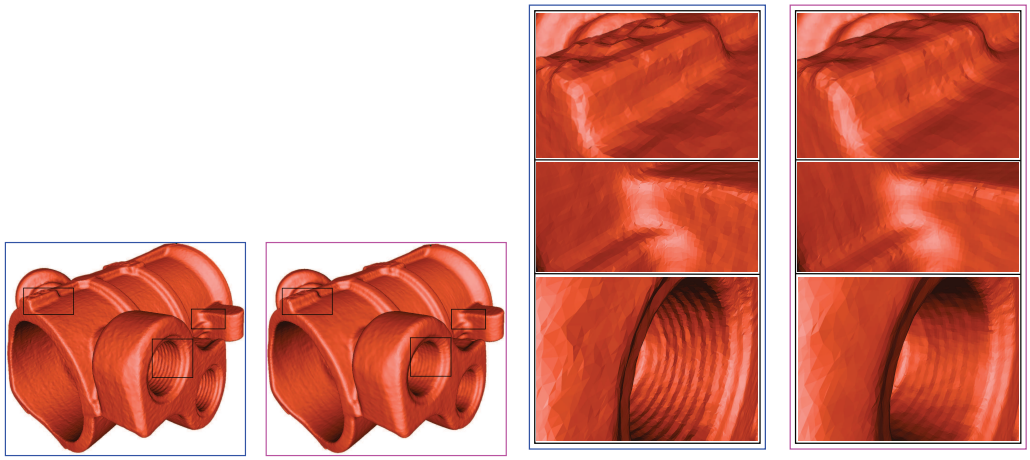


Figure 5.19: *The segmented pieces of Figure 5.18 are spread apart to appreciate the different components of the tool. The different colored areas represent the overlap data from adjacent pieces. The arrows helps visualize the connected areas. Note that each green region is shared by four adjacent pieces.*

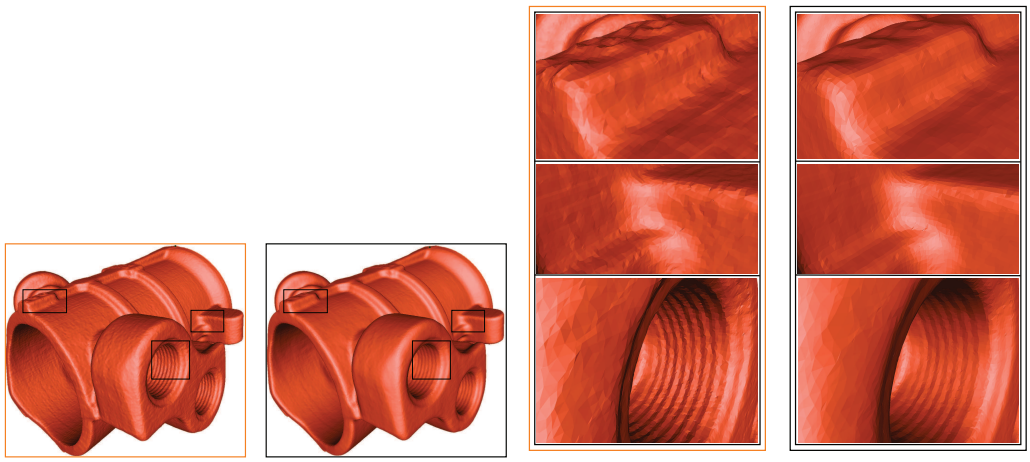
with different color the overlap required data from the adjacent pieces for a smooth stitching after filtering. In Figure 5.19, we spread the pieces apart from one another to appreciate the different components of the tool. The arrows connect the shared overlap data. Note that the each green region is shared by four adjacent pieces of the mechanical tool.

Using this division and stitching method, we can process datasets of arbitrary size. Since dividing the dataset into smaller pieces does not increase the number of voxels (except for the overlap), and since computation times depend linearly on the number of voxels, the division and stitching does not have a significant impact on computation time. The rightmost three points on the curve of Figure 5.17 were computed using this data division. Note that all the datasets pictured in this thesis have been processed in multiple chunks of size 200^3, and were thus processed using data division.



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

Figure 5.20: *Mechanical part (300^3 voxels), using the EED-CED algorithm [Achilleas 2001], with different parameter values: (a)* $\mathrm{K} = 50$, $\alpha = 0.5$, $\mathrm{C} = 0$, *(b)* $\mathrm{K} = 100$, $\alpha = 3$, $\mathrm{C} = 2000$. *(c) and (d) close-up on specific details from (a) and (b).*



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

Figure 5.21: *Mechanical part (300^3 voxels), using bilateral filtering, with different parameter values: (a)* $\sigma_d = 4$, $\sigma_r = 1.5$, *(b)* $\sigma_d = 4$, $\sigma_r = 2$. *(c) and (d) close-up on specific details from (a) and (b).*

Parameter Values	Volume Size		
	100^3	256^3	300^3
σ_d= 4, σ_r=1.5	0.72 s	8.21 s	13.9 s
σ_d= 4, σ_r=2	0.72 s	8.21 s	13.9 s
σ_d= 8, σ_r=2	4.09 s	52.41 s	89.47 s
σ_d= 10, σ_r=2	7.46 s	75.4 s	167.8 s

Table 5.1: *Computation time for bilateral filtering (in seconds), on the GPU, as a function of volume size and parameter values.*

5.6.6 Computing on the GPU

Similar to our 2D method (see Chapter 4), we make use the massively parallel processing power of the GPU to compute the adaptive diffusion tensors and the anisotropic diffusion. Its single-instruction-multiple-data (SIMD) architecture [Flynn 1972] allows us to compute the diffusion tensors and anisotropic diffusion in parallel for entire blocks of pixels and voxels since both operations are *embarrassingly parallel*. Using the GPU, we gain in efficiency by roughly a factor of 1000 compared to the use of a CPU. For instance, filtering a volume of size 300^3 (27 millions) voxels on the CPU would take $\gg$ 24 hours compared to 2 mins on the GPU.

5.6.7 Comparison with Existing Algorithms

For comparison, we implemented the hybrid algorithm, EED-CED (Edge Enhancing Diffusion, Coherence Enhancing Diffusion) [Achilleas 2001] and 3D bilateral filtering [Tomasi 1998, Paris 2007]. Both algorithms were implemented on the GPU using NVidia CUDA.

Figures 5.20 and 5.21 show the results of these algorithms on the mechanical part dataset. Compare with Figure 5.13 for our own algorithm on the same dataset. Both algorithms exhibit the same behavior: for some values of the parameters, they keep the edges and features, but they also kept some of the noise. If we push the parameters until the point where the surface noise disappears, we start losing features and details. For each algorithm, we display the best results we could find.

The EED-CED is basically a diffusion algorithm, and we implemented it on the GPU (only the computation of the kernel changes with our algorithm). As a consequence, it has the same computation time as our algorithm.

Bilateral filtering was also implemented on the GPU, resulting in computation times similar to those of our algorithm. Please refer to Table 5.1 for computation time (in seconds), for several values of the parameters and volume size. It is faster than our algorithm when σ_d is reasonably small. However, as σ_d increases so does its computation time which can become slower than ours.

Chapter 6

Conclusion

Contents

6.1 Summary of Contributions

We have addressed the problem of noise in two-dimensional images and three-dimensional volumes that is unavoidably added during the acquisition of images and volumes. We have presented two new fast multi-scale smoothing methods that adapt to the local features of the subjects of the images and volumes. Moreover, we made use of *anisotropic diffusion* and *continuous histogram* to build our adaptive diffusion tensors.

First, we presented our multi-scale smoothing method in two-dimensional space to tackle the noise in two-dimensional images. To take into account the sharp features, we compute the local continuous histograms of gradients of the image. Then, we compute the adaptive diffusion tensors using these histograms. Finally, we apply anisotropic diffusion in the image using the computed tensors. Since the diffusion at each pixel is guided by the local distribution of gradients, the smoothing respects and preserves the local features.

We then extended the concept of our two-dimensional smoothing method to three-dimensional space to remove noise in volumes improving the quality of their extracted iso-surfaces. To achieve that, we also extended the anisotropic diffusion and local continuous histogram from their two-dimensional space to three-dimensional space to use for computing 3D adaptive diffusion tensors and remove the noise without removing the sharp features.

Our algorithms filter images and volumes extremely fast since their *embarrassingly parallel* nature fits them nicely on parallel machines especially GPU. In addition to, GPU's SIMD architecture allows us to compute the diffusion tensors and anisotropic diffusion in parallel for entire blocks of pixels and voxels in one time step. This makes the run-time of our methods significantly faster compared to a sequential implementation. Furthermore, our technique can process datasets of arbitrary size by applying our division and stitching method.

Despite their simplicity, our algorithms unify previous methods, such as CED and EED, that crafted diffusion tensors, based on a set of special cases, striving

to detect potential edges and corners in the datasets. Our diffusion tensors automatically adapt to the features' shape and diffuse along them to preserve them. Furthermore, we only have one parameter to tune—the feature size—compared to the numerous conditions to follow and parameters to calibrate in other methods.

Finally, we have, along this thesis, demonstrated through concrete examples that our algorithms smooth noise effectively while preserving the sharp features. Our two-dimensional method was tested on several images such as scanned text, photographs, and medical images. In addition to preserving edges and corners, we showed that it repairs cracks and helps better distinguish between different components of the image especially in medical images. Our three-dimensional method was tested on three-dimensional reconstructed mechanical parts and cultural heritage items. Through them, we showed that crude surfaces were eliminated and topological complexes were reduced making the reconstruction objects identifiable.

6.2 Perspectives

In this thesis, we tackled the problem of smoothing noise in two-dimensional images and three-dimensional volumes while keeping the important information of the datasets. As we have demonstrated, our smoothing methods have potential practical applications. The 2D method could be used to smooth scanned noisy texts, medical, and noisy images. And the 3D method is used on (but not limited to) 3D volumes of biological subjects, mechanical parts, and cultural heritage. We hope we have paved the way for additional future research. We see three main directions for possible improvement and extension:

- Currently, anisotropic diffusion is applied iteratively until the user is satisfied with the result. This is problematic because the result might not be optimal since it is based on the user visual interpretation. In addition to, it might be a source of frustration for the user specially when smoothing three-dimensional volumes of objects. In that case, iso-surfaces of the volumes need to be extracted each time several steps of anisotropic diffusion is applied to check if the effect is satisfying. If it's not, then the user might decide to apply more diffusion steps or start over to apply less. Thus, it would be of great utility to have a system that determines the optimal number of diffusion steps that needs to be applied to have optimal smoothing datasets.

- The functionality of our tree-dimensional smoothing method can be extended to smooth noisy meshes acquired by the means of laser scans and their likes. This can be done by first converting the mesh into a three-dimensional volume of distance field [Jones 2006]. Then, apply our three-dimensional multi-scale smoothing method on the distance field volume. Finally, extract the iso-surface of the smooth mesh from the smoothed volume.

- We want to extend our program with an interactive user-friendly interface where users can select specific areas on images, volumes, and meshes to

smooth. Such a software would be useful for passionate users and experts in the domains cited above.

Chapter 7

Conclusion

Contents

7.1 Résumé des contributions

Nous nous sommes intéressés au problème du bruit dans les images 2D et les volumes 3D qui est inévitablement ajouté durant leur acquisition. Nous avons présenté deux nouvelles méthodes de filtrage multi-échelle qui s'adaptent aux caractéristiques locales des images et des volumes. De plus, nous avons utilisé la *diffusion anisotrope* et l'*histogramme continu* pour construire nos tenseurs de diffusion adaptive.

Dans un premier temps, nous avons présenté notre méthode de lissage multi-échelle dans l'espace 2D pour traiter le bruit dans des images. Afin de respecter les lignes caractéristiques, nous calculons l'histogramme continu local des gradients de l'image. Ensuite, en utilisant ces histogrammes nous calculons les tenseurs de diffusion adaptive. Finalement, nous appliquons la diffusion anisotrope sur l'image en utilisant les tenseurs calculés. Le lissage respecte et préserve les caractéristiques locales puisque la diffusion à chaque pixel est guidée par la distribution locale des gradients.

Ensuite, nous avons prolongé le concept de notre méthode de lissage 2D à l'espace 3D pour enlever le bruit dans les volumes afin d'améliorer la qualité des iso-surfaces extraites de ceux-ci. Pour atteindre cela, nous avons étendu les concepts de diffusion anisotrope et d'histogramme continu à l'espace 3D. A nouveau, la méthode proposée calcule des tenseurs de diffusion adaptifs 3D et enlève le bruit sans enlever les lignes caractéristiques.

Nos algorithmes sont efficaces en temps de calcul grâce a leur implémentation parallèle (GPU) comparativement à une implémentation séquentielle. L'architecture du GPU nous permet de calculer et d'appliquer en parallèle les tenseurs de diffusion pour des blocs entiers de pixels et de voxels. De plus, nos méthodes peuvent traiter des données de taille aléatoire en segmentant les données en plusieurs petits morceaux.

Le contexte théorique développé dans cette thèse unifie celui de méthodes antérieures comme les méthodes CED et EED qui prennent en compte des conditions

spéciales lors du calcul des tenseurs, afin de détecter les arrêtes et les coins des données. Nos tenseurs de diffusion s'adaptent aux formes des caractéristiques et diffusent le long de ces formes pour les préserver. Nos méthodes reposent sur un seul paramètre permettant de règler la taille du descripteur, là où les méthodes précedentes nécessitent l'ajustement de nombreux paramètres.

Finalement, nous avons démontré par des exemples concrets que nos algorithmes lissent le bruit tout en préservant les lignes caracteristiques. Nous avons testé notre méthode 2D sur une grande variété d'images : textes scanné, photographies, et images médicales. En plus de préserver les arrêtes et les coins, nous avons montré que notre méthode comble les trous et aide à mieux distinguer entre les différentes composantes de l'image, en particulier dans les images médicales. Nous avons également testé notre algorithme sur des pièces mécaniques 3D et des objets liés au patrimoine culturel. Nous avons montré que notre algorithme élimine les surfaces rudes et réduit les erreurs de topologie dans les volumes qui rendent les objets difficilement identifiables.

7.2 Perspectives

Dans cette thèse, nous avons abordé le problème du lissage de bruit dans les images et volumes tout en préservant les informations importantes des données. Ces méthodes de lissage ont des applications pratiques. La méthode 2D peut être utilisée pour lisser des scans de textes bruités et des images bruitées en général. La méthode 3D peut être utilisée sur entre autres des volumes d'objets biologiques, des pièces mécaniques et des objets liés au patrimoine culturel. Nous voyons trois directions principales pour de possibles améliorations à ces methodes:

- Actuellement, la diffusion anisotrope est appliquée itérativement jusqu'à que l'utilisateur soit satisfait du résultat. Le résultat n'est peut-être pas idéal car basé sur l'interprétation visuelle. Il peut s'agir aussi d'une source de frustration pour l'utilisateur pendant le lissage des volumes 3D des objets. Dans ce cas, les iso-surfaces des volumes doivent être extraites après chaque étape de diffusion anisotrope pour vérifier le niveau de satisfaction. Si l'utilisateur n'est pas satisfait, il devra peut-être décider de diffuser plus ou recommencer du début et diffuser moins. Par conséquent, il serait plus intéressant d'avoir un système adaptatif qui détermine *a priori* le nombre d'étapes idéales de diffusion nécessaires afin d'obtenir des résultats satisfaisants.

- Notre méthode de lissage 3D peut être utilisée pour lisser des maillages bruitées acquis par des moyens de *scan* comme le laser. Cela peut être obtenu en convertissant dans un premier temps le maillage en un champs de distances [Jones 2006]. Dans un deuxième temps, il faut appliquer notre méthode 3D sur celui-ci. Enfin, l'iso-surface du maillage est extraite du volume lissé.

- Nous souhaitons enfin ajouter à notre logiciel une interface interactive et intuitive qui donnera aux utilisateurs des options supplémentaires permettant de

choisir des régions spécifiques sur des images, volumes, et maillages à lisser. Un tel logiciel serait utile pour des utilisateurs amateurs et experts.

Bibliography

[Achilleas 2001] S. F. Achilleas and H. Reiner. *Noise Reduction in Electron Tomographic Reconstructions Using Nonlinear Anisotropic Diffusion.* Journal of Structural Biology, vol. 135, no. 3, pages 239 – 250, 2001. (Cited on pages 44, 45, 46, 47, 48, 105 and 106.)

[Bajla 1993] I. Bajla, M. Marusiak and M. Srámek. *Anisotropic Filtering of MRI Data Based upon Image Gradient Histogram.* In Proceedings of the 5th International Conference on Computer Analysis of Images and Patterns, CAIP '93, pages 90–97, London, UK, 1993. Springer-Verlag. (Cited on page 38.)

[Black 1998] M. J. Black, H Sapiro, D. H. Marimont and D. Heeger. *Robust Anisotropic Diffusion.* IEEE Transaction Image Processing, vol. 7, pages 421–432, 1998. (Cited on pages 36 and 38.)

[Boult 1993] T. E. Boult and R. A. Melter. *G-neighbors.* SPIE, 1993. (Cited on page 26.)

[Bracewell 1999] R. N. Bracewell. The fourier transform and its applications. McGraw-Hill Science/Engineering/Math; 3 edition, 1999. (Cited on page 21.)

[Brigham 1974] E. O. Brigham. The Fast Fourier Transform. Prentice-Hall, Englewood Cliffs, New Jersey, 1974. (Cited on pages 22 and 29.)

[Brigham 1988] E. O. Brigham. The Fast Fourier Transform and its Applications. Prentice-Hall, Englewood Cliffs, New Jersey, 1988. (Cited on pages 22 and 29.)

[Bruno 2009] L Bruno and H. Zhang. *Spectral Mesh Processing.* In ACM SIGGRAPH ASIA 2009 Courses, pages 17:1–17:47, 2009. (Cited on page 86.)

[Canny 1986] J. Canny. *A Computational Approach to Edge Detection.* IEEE Transaction Pattern Analysis Machine Intelligence, vol. 8, pages 679–698, June 1986. (Cited on page 36.)

[Clarenz 2000] U. Clarenz, U. Diewald and M. Rumpf. *Anisotropic Geometric Diffusion in Surface Processing.* In Visualization '00, 2000. (Cited on page 86.)

[Comaniciu 1998] D. Comaniciu and P. Meer. *Distribution Free Decomposition of Multivariate Data.* Pattern Analysis and Applications, vol. 2, pages 22–30, 1998. (Cited on page 43.)

[Comaniciu 2000] D. I. Comaniciu. *Nonparametric Robust Methods for Computer Vision.* PhD thesis, New Brunswick, NJ, USA, 2000. AAI9958400. (Cited on page 43.)

[Comaniciu 2002] D. Comaniciu and P. Meer. *Mean Shift: A Robust Approach Toward Feature Space Analysis*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, pages 603–619, 2002. (Cited on page 43.)

[Davis 1978] L.S. Davis and A. Rosenfeld. *Noise Cleaning by Iterated Local Averaging*. IEEE Transactions on Systems, Man, and Cybernetics, vol. 8, pages 705–710, 1978. (Cited on page 25.)

[Fernandez 2003] G. Fernandez, H. Bischof and R. Beichel. *Nonlinear Filters on 3D CT Imaging - Bilateral Filter and Mean Shift Filter*. In Proceedings of the 8th CVWW, pages 21–26, Valtice, Czech Republic, 2003. (Cited on page 43.)

[Flynn 1972] M. Flynn. *Some Computer Organizations and Their Effectiveness*. IEEE Transaction on Computers, vol. C-21, pages 948+, 1972. (Cited on pages 81 and 106.)

[Funkhouser 1995] T. Funkhouser. *Basic Signal Processing*. 1995. (Cited on page 17.)

[Gallagher 1981] N. Gallagher and G. Wise. *A Theoretical Analysis of the Properties of Median Filters*. IEEE Transactions on Acoustics Speech and Signal Processing, vol. ASSP-29, no. 6, pages 1136–1141, 1981. (Cited on page 25.)

[Gonzalez 2001] R. C. Gonzalez and R. E. Woods. Digital image processing. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd édition, 2001. (Cited on page 24.)

[Graham 1962] R. Graham. *Snow Removal–A Noise-Stripping Process for Picture Signals*. Information Theory, IRE Transactions, vol. 8, pages 129–144, 1962. (Cited on page 25.)

[Gropp 1998] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir and M. Snir. MPI - the complete reference: Volume 2, the MPI-2 extensions. MIT Press, Cambridge, MA, USA, 1998. (Cited on page 43.)

[Herman 2009] G. T. Herman. Fundamentals of computerized tomography: Image reconstruction from projections. Springer, 2nd édition, 2009. (Cited on page 85.)

[Huang 1979] T. Huang, G. Yang and G. Tang. *A Fast Two-Dimensional Median Filtering Algorithm*. IEEE Transactions on Acoustics Speech and Signal Processing, vol. 27, no. 1, pages 13–18, 1979. (Cited on page 25.)

[Hy 2006] K. Hy. *Gradient Histogram-Based Anisotropic Diffusion*. Personal Communication, 2006. (Cited on page 36.)

[Jammalamadaka 2001] S. R. Jammalamadaka and A. Sengupta. Topics in circular statistics. World Scientific Publishing Company, 2001. (Cited on pages 57 and 90.)

[Jiang 2003] W. Jiang, M. L. Baker, Q. Wu, C. Bajaj and W. Chiu. *Applications of a Bilateral Denoising Filter in B electron Microscopy.* Journal of Structural Biology, vol. 144, no. 1-2, pages 114–122, 2003. (Cited on page 43.)

[Jones 2003] T. R. Jones, F. Durand and M. Desbrun. *Non-Iterative, Feature-Preserving Mesh Smoothing.* ACM Trans. Graph., vol. 22, no. 3, pages 943–949, July 2003. (Cited on page 86.)

[Jones 2006] Mark W. Jones, J. Andreas Baerentzen and M. Sramek. *3D Distance Fields: A Survey of Techniques and Applications.* IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, vol. 12, pages 581–599, July 2006. (Cited on pages 108 and 112.)

[Ju 2002] T. Ju, F. Losasso, S. Schaefer and J. Warren. *Dual Contouring of Hermite Data.* ACM Trans. Graph., vol. 21, no. 3, pages 339–346, July 2002. (Cited on page 86.)

[Kass 2010] M. Kass and J. Solomon. *Smoothed Local Histogram Filters.* ACM Trans. Graph., vol. 29, no. 4, pages 100:1–100:10, July 2010. (Cited on pages 55 and 89.)

[Lee 1980] J. S. Lee. *Digital Image Enhancement and Noise Filtering by Use of Local Statistics.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 2, no. 2, pages 165–168, 1980. (Cited on page 25.)

[Lee 1981] J.-S. Lee. *Refined Filtering of Image Noise Using Local Statistics.* Computer Graphics and Image Processing, vol. 15, no. 4, pages 380–389, 1981. (Cited on page 25.)

[Lorensen 1987] W. E. Lorensen and H. E. Cline. *Marching cubes: A High Resolution 3D Surface Construction Algorithm.* Computer Graphics, vol. 21, no. 4, pages 163–169, July 1987. (Cited on page 85.)

[Mayer 2007] M. Mayer, A. Borsdorf, H. Köstler, J. Hornegger and U. Rüde. *Nonlinear Diffusion Noise Reduction in CT Using Correlation Analysis.* In 3rd Russian-Bavarian Conference on Biomedical Engineering, volume 1, pages 155–159, Erlangen, 2007. (Cited on page 38.)

[Meijering 2002] E. Meijering, W.J. Niessen, J. Weickert and M. Viergever. *Diffusion-Enhanced Visualization and Quantification of Vascular Anomalies in Three-Dimensional Rotational Angiography: Results of an In-Vitro Evaluation.* Medical Image Analysis, vol. 6, no. 3, pages 215–233, 2002. (Cited on pages 44, 45 and 47.)

[Mendrik 2009] A. Mendrik, E.-J. Vonken, A. Rutten, M. Viergever and B. Ginneken. *Noise Reduction in Computed Tomography Scans Using 3-D Anisotropic Hybrid Diffusion With Continuous Switch.* IEEE Transactions

Medical Imaging, vol. 28, no. 10, pages 1585–1594, 2009. (Cited on pages 46 and 47.)

[Narendra 1981] P. M. Narendra. *A Separable Median Filter for Image Noise Smoothing.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 3, no. 1, pages 20–29, 1981. (Cited on page 25.)

[Okada 1985] M. Okada. *Noise Evaluation and Filter Design in CT Images.* IEEE Transactions on Biomedical Engineering, vol. 32, no. 9, pages 713 –719, September 1985. (Cited on pages 25 and 26.)

[Paris 2007] S. Paris, P. Kornprobst, J. Tumblin and F. Durand. *A Gentle Introduction to Bilateral Filtering and its Applications.* In ACM SIGGRAPH 2007 courses, New York, NY, USA, 2007. ACM. (Cited on pages 29 and 106.)

[Paris 2008] S. Paris, P. Kornprobst, J. Tumblin and F. Durand. *Bilateral Filtering: Theory and Applications.* Foundations and TrendsÂ® in Computer Graphics and Vision, vol. 4, no. 1, pages 1–75, 2008. (Cited on page 29.)

[Paris 2009] S. Paris, P. Kornprobst and J. Tumblin. Bilateral filtering. Now Publishers Inc., Hanover, MA, USA, 2009. (Cited on page 29.)

[Perona 1990] P. Perona and J. Malik. *Scale-Space and Edge Detection Using Anisotropic Diffusion.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, no. 7, pages 629–639, July 1990. (Cited on pages 35, 36, 37, 49, 51, 52, 77, 78 and 86.)

[Romeny 1994] H. Romeny. Geometry-driven diffusion in computer vision. 1994. (Cited on page 38.)

[Sanders 2010] J. Sanders and E. Kandrot. Cuda by example: An introduction to general-purpose GPU programming. Addison-Wesley Professional, 1st édition, 2010. (Cited on pages 43, 63 and 94.)

[Schaap 2008] M. Schaap, A. Schilham, K.J. Zuiderveld, M. Prokop, E.-J. Vonken and W.J. Niessen. *Fast Noise Reduction in Computed Tomography for Improved 3D Visualization.* IEEE Transactions on Medical Imaging, vol. 27, no. 8, pages 1120 –1129, August 2008. (Cited on page 48.)

[Snir 1998] M. Snir, S. Otto, S. Huss-Lederman, D. Walker and J. Dongarra. MPI-the complete reference, volume 1: The MPI core. MIT Press, Cambridge, MA, USA, 2nd. (revised) édition, 1998. (Cited on page 43.)

[Tasdizen 2002] T. Tasdizen, R. Whitaker, P. Burchard and S. Osher. *Geometric Surface Smoothing Via Anisotropic Diffusion of Normals.* In Visualization '02, pages 125–132, 2002. (Cited on page 86.)

[Tomasi 1998] C. Tomasi and R. Manduchi. *Bilateral Filtering for Gray and Color Images*. In International Conference on Computer Vision, pages 839 –846, January 1998. (Cited on pages 22, 29, 43, 78 and 106.)

[Tukey 1977] John W. Tukey. Exploratory data analysis. Addison-Wesley, 1977. (Cited on page 25.)

[Vialaneix 2011] G. Vialaneix and T. Boubekeur. *SBL Mesh Filter: A Fast Separable Approximation of Bilateral Mesh Filtering*. In Vision, Modeling and Visualization (VMV) 2011, 2011. (Cited on page 86.)

[Wang 1981] D.C.C. Wang, A.H Vagnucci and C.C Li. A gradient inverse weighted smoothing scheme and the evaluation of its performance., volume 15. Computer Vision, Graphics, and Image Processing, 1981. (Cited on page 25.)

[Weickert 1994] J. Weickert. *Scale-Space Properties of Nonlinear Diffusion Filtering with a Diffusion Tensor*. Rapport technique, Laboratory of Technomathematics, University of Kaiserslautern, P.O, 1994. (Cited on pages 39, 40 and 82.)

[Weickert 1997] J. Weickert and E. Heidelberglaan. *A Review of Nonlinear Diffusion Filtering*, 1997. (Cited on pages 39, 40 and 82.)

[Weickert 1998] J. Weickert. Anisotropic diffusion in image processing. Teubner-Verlag, 1998. (Cited on pages 36 and 38.)

[Weickert 1999a] J. Weickert. *Coherence-Enhancing Diffusion Filtering*. Int. J. Comput. Vision, vol. 31, pages 111–127, April 1999. (Cited on pages 40 and 82.)

[Weickert 1999b] J. Weickert. *Coherence-Enhancing Diffusion of Colour Images*. Image Vision Comput., vol. 17, no. 3-4, pages 201–212, 1999. (Cited on pages 39, 40 and 82.)

[Witkin 1983] A. P. Witkin. *Scale-Space Filtering*. In Proceedings of the Eighth international joint conference on Artificial intelligence - Volume 2, IJCAI'83, pages 1019–1022, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc. (Cited on page 35.)

[Yoo 2001] Y. Yoo. *Tutorial on Fourier Theory*, 2001. (Cited on page 21.)