Université de Montréal

Architecture et filtres pour la détection des chenaux dans la glace de l'océan Arctique

par
Daniel Léonard

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Août, 2008

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

Architecture et filtres pour la détection des chenaux dans la glace de l'océan Arctique

présenté par:

Daniel Léonard

a été évalué par un jury composé des personnes suivantes:

Pierre Poulin
président-rapporteur

Max Mignotte
directeur de recherche

Bruno Tremblay
codirecteur

Sébastien Roy
membre du jury

Mémoire accepté le

RÉSUMÉ

L'imagerie par satellite est utilisée par les océanographes et climatologues pour détecter et quantifier les chenaux, d'immenses craques dans la banquise de l'océan Arctique. Ces activités sont reliées à leurs recherches sur le réchauffement de la planète.

Afin de faciliter leur travail, ce projet de maîtrise présente d'abord un procédé pour l'extraction, la préparation et la visualisation d'images du satellite artificiel Aqua. Il présente aussi des filtres passe-haut rehausseurs de coutours et leurs effets sur ces images de l'océan Arctique dans le but de bien repérer les chenaux créés lorsque la banquise se sépare. Ensuite, ce travail présente l'effet de filtres de classification sur ces images dans le but de reconnaître automatiquement ces chenaux.

Finalement, ce travail offre une suite de logiciels pour travailler avec ces images et une bibliothèque *Open Source* pour le traitement d'image avec le langage de programmation Java. Cette suite comprend des programmes pour extraire et préparer les images. Elle comprend aussi d'autres programmes pour visualiser, restaurer et classifier le contenu de ces images.

Mots clés : restauration, images satellites, chenaux, télédétection, classification, filtres passe-haut, filtres rehausseurs de contours, k-moyennes, `GUIAnalyser`, `jai-operators`.

ABSTRACT

Satellite imagery is used by ocean and climate scientists to detect and measure leads, huge cracks in the Arctic Ocean sea ice, in activities that are related to their studies on global warming.

To facilitate their work, this master thesis project first presents a process for extracting, preparing and viewing images from the Aqua artifical satellite. Also are presented edge enhancing high-pass filters and their effects on these images to help better find leads. Furthermore, this work presents the effects of classifying filters on these images in the hope to automate the recognition of these leads.

Finally, this work offers a software suite to work with these images as well as an *Open Source* Java library to work with images. This suite contains programs to extract and prepare the images and also to view, restore and classify the content of these images.

Keywords: restoration, satellite images, sea ice leads, teledetection, classification, high pass filters, edge enhancer filters, k-means, `GUIAnalyser`, `jai-operators`.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTINGS

LISTE DES SIGLES

AMSR-E	Advanced Microwave Scanning Radiometer for EOS
EOS	Earth Observing System
EOSDIS	Earth Observing System Data and Information System
FFT	*Fast Fourier Transform*, transformée de Fourier rapide
HDF	Hierarchical Data Format
JAI	Java™ Advanced Imaging
Java	Java™
JAXA	Japan Aerospace Exploration Agency
NASA	National Aeronautics and Space Administration
`c.u.i.i.a`	`ca.umontreal.iro.image.arcticice`

À ma famille.

REMERCIEMENTS

Je tiens à remercier toutes les personnes qui m'ont apporté un support très apprécié lors des travaux reliés à ce mémoire de maîtrise. Je tiens à remercier particulièrement mon directeur Max Mignotte et mon co-directeur Bruno Tremblay pour leur patience et leurs conseils judicieux.

Je tiens aussi à remercier Brian Burkhalter de Sun Microsystems inc. pour ses conseils sur la bibliothèque `JAI` et Karsten Lentzsch de la compagnie jGoodies pour ses conseils sur ses bibliothèques.

Finalement, je tiens à remercier mes amis Denis Allaire et François Paulin pour leurs encouragements ainsi que tous les membres de ma famille : Claude, Rachel, Maxime, Nicolas, Dominique, Karine, Simon, Noémie et Hugo pour leur soutien inconditionnel.

INTRODUCTION

Ce travail de maîtrise présente les travaux faits dans le cadre d'une recherche dans le domaine de la segmentation et de la classification du contenu d'images. En particulier, ce travail s'intéresse surtout à la restauration d'images satellite et à la localisation et à la quantification de la longueur et de la largeur des chenaux dans la banquise de l'océan Arctique à partir de celles-ci. Cette détection et quantification est importante pour les scientifiques qui étudient le comportement de l'océan Arctique puisqu'ils espèrent pouvoir ajouter la dimension de la variation de l'épaisseur de la glace dans leurs modèles afin de prévoir avec plus d'exactitude le premier été durant lequel cet océan sera libre de glace.

La segmentation des chenaux consiste à détecter dans les images des zones filiformes et non-rectilignes. En traitement d'image, ce problème est particulièrement difficile car ces formes d'épaisseur d'environ un pixel sont difficiles à extraire du bruit de l'image, bruit causé entre autres par le manque de données et les nuages. Ce problème est plus difficile que la détection des routes car celles-ci sont généralement rectilignes et de largeur constante (hypothèse facilement modélisée mathématiquement).

Ce travail présente des filtres passe-hauts et rehausseurs de contours ainsi que des logiciels spéciaux qui ont été créés pour ces recherches et qui sont fonctionnels pour continuer celles-ci dans les directions futures.

Le chapitre 1 explique le rôle des chenaux sur le climat Arctique et comment la mesure de leur longueur et leur largeur pourrait contribuer aux modèles des climatologues.

Le chapitre 2 parle des travaux déjà faits dans le domaine, que ce soit des travaux dans le domaine du traitement d'image ou travaux dans le domaine de la modélisation du climat.

Le chapitre 3 discute des images qui seront utilisées, de leur provenance, de leur format et ainsi que des traitements préliminaires afin de pouvoir les exploiter.

Le chapitre 4 explique les divers filtres utilisés pour nettoyer les images. Ces

filtres contiennent des filtres classiques et mais aussi des filtres utilisant plusieurs images consécutives puisque les images forment une séquence temporelle. D'autres filtres sont aussi explorés pour classifier les images ainsi nettoyées.

Le chapitre 5 présente les résultats des divers filtres de nettoyage ainsi que les divers filtres de classification.

Le chapitre 6 présente les divers logiciels utilisés durant ce travail de maîtrise. Ces logiciels peuvent être utilisés pour continuer le travail ou encore reproduire les résultats.

Le chapitre 7 offre différentes avenues pour continuer le travail de cette maîtrise.

Les annexes contiennent les manuels d'utilisation ainsi qu'un survol de l'architecture des logiciels présentés au chapitre 6.

CHAPITRE 1

LA PROBLÉMATIQUE

Les scientifiques ont longtemps étudié le comportement de la banquise de l'océan Arctique et en particulier les chenaux qui apparaissent quand la glace se fend et expose alors l'eau chaude de l'océan (environ -2° C) à l'air froid de l'atmosphère (environ -25° C) [15]. Bien que ces chenaux ne représentent qu'une petite partie de la surface de la région Arctique, ils sont les grands responsables des échanges de chaleur dans cette région [16], la glace épaisse de la banquise étant un excellent isolant.

Les chenaux sont des fissures naturelles dont la largeur varie de quelques mètres à quelques kilomètres, se produisant dans la glace de l'océan Arctique lorsque des plaques de glace se séparent à cause de courants marins et de la force du vent sur des milliers de kilomètres.

FIG. 1.1 – Un chenal en train de se reglacer (image tirée de [15]).

La plupart des études sur les chenaux sont des études locales, c'est-à-dire réalisées sur une petite surface de l'océan Arctique, à l'aide de senseurs aéroportés ne fournissant que des données transects[1] [14,15]. Les scientifiques sont maintenant intéressés à identifier les chenaux sur une échelle plus globale, celle sur l'ensemble de l'océan Arctique, afin d'étudier le réchauffement global de notre planète. Une étude [15] des données

[1]Les données transects sont des données linéaires recueillies le long d'un trajet, dans notre cas recueillies le long du trajet de l'aéronef transportant les senseurs.

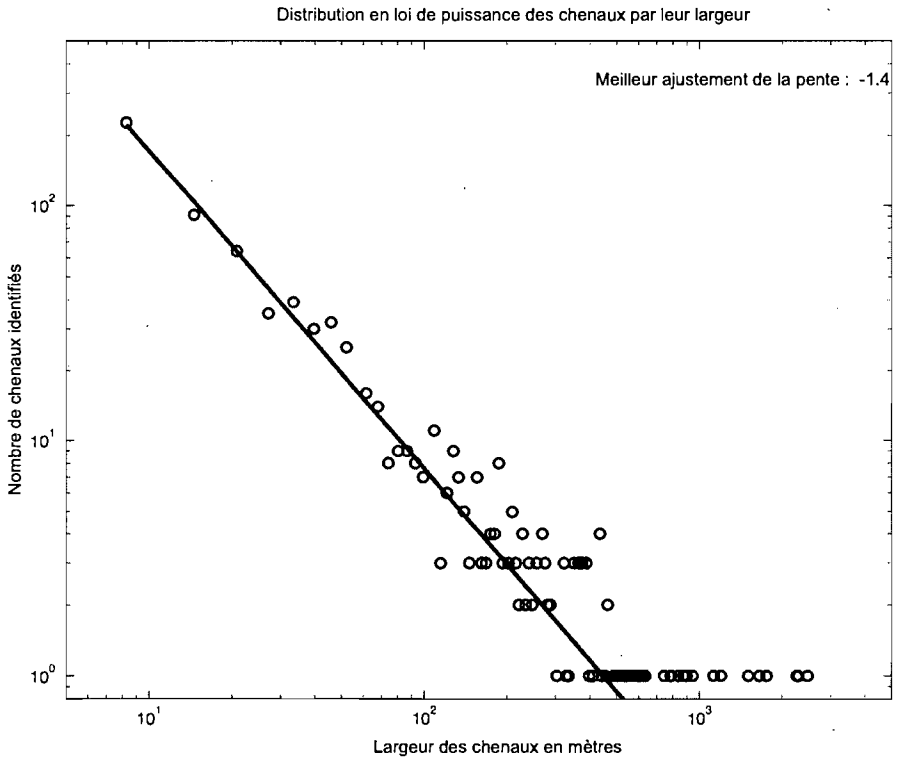


FIG. 1.2 – Distribution de la largeur des chenaux identifiés dans les données transects du *Arctic Lead Experiment* (image tirée de [15]).

recueillies lors du *Arctic Lead Experiment* en 1992 démontre que la distribution de la largeur des chenaux suit la loi de puissance $N(w) \propto w^{-1,4}$, illustrée à la figure 1.2. Cette démonstration corrobore une même constatation sur la distribution des failles dans la croûte terrestre [17]. Pour recenser la population totale de chenaux, il n'est donc pas nécessaire de tous les identifier mais seulement d'identifier les plus gros et, grâce à la loi de puissance, il est alors possible d'estimer la population des plus petits chenaux. Les chenaux seront détectés en analysant les images micro-ondes de l'océan Arctique recueillies par le satellite Aqua (cf. chapitre 3).

Les modèles utilisés pour prédire la disparition de la glace de l'océan Arctique

durant l'été se basent sur les variations de l'étendue de la banquise. Par contre, aucun modèle ne tient compte des variations d'épaisseur de la glace, ces données étant presque inexistantes. Les scientifiques veulent pouvoir ajouter cette dimension dans leur modèle. L'épaisseur de la glace est très difficile à mesurer directement[2] mais grâce aux équations de ses propriétés mécaniques[3], qui dépendent toutes de l'épaisseur, il serait alors possible d'estimer la variation dans le temps de l'épaisseur de la banquise en calculant la variation du rapport longueur-largeur des chenaux.

En connaissant l'épaisseur de la glace, il est alors possible d'observer des variations et de les inclure dans de nouveaux modèles afin de mieux prédire le moment où l'océan Arctique sera libre de glace.

[2]Des sous-marins font parfois des relevés transects de l'épaisseur de la glace, mais pas régulièrement.

[3]Entre autre la résistance en tension, la compression et le cisaillement.

CHAPITRE 2

LES TRAVAUX DÉJÀ FAITS

Peu de travaux ont été faits sur la détection et l'examen des chenaux [14–16]. De plus, tous ces travaux ont été faits à l'échelle locale, sur une petite région de l'océan Arctique. Le présent projet sur la détection à l'échelle globale des chenaux est donc un travail pionnier dans ce domaine.

Dans le domaine du pré-traitement des images des régions polaires, les filtres spectraux passe-hauts classiques (voir la section 4.1) sont généralement utilisés pour retirer des artéfacts de basses fréquences comme les nuages. Une technique spéciale utilisant un filtre spatial passe-haut pour retirer ces artéfacts existe aussi et sera expliquée à la section 2.1.

Quant à la reconnaissance des chenaux, un autre article utilisant un réseau de neurones est résumé à la section 2.2 tandis qu'un autre article sur la détection des zones filiformes dans des images bruitées l'est à la section 2.3. L'article de la section 2.4 parle quant à elle d'une technique pour la segmentation des routes.

2.1 *An Improved Method for Cloud Removal in ASTER Data Change Detection*

L'article de Feng Chun et al. [10] propose une méthode pour enlever les nuages des images produites par télé-détection, plus spécifiquement les images ASTER produites par le satellite Terra, un satellite cousin du satellite Aqua (voir la section 3.1).

Dans cet article, il est proposé que les nuages soient considérés comme un bruit multiplicatif de basse fréquence. Le bruit de basse fréquence est enlevé facilement par un filtre homomorphique (voir la section 4.1) passe-haut dans le domaine spectral.

Cet article innove en utilisant un filtre homomorphique passe-haut dans le do-

maine spatial, c'est-à-dire qu'aucune transformée de Fourier n'est faite. Ce filtre spatial consiste, en résumé, à soustraire de l'image initiale une version floue de cette même image. La soustraction est faite dans le domaine logarithmique, transformant le bruit multiplicatif en bruit additif.

La méthode de cet article a été reproduite comme le filtre homomorphique spatial pour la partie nettoyage et est décrite à la section 4.1.2.

2.2 *Arctic Sea Ice, Cloud, Water, and Lead Classification Using Neural Networks and 1.6-micrometer Data*

L'article de McIntire, T.J et al. [11] propose une technique utilisant un réseau de neurones pour classifier la glace, les nuages, l'eau et les craques à partir des images en lumière visible du satellite chinois Fengyun-1C [25]. Il est à noter que le satellite Fengyun 1-C a été détruit au début de l'année 2007 lors d'un test de missile anti-satellite [26].

Autant le filtre présenté par [10] a été réalisé comme filtre pour la partie nettoyage, autant cette méthode pourrait être ajoutée aux filtres de classification décrits à la section 4.2.

2.3 *Enhancement of Low-Contrast Curvilinear Feature in Imagery*

L'article de Mark J. Carlotto [12] propose une technique pour découvrir des zones filiformes dans des images bruitées et de bas contraste. Cette technique utilise des points de départ ou encore suppose *a priori* que la zone à détecter traverse l'image, c'est-à-dire qu'il existe une paire de pixels appartenant à la zone sur des côtés opposés de l'image.

Un des buts de ce travail est d'automatiser la détection des chenaux. Aussi, les chenaux dans les images satellites de ce travail ne traversent pas l'image, ils peuvent être partout. Cependant, ce filtre de classification pourrait être utilisé pour la détection par vignette comme expliqué au chapitre 7.

2.4 *Road Extraction From Aerial Images Using a Region Competing Algorithm*

L'article de Miriam Amo et al. [13] propose une technique semi-automatique pour découvrir les routes sur les images satellites en utilisant divers points de départ fournis par un usager. Cette technique suppose qu'une route possède sensiblement la même luminance dans toute l'image et que celle-ci possède une largeur généralement constante de plusieurs pixels sur une distance assez grande. Mathématiquement, la détection de régions filiformes d'épaisseur et de luminance constantes peut facilement se modéliser par des *a priori* ou des régularisations fortes qui permettent de contraindre le problème de détection.

Bien que la plupart des chenaux n'aient une épaisseur que d'un seul pixel ou moins, cet algorithme pourrait être étudié comme filtre de classification supervisé comme décrit au chapitre 7. Il permettrait alors de raffiner l'identification des chenaux lors de la segmentation en demandant à l'utilisateur d'identifier des régions plus propices à les contenir.

2.5 Les travaux de Ron Kwok

Ron Kwok[1] est un scientifique de la NASA ayant beaucoup étudié le comportement de la glace à l'aide de la télédétection. Son groupe étudie particulièrement l'épaisseur de la glace à l'aide, entre autres, des données du satellite canadien RADARSAT. Le satellite RADARSAT produit une image complète du pôle nord une fois tous les trois jours à une résolution maximale de 10 mètres[2].

Ses travaux ont donné naissance au *RADARSAT Geophysical Processor System*[3] qui a produit plusieurs types de cartes de la banquise Arctique. Un type de carte produit est une carte de vecteurs, produit des différences entre deux images consécutives et qui indiquent le déplacement de la banquise. De ces cartes de vec-

[1] `http://experts.nasa.gov/get_expert.php?id=1483`
[2] `http://en.wikipedia.org/wiki/RADARSAT-1`
[3] `http://www-radar.jpl.nasa.gov/rgps/radarsat.html`

teurs sont aussi produites des cartes maillées indiquant avec plus de précision la divergence et la convergence, le cisaillement et la rotation de la glace. En étudiant les mouvements de la glace il est alors possible de trouver des chenaux puisque ces mouvements produisent, et se produisent le long de, ces chenaux. La figure 2.1 illustre un aperçu de ces produits.

Il serait intéressant de voir les résultats des algorithmes de Ron Kwok sur les images AMSR-E quotidiennes utilisées dans ce projet bien que celles-ci soient beaucoup plus grossières que celles de RADARSAT. Aussi, il serait intéressant d'observer le comportement des images RADARSAT dans le procédé de ce projet. Il est à noter que les images du satellite RADARSAT ne sont pas gratuites, contrairement aux images AMSR-E utilisées dans ce projet.

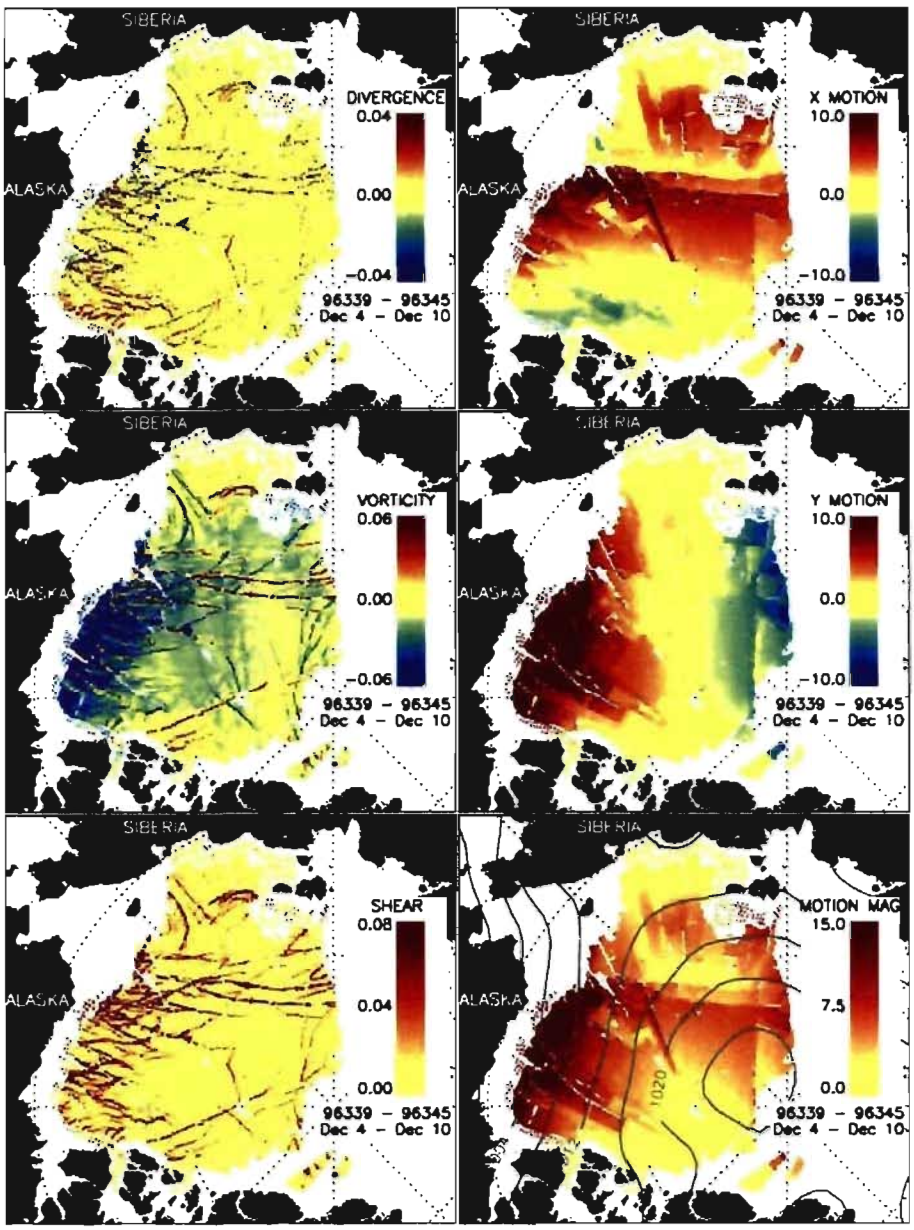


FIG. 2.1 – Un aperçu des résultats des travaux de Ron Kwok : la divergence, le tourbillon (*vorticity*), le cisaillement et le déplacement entre les journées du 4 décembre 1996 et le 10 décembre 1996 (images tirées de `http://www-radar.jpl.nasa.gov/rgps/pdef6/w9697n/pdef6_30_96339_96345_1.html`.

CHAPITRE 3

LES IMAGES

Ce chapitre traite des caractéristiques des images utilisées, soient : leur format, leurs pré-traitements, ainsi que de leur origine, du satellite Aqua aux fichiers utilisés par le programme `GUIAnalyser`.

Les données recueillies durant une journée par le satellite Aqua sont compilées et traitées pour produire quotidiennement un ensemble d'images. Bien que chacun des ensembles d'images peut être utilisé séparément, l'utilisation d'une séquence de ces ensembles donne une plus grande souplesse de traitement car il est possible d'utiliser les ensembles précédents et suivants pour analyser et traiter une image. Plusieurs algorithmes décrits dans le chapitre 4 utilisent une telle séquence pour leur traitement.

3.1 Le satellite

Les images utilisées durant ce projet proviennent du satellite Aqua, un satellite artificiel de recherche scientifique en orbite autour de la Terre lancé le 4 mai 2002. Les données de ce satellite sont utilisées pour étudier principalement les précipitations, l'évaporation et le cycle de l'eau [1].

FIG. 3.1 – Le satellite Aqua.

Ce satellite est le second satellite majeur de la constellation *Earth Observing System* (EOS) [2, 3], un groupe de satellites sur une même

orbite héliosynchrone, tous passant au-dessus d'un même point sur la Terre dans un intervalle de quinze minutes, permettant des observations quasi-simultanées de ce même endroit. Une orbite héliosynchrone est une orbite géocentrique qui permet à un satellite de repasser au dessus d'un même point sur la Terre à la même heure solaire locale, d'orbite en orbite [20, 24].

3.2 Les images de base

Le satellite Aqua produit plusieurs ensembles de données. L'ensemble d'intérêt pour ce projet sont les données *AE_SI6 - Brightness temperatures*, c'est-à-dire les températures des zones polaires, mesurées par l'instrument *Advanced Microwave Scanning Radiometer* (AMSR-E) [5]. Les données sont recueillies par deux senseurs polarisés mesurant les micro-ondes de fréquence 89 Ghz et transformées en une collection de douze images d'une résolution de 6,25 kilomètres. Ces douze images sont assemblées, avec leurs méta-données, dans un seul fichier de 46,3 méga-octets et mises dans le site de dépôt[1] de l'ensemble de données *AE_SI6*.

3.3 Les données originales

Les fichiers du site de dépôt de *AE_SI6* sont encodés dans le format HDF [7]. En utilisant la suite de logiciels HDF [8], en particulier l'outil `hdp`, il est possible de connaître et d'extraire le contenu d'un fichier HDF. La figure 3.2 illustre une partie des méta-données d'un fichier HDF. Une alternative pour explorer ces fichiers HDF et obtenir ces méta-données est d'utiliser l'application Java `HDFView`[2].

Dans notre cas, tous ces fichiers HDF contiennent, entre autres, douze images dont le nom est encodé avec une notation particulière selon trois critères :

Le pôle Le satellite accumule des données pour l'océan Arctique au pôle nord et pour le continent Antarctique au pôle sud. Les images de la région Arctique

[1] `ftp://n0dps01u.ecs.nasa.gov/SAN/AMSA/AE_SI6.001/`
[2] `http://www.hdfgroup.org/hdf-java-html/hdfview/`

```
$ hdp dumpsds -h AMSR_E_L3_SeaIce6km_B06_20070506.hdf

Variable Name = SI_06km_NH_89H_DAY
         Index = 5
         Type= 16-bit signed integer
         Ref. = 10
         Rank = 2
         Number of attributes = 0
         Dim0: Name=YDim:NpPolarGrid06km
                  Size = 1792
                  Scale Type = number-type not set
                  Number of attributes = 0
         Dim1: Name=XDim:NpPolarGrid06km
                  Size = 1216
                  Scale Type = number-type not set
                  Number of attributes = 0
```

FIG. 3.2 – Les méta-données de l'image `SI_06km_NH_89H_DAY`.

sont dénotées avec le marqueur `NH` et celles de la région antarctique par le marqueur `SH`.

Le senseur Le satellite utilise deux senseurs polarisés pour accumuler les données. Les images produites par le senseur de polarité horizontale sont dénotées avec le marqueur `89H` et celles produites par le senseur de polarité verticale avec le marqueur `89V`. Le marqueur `89` indique que les senseurs opèrent dans la bande des 89 GHz.

L'orbite L'orbite héliosynchrone du satellite fait que le satellite accumule des données sur la surface éclairée de la Terre durant son orbite descendante et accumule des données sur la surface ombragée durant son orbite ascendante. Les images de l'orbite descendante sont dénotées avec le marqueur `DES`, celles de l'orbite ascendante par le marqueur `ASC`, et la moyenne de ces deux orbites par le marqueur `DAY`.

D'après ces critères, l'image `SI_06km_NH_89H_DAY` de la figure 3.2 est celle de l'hémisphère nord (`NH`), produite par le senseur de polarité horizontale (`89H`) et de la moyenne de la journée (`DAY`). Ces méta-données indiquent aussi que l'image est de dimension 1216 pixels par 1792 pixels et que le type de ces pixels est un entier signé de 16 bits. Les spécifications du format [9] donnent plus de détails sur les autres attributs.

```
$ hdp dumpsds -n SI_06km_NH_89H_DAY -o 20070506-SI_06km_NH_89H_DAY.bin \
  -b AMSR_E_L3_SeaIce6km_B06_20070506.hdf
$ hdp dumpsds -n SI_06km_NH_89V_DAY -o 20070506-SI_06km_NH_89V_DAY.bin \
  -b AMSR_E_L3_SeaIce6km_B06_20070506.hdf
```

Listing 3.1 – Extraction des images `SI_06km_NH_89H_DAY` et `SI_06km_NH_89V_DAY` du fichier `AMSR_E_L3_SeaIce6km_B06_20070506.hdf`.

Pour créer les images qui seront par la suite analysées, il faut extraire les données du fichier HDF (cf. section 3.4), remplir les zones sans données (cf. section 3.6) et ensuite unifier les images du senseur de polarité horizontale et de polarité verticale (cf. section 3.7).

3.4 L'extraction des images

Dans chacun des fichiers HDF, les deux images d'intérêt sont celles des moyennes de la journée dans l'hémisphère nord : l'image du senseur de polarité horizontale `SI_06km_NH_89H_DAY` et l'image du senseur de polarité verticale `SI_06km_NH_89V_DAY`. Ces deux images sont extraites par les deux commandes (pour la journée du 6 mai 2007) illustrés par le listing 3.1. Ces deux commandes créent les fichiers contenant les images brutes, sans méta-données, `20070506-SI_06km_NH_89H_DAY.bin` pour l'image du senseur de polarité horizontale et `20070506-SI_06km_NH_89V_DAY.bin` pour l'image du senseur de polarité verticale.

3.5 La zone d'intérêt

La figure 3.3 illustre une image telle que fournie par la NASA. Dans le cadre de ce projet, qui est d'identifier les chenaux dans la glace, cette image contient beaucoup trop d'information. En partant du haut et en allant dans le sens horaire, il est possible de voir une grande partie de la Russie, la Scandinavie, le Groenland, le nord du Canada et l'Alaska.

Les glaces les plus susceptibles d'avoir des chenaux sont les glaces au nord de

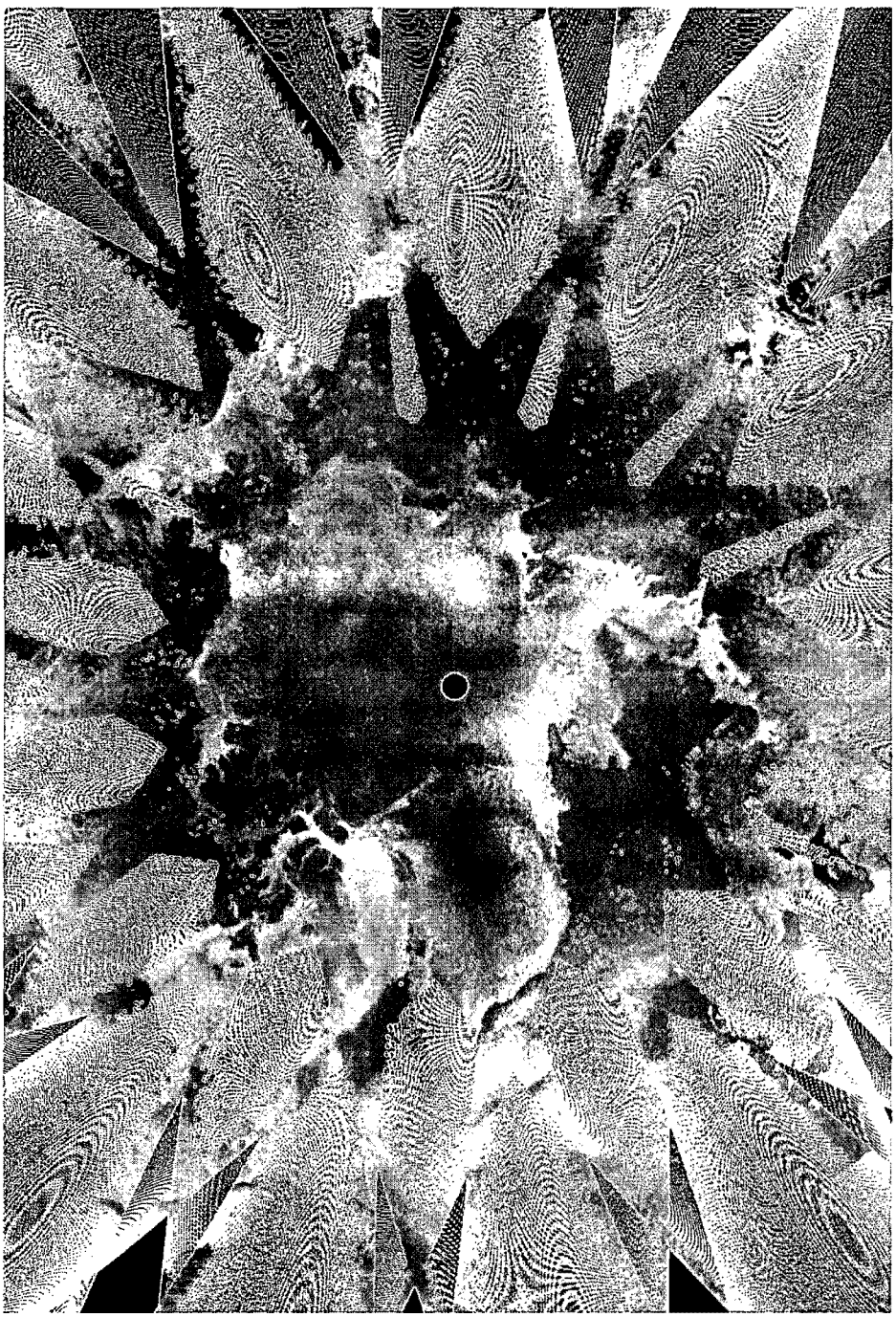


FIG. 3.3 – L'image originale du senseur horizontal de la journée du 21 février 2007. Il est à noter que les moirés ne sont pas des artéfacts d'impression.

l'Alaska, endroit où sont situées les plus vieilles glaces[3]. Cet endroit est donc la zone d'intérêt.

Puisque plusieurs des algorithmes décrits au chapitre 4 utiliseront la version rapide de la transformée de Fourier (FFT), la zone d'intérêt choisie est une image carrée ayant une hauteur et une largeur de 512 ou 256 pixels (i.e. une puissance de deux).

L'image ayant une dimension de 512 pixels couvre presque tout l'océan Arctique, elle est illustrée à la figure 3.4 tandis que l'image de 256 pixels à la figure 3.5 ne couvre que les anciennes glaces. Il est à noter que les images de ces deux figures ont subi le pré-traitement complet expliqué dans ce chapitre et sont des images de pré-visionnement.

3.6 Le remplissage des pixels noirs

Les images finales contenues dans le fichier HDF sont un assemblage des données prises par le satellite au cours de ses orbites pendant une journée. Due à l'orbite héliosynchrone, le satellite ne passe pas directement au-dessus du pôle nord géographique, créant un trou noir dans les données. De plus, dans les zones visitées par le satellite, certaines régions n'ont pas été couvertes, créant ainsi de petits pixels noirs. Ces petits pixels noirs sont plus apparents plus la région est éloignée du pôle nord. La figure 3.3, une image originale, montre très bien ces pixels noirs (qui ne sont en aucun cas des artéfact d'impressions, mais des zones n'ayant pas de valeur, c'est-à-dire des moirés).

Pour remplir ces pixels noirs, l'hypothèse de proximité a été choisie, c'est-à-dire que la valeur du pixel noir est probablement proche de celle de ses voisins. Une analyse de l'histogramme de l'image montre que la valeur 0 est assignée aux points noirs et que cette valeur est beaucoup plus petite que la valeur du niveau de gris associée à la température la plus basse.

L'algorithme de remplissage dresse d'abord la liste de tous les pixels noirs (les

[3]Discussion personnelle avec Bruno Tremblay.

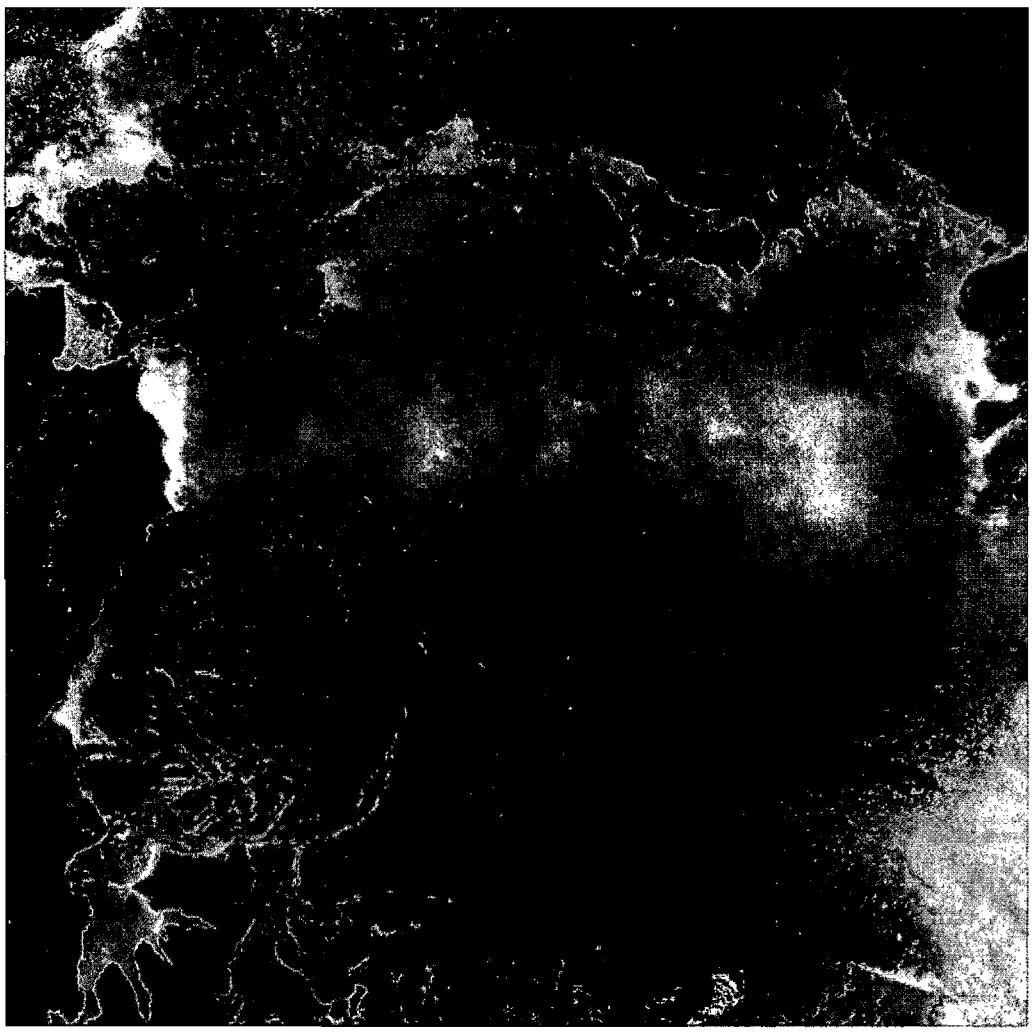


FIG. 3.4 – La zone d'intérêt de taille 512 par 512 pixels de la journée du 21 février 2007. Cette image contient dans la zone inférieure gauche la zone d'intérêt de taille 256 par 256 pixels (figure 3.5).

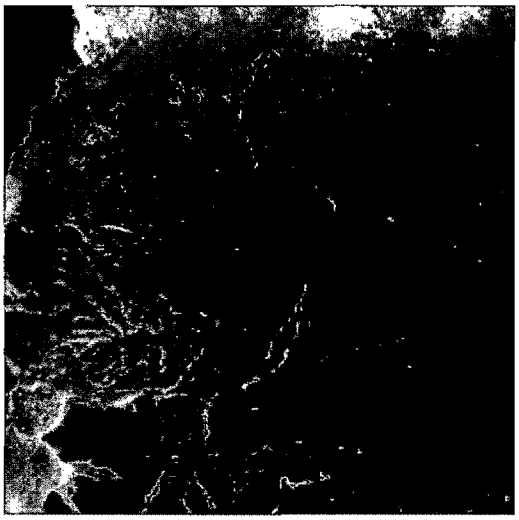


FIG. 3.5 – La zone d'intérêt de taille 256 par 256 pixels de la journée du 21 février 2007. Cette zone est la partie inférieure gauche de la figure 3.4

pixels ayant une valeur de zéro en niveau de gris)[4]. Ensuite, tant qu'il y a des points dans cette liste, le premier point est retiré. Les quatre voisins - haut, bas, gauche et droite - sont trouvés et leurs valeurs analysées. Si deux ou plus de ses voisins ont une valeur non-noire, la valeur du pixel noir est remplacée par la moyenne de ses voisins non-noirs. S'il n'y a aucun voisin non-noir, ou seulement un, le point est remis à la fin de la liste ; ses voisins seront de nouveau analysés une fois que tous les autres points noirs auront été traités. Le listing 3.2 illustre l'algorithme de remplissage.

L'algorithme est garanti de converger tant que les pixels noirs sont complétement entourés de pixels non-noirs (il y a alors toujours au moins un pixel avec deux voisins).

Dans les zones d'intérêt, le trou noir du pôle est toujours entouré de pixels non-

[4]Durant la réalisation de l'algorithme de remplissage, l'opérateur `extrema` de la bibliothèque `JAI` a été utilisé afin d'obtenir une liste des points noirs. Une analyse des points a démontré des erreurs, entre autres que des points dans la liste n'étaient pas noirs et que des points noirs n'étaient pas dans la liste. L'algorithme a été remplacé par une version n'utilisant pas cet opérateur et ce bogue a été rapporté aux auteurs. Le numéro du bogue est 97 (`https://jai-core.dev.java.net/issues/show_bug.cgi?id=97`).

```
Faire la liste de tous les points noirs
Pour chacun des points noirs
   Retirer le point noir de la liste
   Trouver les 4-voisins
   Calculer le nombre de voisins non-noir
   Si deux ou plus voisins non-noirs
      Remplacer le point noir par la moyenne
   Sinon
      Remettre le point à la fin de la liste
```

Listing 3.2 – Pseudo-code pour le remplissage des trous noirs.

noirs. Les autres trous noirs sont tous petits et aussi entourés. Ces deux observations impliquent qu'il y aura toujours au moins un point à chaque itération qui possède au moins deux voisins, point qui sera rempli et retiré de la liste. La figure 3.4 montre un remplissage complet, le trou du pôle (dans le quadrant inférieur droit) et les petits trous ayant tous été remplis.

Afin de faciliter l'algorithme et d'éviter des pixels manquants lors du remplissage de pixels noir sur les côtés de l'images, celle-ci est d'abord découpée pour laisser une bordure d'un seul pixel autour de la zone d'intérêt, c'est-à-dire que pour créer une image carrée de 512 pixels de côté, une première image carrée de 514 pixels de côté est découpée. Alors seulement les points noirs dans la zone d'intérêt sont listés, les points noirs dans la bordure ne le sont pas.

3.7 L'unification

Une fois les zones d'intérêt remplies, il ne reste qu'à unifier l'image du senseur de polarité horizontale et celle du senseur de polarité verticale. Cette dernière étape est très facile, il suffit de créer une image qui est la somme des deux images traitées.

Une analyse de l'histogramme (i.e. section 3.6) montre que la valeur minimale des pixels est zéro, la valeur des trous noirs, et que la valeur maximale est d'au plus 3000. Il est donc possible d'additionner les images sans perdre de l'information en commettant un débordement (*overflow* en anglais), le type des pixels étant des entiers de 16 bits.

L'image de l'unification est ensuite conservée sur disque, dans le même format que les images extraites afin de ne pas perdre d'information (la conservation sous

d'autres formats entraînent la conversion des pixels de 16 bits vers 8 bits, une perte d'information).

Finalement, des images de pré-visionnement des différentes étapes, recalées[5] pour augmenter le contraste, ont été sauvegardées en format PNG afin de pouvoir les manipuler avec des outils de visionnement classiques.

[5]L'opération de recalage est une transformation linéaire de la valeur des pixels vers l'intervalle conventionnel allant de 0 et se terminant à 255.

CHAPITRE 4

LES FILTRES

Ce chapitre traite des filtres utilisés pour nettoyer les images et classifier celles-ci une fois nettoyées. En premier lieu, l'image originale est toujours nettoyée par un filtre passe-haut pour retirer les nuages et accentuer les chenaux. L'image résultante est ensuite passée dans un filtre de classification non-supervisé afin de donner une classe à chacun des pixels.

Il y a deux types de filtres de nettoyage, les filtres spatiaux et les filtres spectraux. Les filtres spatiaux restent dans le même domaine que l'image et travaillent directement sur celle-ci. Les filtres spectraux par contre travaillent dans le domaine de la transformée de Fourier.

Tous les filtres spectraux ont une équation de la forme :

$$H(u,v) = (\gamma_{MAX} - \gamma_{MIN})f(u,v) + \gamma_{MIN} \tag{4.1}$$

Le terme $f(u,v)$ est la valeur initiale du filtre passe-haut échelon aux coordonnées u et v tandis que $H(u,v)$ est la valeur finale du filtre à ces coordonnées. Dans le cas des filtres en trois dimensions prenant une suite d'images, la troisième dimension est dénotée w.

Dans toutes les équations des filtres spectraux, $f(u,v)$ est initialement borné entre 0 et 1, mais les termes γ_{MIN} et γ_{MAX} peuvent modifier ces bornes pour donner à l'équation du filtre un nouvel intervalle. Quand ces nouvelles bornes sont telles que $\gamma_{MIN} > 0$ et $\gamma_{MAX} > 1$, le filtre passe-haut obtenu est en fait un filtre rehausseur de contours.

Toutes les équations des filtres spectraux représentent des filtres centrés[1] et circulaires dans le cas des filtres en deux dimensions puisque ceux-ci sont symétriques.

[1]Un filtre est centré quand la position du pixel représentant la moyenne de l'image est placée au centre de l'image.

Par contre, pour les équations des filtres à trois dimensions, chacune des dimensions est indépendante. Pour respecter l'aspect symétrique des deux premières dimensions u et v, le logiciel `GUIAnalyser` donne toujours à ces deux premières dimensions la même valeur.

4.1 Les filtres de nettoyage

Les premiers filtres utilisés sont les filtres de nettoyage, les filtres appliqués sur les images de base dans le but d'accentuer les chenaux et de retirer les nuages.

Tous les filtres pouvant opérer sur une seule image à la fois sont dénommés des filtres à deux dimensions. Cependant, puisque les images forment une séquence, certains filtres ont aussi une version tridimensionnelle et utilisent les images voisines dans la séquence.

Finalement, tous les filtres spectraux, les filtres opérant sur une transformée de Fourier, ont une version normale et une version homomorphique. La version normale suppose que le bruit causé par les nuages est additif tandis que la version homomorphique suppose que ce même bruit est multiplicatif. Pour séparer un bruit multiplicatif, la version homomorphique crée d'abord une image logarithmique, c'est-à-dire une image dont la valeur de chacun des pixels est le logarithme de la valeur originale (avec une valeur minimale originale supérieure à 1), avant d'appliquer la transformée de Fourier et ensuite de faire l'exponentielle de l'image fournie par la transformée inverse après le traitement par le filtre spectral.

4.1.1 Filtre identité

Le premier filtre réalisé et étudié a été le filtre identité. Le filtre identité est le filtre qui n'a aucune fonction, l'image d'origine et l'image filtrée sont identiques.

À première vue, ce filtre semble inutile, mais il permet l'utilisation de l'image d'origine comme image filtrée, que ce soit pour comparer les résultats avec les autres filtres ou encore parce qu'elle est déjà aussi belle que d'autres images filtrées.

4.1.2 Filtre homomorphique spatial

Le filtre homomorphique spatial est un filtre 2D et une réalisation de l'algorithme expliqué dans l'article [10] et résumé à la section 2.1. Ce filtre a deux paramètres : la taille du filtre boîte et le poids de l'image floue lors de la soustraction.

La première étape consiste à faire passer dans le domaine logarithmique l'image originale après s'être assuré qu'aucun pixel n'avait une valeur inférieure à 1.

La seconde étape consiste à appliquer un filtre boîte sur l'image logarithmique pour produire une image floue. Un filtre boîte est un noyau de convolution dans lequel toutes les valeurs ont le même poids. Dans notre cas précis, le noyau est carré et sa taille est le premier paramètre du filtre. L'image floue est ensuite pondérée par un poids α (idéalement compris entre $0,6$ et $0,9$). Un poids trop important et les détails des régions sans nuages seront trop rehaussés, un poids trop faible et les nuages ne seront pas vraiment retirés. Ce poids α est le deuxième paramètre du filtre.

La troisième étape consiste à soustraire l'image floue pondérée de l'image logarithmique, produisant ainsi une image dans laquelle les basses fréquences sont retirées, ne laissant que les hautes fréquences.

Finalement, l'image est ramenée dans le domaine spatial par l'opération d'exponentiation après s'être assuré qu'aucun pixel n'avait une valeur négative.

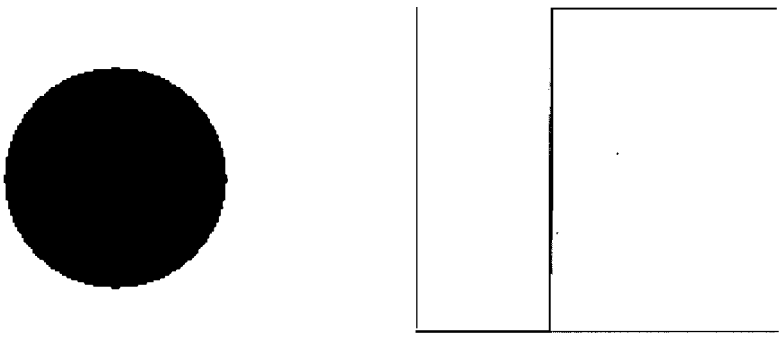
4.1.3 Filtre passe-haut idéal

Le filtre passe-haut idéal est un filtre spectral avec une version en deux dimensions et une version en trois dimensions.

4.1.3.1 Filtre passe-haut idéal 2D

Le filtre passe-haut idéal est défini comme suit en deux dimensions [19] :

$$f(u,v) = \begin{cases} 0 & \text{si } D(u,v) \leq D_0 \\ 1 & \text{si } D(u,v) > D_0 \end{cases} \tag{4.2}$$



(a) (b)

FIG. 4.1 – Le filtre passe-haut idéal représenté (a) par une image et (b) par sa coupe (l'axe vertical représente la distance unité et l'axe horizontal représente la fréquence).

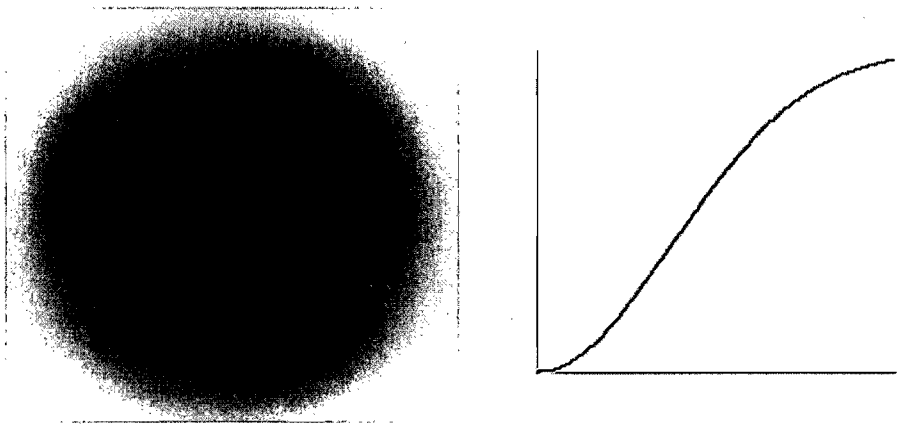
C'est-à-dire que le filtre laisse passer toutes les fréquences $D(u,v)$ qui sont plus distantes du centre que la fréquence de coupure D_0 et bloque toutes les autres fréquences plus proches. La figure 4.1, tirée du logiciel `GUIAnalyser` illustre ce filtre.

4.1.3.2 Filtre passe-haut idéal 3D

Le filtre passe-haut idéal en trois dimensions est défini comme suit :

$$f(u,v,w) = \begin{cases} 0 & \text{si } \frac{D(u)^2}{D_{0_u}^2} + \frac{D(v)^2}{D_{0_v}^2} + \frac{D(w)^2}{D_{0_w}^2} < 1 \\ 1 & \text{autrement} \end{cases} \tag{4.3}$$

C'est-à-dire que le filtre laisse passer toutes les fréquences qui sont plus distantes du centre que la fréquence de coupure D_0 et bloque toutes les autres fréquences plus proches. Autant la version 2D est illustrée par un cercle, autant la version 3D peut être illustrée par une sphère.



(a) (b)

FIG. 4.2 – Le filtre passe-haut gaussien représenté (a) par une image et (b) par sa coupe (l'axe vertical représente la distance unité et l'axe horizontal représente la fréquence).

4.1.4 Filtre passe-haut gaussien

Le filtre passe-haut gaussien est un filtre spectral avec une version en deux dimensions et une version en trois dimensions.

4.1.4.1 Filtre passe-haut gaussien 2D

Le filtre passe-haut gaussien en deux dimensions est défini par l'équation [19] :

$$f(u,v) = 1 - \exp^{-c\left(\frac{D^2(u,v)}{2D_0^2}\right)} \tag{4.4}$$

dans laquelle $D^2(u,v)$ est le carré de la distance au centre du filtre et D_0^2 le carré de la fréquence de coupure. Le paramètre c contrôle la pente de la gaussienne. La figure 4.2, tirée du logiciel `GUIAnalyser` illustre ce filtre.

4.1.4.2 Filtre passe-haut gaussien 3D

Le filtre passe-haut gaussien en trois dimensions est défini par l'équation :

$$f(u,v,w) = 1 - \exp^{-c\left(\frac{D^2(u)}{2D_{0_u}^2} + \frac{D^2(v)}{2D_{0_v}^2} + \frac{D^2(w)}{2D_{0_w}^2}\right)} \tag{4.5}$$

dans laquelle D^2 est le carré de la distance au centre du filtre pour la dimension et D_0^2 le carré de la fréquence de coupure dans cette dimension. Le paramètre c contrôle la pente de la gaussienne. Autant le filtre passe-haut 2D est imagé par un V gaussien, autant le filtre passe-haut 3D peut être imagé par un *trou noir* gaussien.

4.1.5 Filtre de Butterworth

Le filtre passe-haut de Butterworth est un filtre spectral en deux dimensions défini par l'équation [19] :

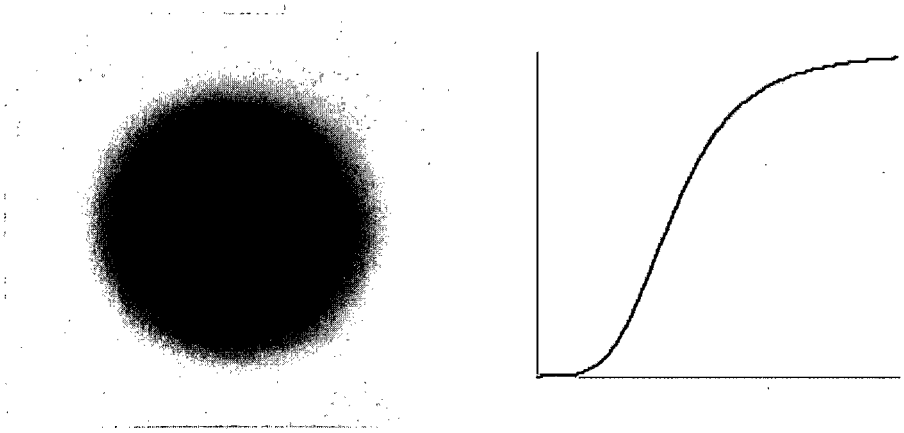
$$f(u,v) = \frac{1}{1 + \left[\frac{D_0}{D(u,v)}\right]^{2n}} \tag{4.6}$$

dans laquelle $D(u,v)$ est la distance au centre du filtre et D_0 la fréquence de coupure. Le paramètre n représente l'ordre du filtre. La figure 4.3, tirée du logiciel `GUIAnalyser` illustre ce filtre.

Ce filtre n'a pas de version tridimensionnelle.

4.1.6 Filtre médian temporel

Le filtre médian temporel est un filtre qui produit une image dont la valeur de chacun des pixels est la médiane des pixels correspondants dans toutes les images de la série.



(a) (b)

FIG. 4.3 – Le filtre passe-haut de Butterworth représenté (a) par une image et (b) par sa coupe (l'axe vertical représente la distance unité et l'axe horizontal représente la fréquence).

4.2 Les filtres de classification

Les seconds filtres utilisés sont les filtres de classification. Ces filtres prennent l'image filtrée et tentent de donner un sens aux valeurs des pixels en regroupant dans une même classe les pixels ayant des traits en commun.

Les filtres décrits dans ce mémoire sont des filtres de classification non-supervisés, c'est-à-dire que les filtres déterminent seuls, sans apport extérieur, la classe de chacun des pixels. Par opposition, le filtre décrit dans [11] est un filtre supervisé, il a été entraîné à détecter les chenaux.

4.2.1 Filtre k-moyennes

Le filtre de classification des k-moyennes est un filtre qui assigne à chacun des éléments d'un ensemble une classe selon une fonction d'évaluation.

Dans notre cas, les éléments sont les pixels de l'image et la valeur à évaluer dans la plupart des cas est le niveau de gris ou la couleur du pixel lui-même, mais cette valeur pourrait aussi dépendre du voisinage.

Le filtre de classification des k-moyennes commence en choisissant k centres de classe au hasard. Ces centres sont choisis dans l'espace d'évaluation. Une fois ces centres choisis, les trois étapes suivantes sont accomplies :

1. Pour chacun des éléments de l'ensemble, générér le vecteur d'évaluation.
2. Pour chacun des éléments du vecteur d'évaluation, trouver le centre de classe le plus près et assigner à l'élément qu'il représente la classe de ce centre.
3. Une fois tous les éléments classés, recalculer les centres en faisant la moyenne des vecteurs d'évaluation associés à cette classe.

Ces trois étapes sont recommencées jusqu'à convergence, c'est-à-dire jusqu'à ce qu'aucun élément ne change de classe durant une itération. Alternativement, il est possible d'arrêter une fois qu'un certain nombre d'itérations aient été accomplies, mais en général seulement un petit nombre d'itérations est nécessaire [18]. Dû au hasard dans le placement initial des centres de classes, il se peut que plusieurs classifications différentes soient obtenues à partir d'un même ensemble de départ.

Il est important de faire la différence entre les éléments de l'ensemble et les vecteurs d'évaluation. Dans le cas des images, les éléments sont les pixels de l'image et les vecteurs d'évaluation peuvent être n'importe quelle valeur associée à ce pixel. Cette valeur peut être simplement la couleur du pixel mais peut être aussi une valeur plus complexe qui tient en compte la position dudit pixel ou même ses voisins.

La figure 4.4 contient le pseudocode décrivant l'algorithme des k-moyennes inclus dans le fichier du code source de la classe `KMeansOpImage`[2], la classe contenant le code réalisant l'algorithme des k-moyennes comme opérateur `JAI`.

Dans cette version, le vecteur d'évaluation est demandé à chacune des itérations. Cette demande est délibérée puisqu'elle permet de générer des vecteurs différents d'itération en itération. Cette capacité est par contre dangereuse puisqu'elle interfère avec la garantie de convergence.

[2] `ca.forklabs.media.jai.opimage.KMeansOpImage`

```
// for each iteration until stability :
// 1) sort and reset the centers - the centers are already normalized
// 2) for each row
//      for each column
//         store the pixel vector
// 3) normalize each pixel vector
// 4) for each pixel vector
//   4a) calculate its distance from each center
//   4b) find the center that is closest
//   4c) update the data structure for the next iteration centers
//   4d) set the pixel value of the sink to the cluster number and
//       record if any change occured
// 5) calculate the position of the new centers
```

FIG. 4.4 – Commentaires accompagnant le code du filtre des k-moyennes dans la classe `KMeansOpImage`.

4.2.1.1 Filtre k-moyennes original

La version originale du filtre des k-moyennes n'utilise que la couleur du pixel comme vecteur à classifier. Ni la position du pixel, ni les pixels avoisinants n'entrent en ligne de compte. Cette version est en fait la fonction d'évaluation utilisée la plus souvent lors de l'explication de l'algorithme.

La fonction d'évaluation par défaut incluse dans l'opérateur `kmeans` ne tient compte que de la couleur du pixel. C'est cette fonction que la version originale du filtre utilise.

4.2.1.2 Filtre k-moyennes contextuel

La version du filtre des k-moyennes dite version contextuelle prend comme hypothèse que les chenaux dans l'image correspondent à une région chaude et sont donc représentés par une ligne continue pâle (d'une forte luminance) d'une largeur souvent égale à un seul pixel dans la banquise froide qui, elle, est représentée par une zone uniforme foncée (d'une faible luminance).

La fonction d'évaluation illustrée à la figure 4.5 et au listing 4.1 est celle utilisée pour modéliser cette hypothèse et retourner la valeur à classer. L'évaluation se fait dans ce cas avec un vecteur à deux dimensions, la première valeur représente la température à cet endroit et la seconde représente l'information contextuelle, c'est-à-dire la différence de température maximale existant avec les couples de pixels

$s1$	$s2$	$s3$
$s4$	s	$s5$
$s6$	$s7$	$s8$

FIG. 4.5 – Identification du noyau de l'algorithme des k-moyennes contextuel.

```
a = min(|s - s1|, |s - s8|)
b = min(|s - s2|, |s - s7|)
c = min(|s - s3|, |s - s6|)
d = min(|s - s4|, |s - s5|)

f = max(a, b, c, d)

return (s, f)
```

Listing 4.1 – Pseudocode du calcul du vecteur de l'algorithme des k-moyennes contextuel.

voisins (voisin du 2^e ordre).

La formule pour trouver la température à l'endroit du pixel est simple, il s'agit de la couleur du pixel (i.e. sa luminance).

La formule pour trouver la différence de température avec les pixels voisins est expliquée dans le pseudocode de la figure 4.5. L'information contextuelle exploite simplement le fait que si deux voisins opposés ($s1$ et $s8$ par exemple) ont une grande différence avec le pixel central s, alors il y a une forte chance d'avoir une ligne passant par le pixel milieu.

Le vecteur résultat, la température du pixel central en première position et la différence de température des pixels avoisinants en seconde position, a donc quatre interprétations, quatre classes qui seront identifiés par l'algorithme des k-moyennes :

↓ **température** ↓ **différence** Une température froide et une petite différence veut dire que le pixel représente une surface appartenant à la banquise, il est froid et ses voisins sont aussi froids.

↓ **température** ↑ **différence** Une température froide et une grande différence veut dire que le pixel correspond à une côte de la banquise ou d'un iceberg,

un morceau de glace entouré d'eau, il est froid et ses voisins sont chauds.

↑ **température** ↓ **différence** Une température chaude et une petite différence veut dire que le pixel est sur l'océan, il fait chaud et ses voisins sont aussi chauds.

↑ **température** ↑ **différence** Une température chaude et une grande différence veut dire que le pixel appartient à un chenal, il est chaud et ses voisins sont froids.

Dans le cadre de ce projet, l'interprétation la plus recherchée est la quatrième interprétation, celle disant que le pixel est un chenal.

En fixant le nombre de classes à quatre, il est possible que l'algorithme mélange en fait un peu ces quatre classes. Pour contrecarrer ce problème, une technique consiste à fixer $k > 4$ et d'obtenir une segmentation dans laquelle l'union des classes correspondant à la quatrième interprétation devrait représenter les chenaux.

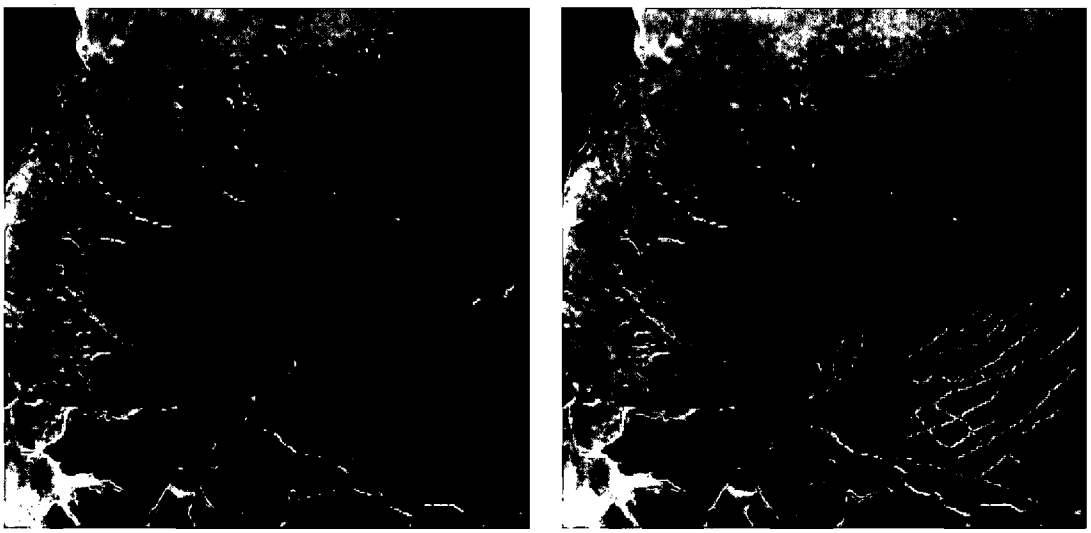
CHAPITRE 5

LES RÉSULTATS

Ce chapitre présente les images résultant des différents filtres de nettoyage ainsi que l'application des filtres de classification. Le résultat recherché est une image classifiée dans laquelle une ou l'union de plusieurs classes représentent fidèlement les plus grands chenaux qui, une fois identifiés et mesurés, serviront à estimer la population des plus petits chenaux à l'aide de la loi de puissance expliquée au chapitre 1. Tous les filtres de nettoyage ont été testés avec la même séquence de seize images du 21 février 2007 au 8 mars 2007 en utilisant l'image centrale du 28 février 2007 comme image de base. Cette image et une copie de celle-ci avec un sous-ensemble des chenaux recherchés mis en évidence manuellement sont illustrées à la figure 5.1.

Pour bien comparer les filtres de nettoyage, les paramètres de ceux-ci ont été gardés constants dans la mesure du possible. C'est-à-dire que tous les filtres spectraux ont des épreuves avec les mêmes fréquences de coupure et les mêmes bornes pour le rehaussement de contours. Quant aux paramètres des deux filtres de classification, ils sont toujours restés constants avec huit classes. Il est à noter parfois que deux ou plusieurs classes fusionnent lors de la segmentation avec l'algorithme des k-moyennes. Cette fusion n'est pas visible dans les résultats de ce chapitre mais elle est visible lors de l'utilisation du logiciel `ClusterViewer` (cf section 6.5), celui-ci présentant moins de classes qu'espéré.

Les résultats seront présentés à l'aide de figures contenant quatre images. La première image (a) en haut à gauche est toujours l'image de référence, la deuxième image (b) en haut à droite est l'image nettoyée, la troisième image (c) en bas à gauche est l'image classifiée et finalement la quatrième et dernière image (d) en bas à droite est l'image des classes représentant le mieux les chenaux. Les images résultats seront aussi mesurés à l'aide de la *mesure-f* (en anglais et dans ce texte, la *F-measure*) appliquée au quadrant inférieur droit de l'image. L'image classifiée sera



(a) (b)

FIG. 5.1 – L'image du 28 février 2007 en (a) et un sous-ensemble des chenaux recherchés en (b).

comparée à l'image vérité de la figure 5.1(b). Toutes les images résultats ainsi que le classement complet selon le critère de la *F-measure* peuvent être vus de manière électronique sur le site web `http://www-etud.iro.umontreal.ca/~leonard/`

Finalement, une discussion des meilleurs filtres ainsi que des commentaires sur le problème lui-même seront présentés.

5.1 Classification avec la *F-measure*

Il existe dans le domaine de la recherche de l'information une mesure permettant d'ordonner et de quantifier, pour une même requête, l'ensemble résultat de divers engins de recherche (ce travail est un engin de recherche recherchant les chenaux dans une image), il s'agit de la *F-measure* [22], une mesure composée de deux autres mesures aussi très présentes dans ce même domaine, la précision et le rappel.

5.1.1 La précision et le rappel

La précision et le rappel (respectivement *precision* et *recall* en anglais) sont deux mesures fréquemment utilisées en recherche de l'information et en classification statistique pour évaluer la qualité d'une classification [23].

La précision mesure l'exactitude et la fidélité des résultats et s'oppose au bruit, c'est-à-dire qu'elle mesure le quotient du nombre de documents pertinents trouvés divisé par le nombre total de documents trouvés. Ce quotient est défini à l'équation 5.1.

$$\text{précision} = \frac{|\{\text{documents pertinents}\} \cap \{\text{documents trouvés}\}|}{|\{\text{documents trouvés}\}|} \tag{5.1}$$

Le rappel quant à lui mesure la complétude des résultats et s'oppose au silence, c'est-à-dire qu'il mesure le quotient du nombre de documents pertinents trouvés divisé par le nombre total de documents pertinents. Ce quotient est défini à l'équation 5.2.

$$\text{rappel} = \frac{|\{\text{documents pertinents}\} \cap \{\text{documents trouvés}\}|}{|\{\text{documents pertinents}\}|} \tag{5.2}$$

Dans le cas spécifique des images résultats les équations de précision et de rappel peuvent être spécialisées tel qu'énoncé aux équations 5.3 et 5.4, les points mauves faisant référence aux pixels mauves de la figure 5.1(b) tandis que les points blancs font référence aux pixels blancs des diverses images résultats.

$$\text{précision} = \frac{|\{\text{points mauves}\} \cap \{\text{points blancs}\}|}{|\{\text{points blancs}\}|} \tag{5.3}$$

$$\text{rappel} = \frac{|\{\text{points mauves}\} \cap \{\text{points blancs}\}|}{|\{\text{points mauves}\}|} \tag{5.4}$$

L'opérateur `JAI precisionandrecall` de la bibliothèque `jai-operators` calcule la précison et le rappel entre deux images, l'une considérée comme image vérité et l'autre considérée comme image candidate.

F-measure	Algorithme	k-moyennes	Figure
0,154	identité	contextuel	figure 5.3
0,146	identité	original	figure 5.2

TAB. 5.1 – Classement du filtre identité selon la *F-measure*.

5.1.2 La *F-measure*

La *F-measure* est la moyenne harmonique de la précision et du rappel et est définie à l'équation 5.5.

$$\mathrm{F} = \frac{2 \cdot (\text{précision} \cdot \text{rappel})}{(\text{précision} + \text{rappel})} \tag{5.5}$$

5.2 Les filtres spatiaux

Cette section montre les résultats des filtres spatiaux, c'est-à-dire les filtres n'utilisant pas de transformée de Fourier.

5.2.1 Le filtre identité

Le filtre identité, expliqué à la section 4.1.1, ne nettoie pas l'image comme tel, il ne fait que la propager dans le pipeline de classification. Les résultats de nettoyage du filtre identité sont illustrés à figure 5.2 pour le filtre de classification des k-moyennes original et à la figure 5.3 pour le filtre de classification des k-moyennes contextuel.

La performance de ce filtre est piètre, des zones entières de banquise sont désignées comme étant des chenaux. Par contre l'existence de ce filtre est fondamentale puisque bien que maintenant les images d'entrée sont des images brutes dont le seul traitement a été de remplir les trous noirs, rien n'empêche un utilisateur d'utiliser d'autres traitements sur les images d'entrée.

Le tableau 5.1 indique le filtre de classification des k-moyennes contextuel apporte un léger gain en classification par rapport au filtre des k-moyennes original.

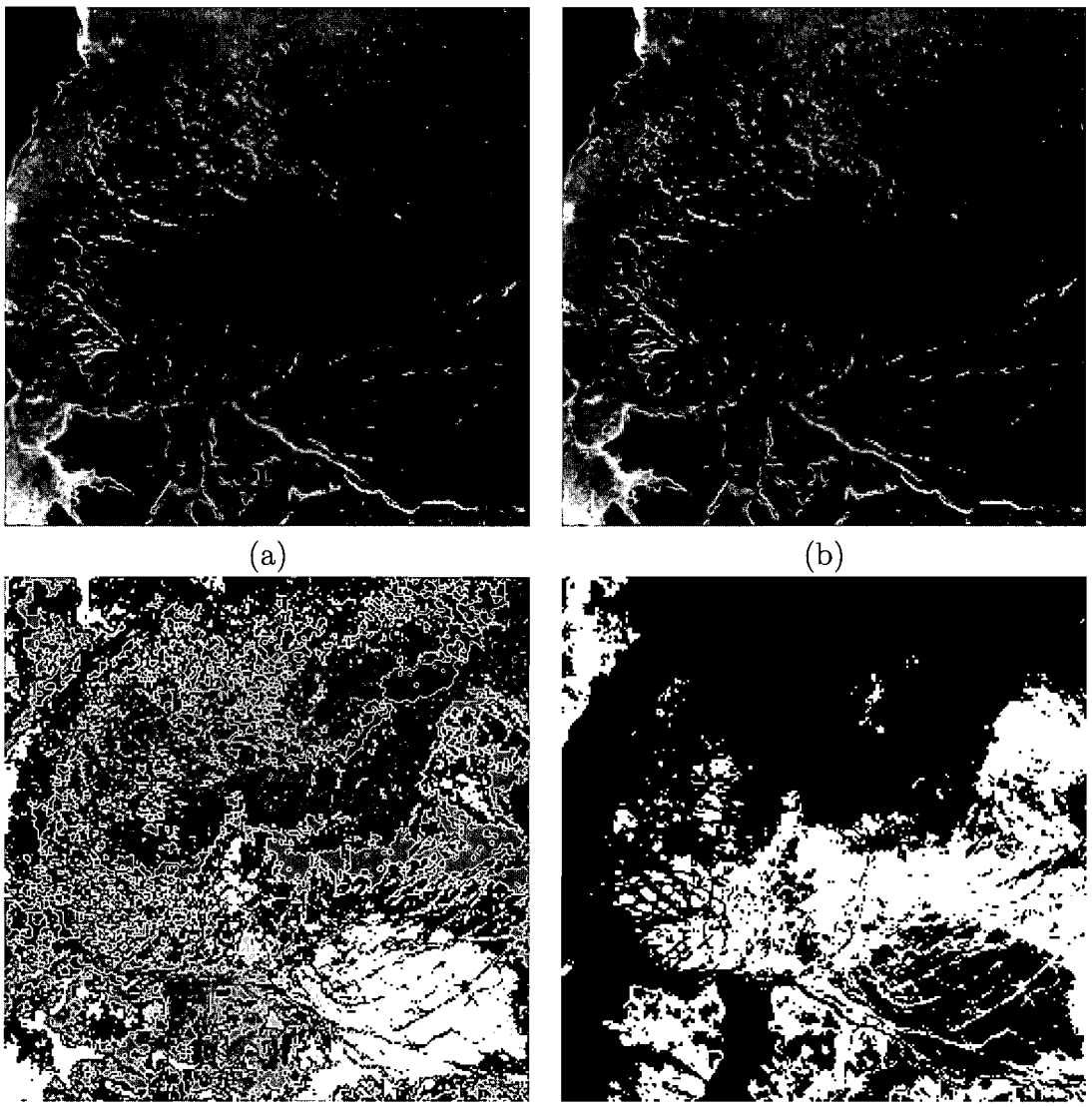


FIG. 5.2 – Le nettoyage du filtre identité en utilisant le filtre des k-moyennes original pour la reconnaissance.

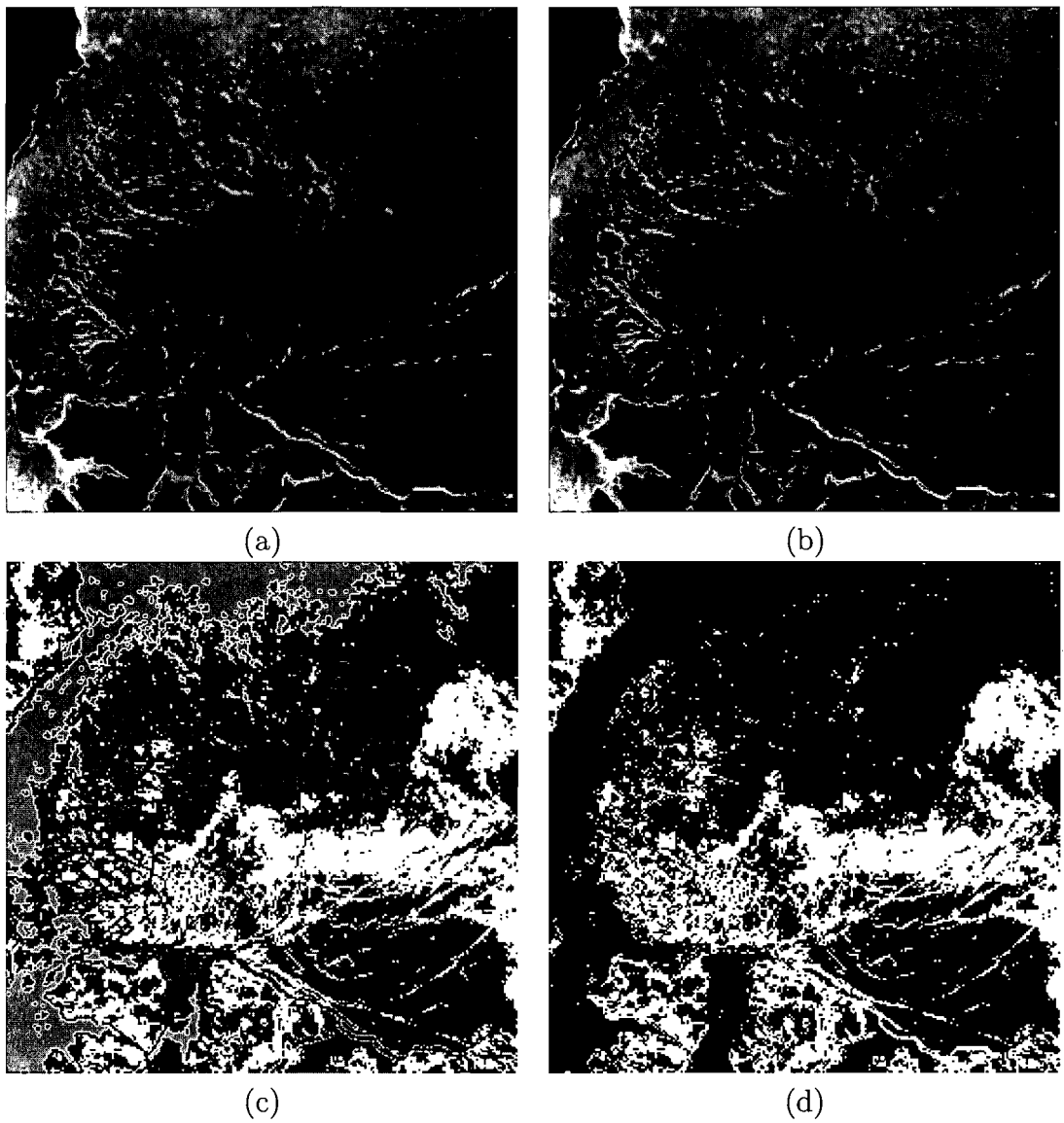
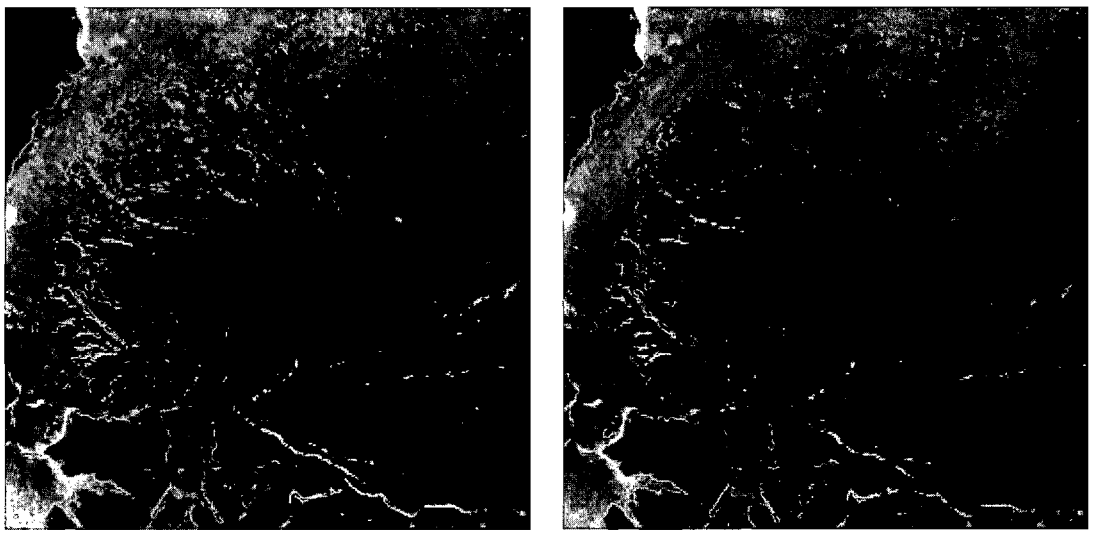


FIG. 5.3 – Le nettoyage du filtre identité en utilisant le filtre des k-moyennes contextuel pour la reconnaissance.



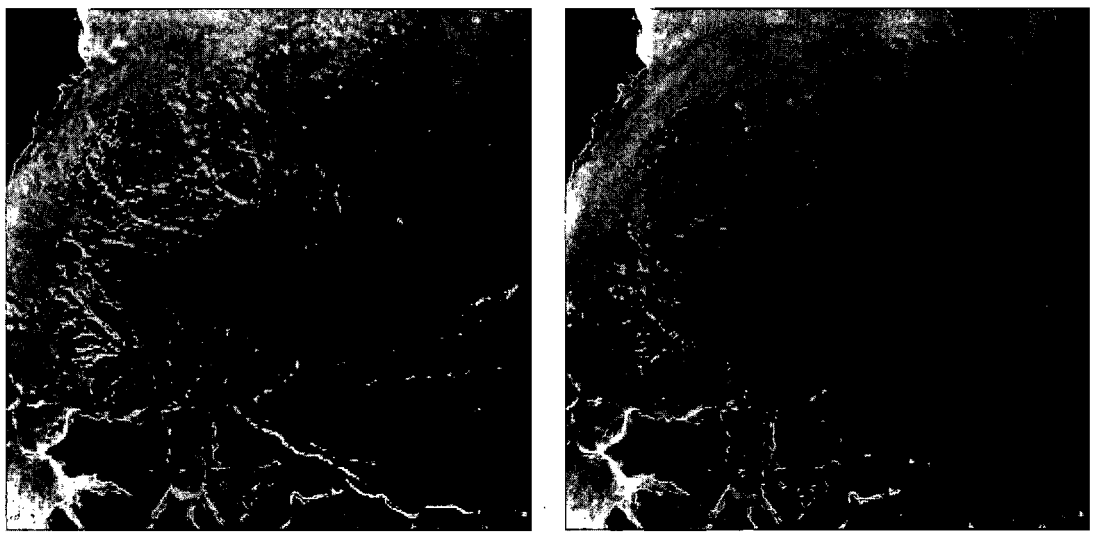
(a) (b)

FIG. 5.4 – Le nettoyage du filtre médian temporel avec trois voisins, un total de sept images.

5.2.2 Le filtre median temporel

Le filtre médian temporel, expliqué à la section 4.1.6, nettoie l'image en faisant une médiane de celle-ci et de ses voisines. La figure 5.4 illustre le résultat de l'application de ce filtre avec trois voisins, un total de sept images et la figure 5.5 illustre le résultat de l'application de ce filtre avec huit voisins, un total de 17 images.

Le filtre médian temporel semble enlever les nuages sans trop de problèmes, par contre il rend les chenaux flous et difficiles à reconnaître. Pour cette raison, aucune classification n'a été tentée. Sa performance peut être expliquée par la non-stationnarité des températures dans le temps ainsi que le mouvement des chenaux, ceux-ci variant de position et de forme aussi dans le temps. Bien que la performance de ce filtre dans le but de classifier les chenaux soit nulle, d'autres utilisations de ce filtre pourraient être faites.



(a) (b)

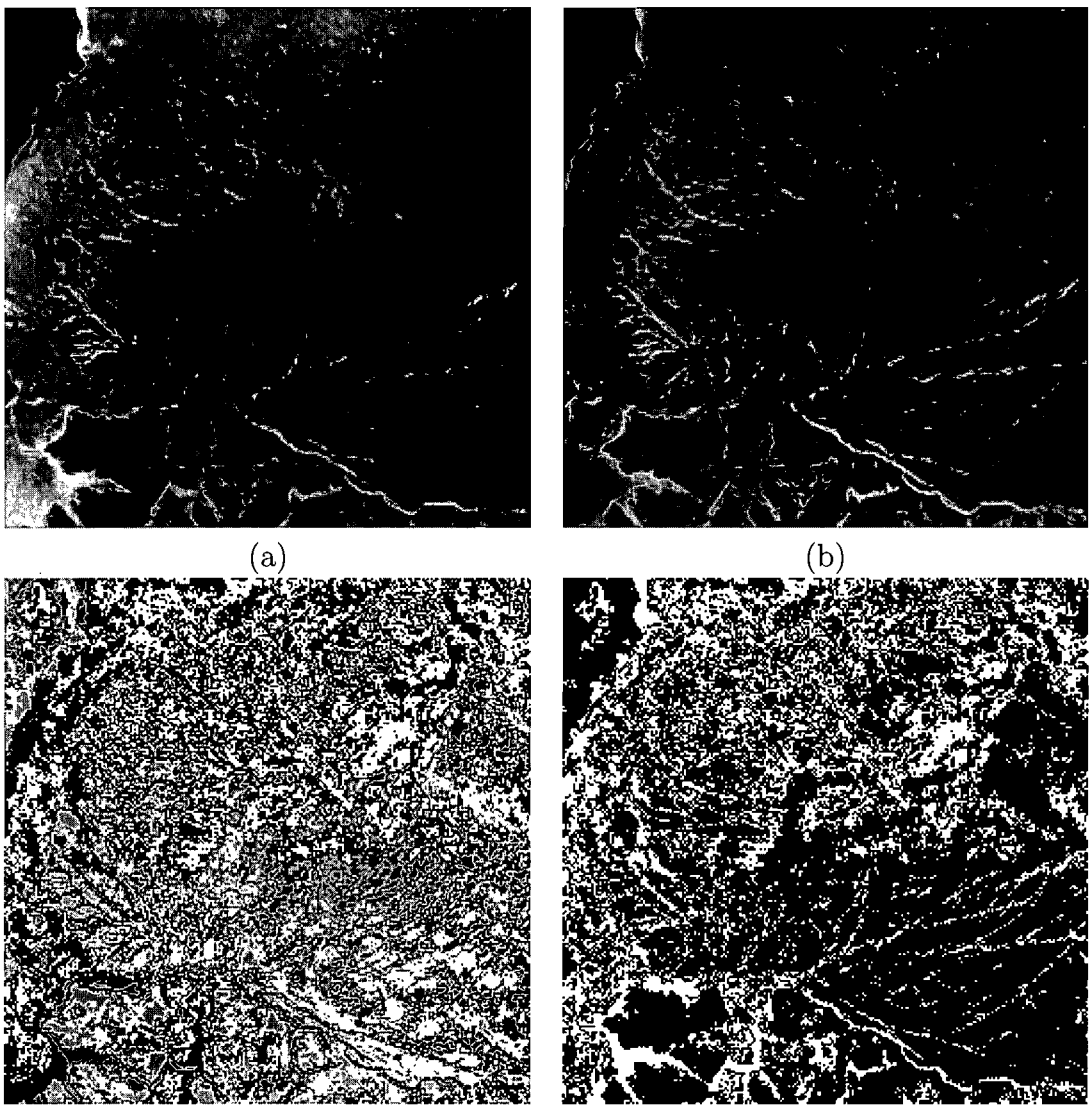
FIG. 5.5 – Le nettoyage du filtre médian temporel avec huit voisins, un total de 17 images.

5.2.3 Le filtre homomorphique spatial

Le filtre homomorphique spatial passe-haut, expliqué à la section 4.1.2, nettoie l'image en y soustrayant, dans le domaine logarithmique, une version floue d'elle-même.

Les figures 5.6 et 5.7 illustrent le résultat de l'application de ce filtre avec une boîte carrée de 11 pixels de côté. La première figure montre le classement de l'algorithme des k-moyennes original et la deuxième le classement avec la version contextuelle. Les figures 5.8 et 5.9 illustrent le résultat de l'application de ce filtre mais cette fois-ci avec une boîte carrée de 49 pixels de côté. La première figure montre le classement de l'algorithme des k-moyennes original et la deuxième le classement avec la version contextuelle. Dans les quatre cas, la pondération de l'image floue lors de la soustraction est de 80%.

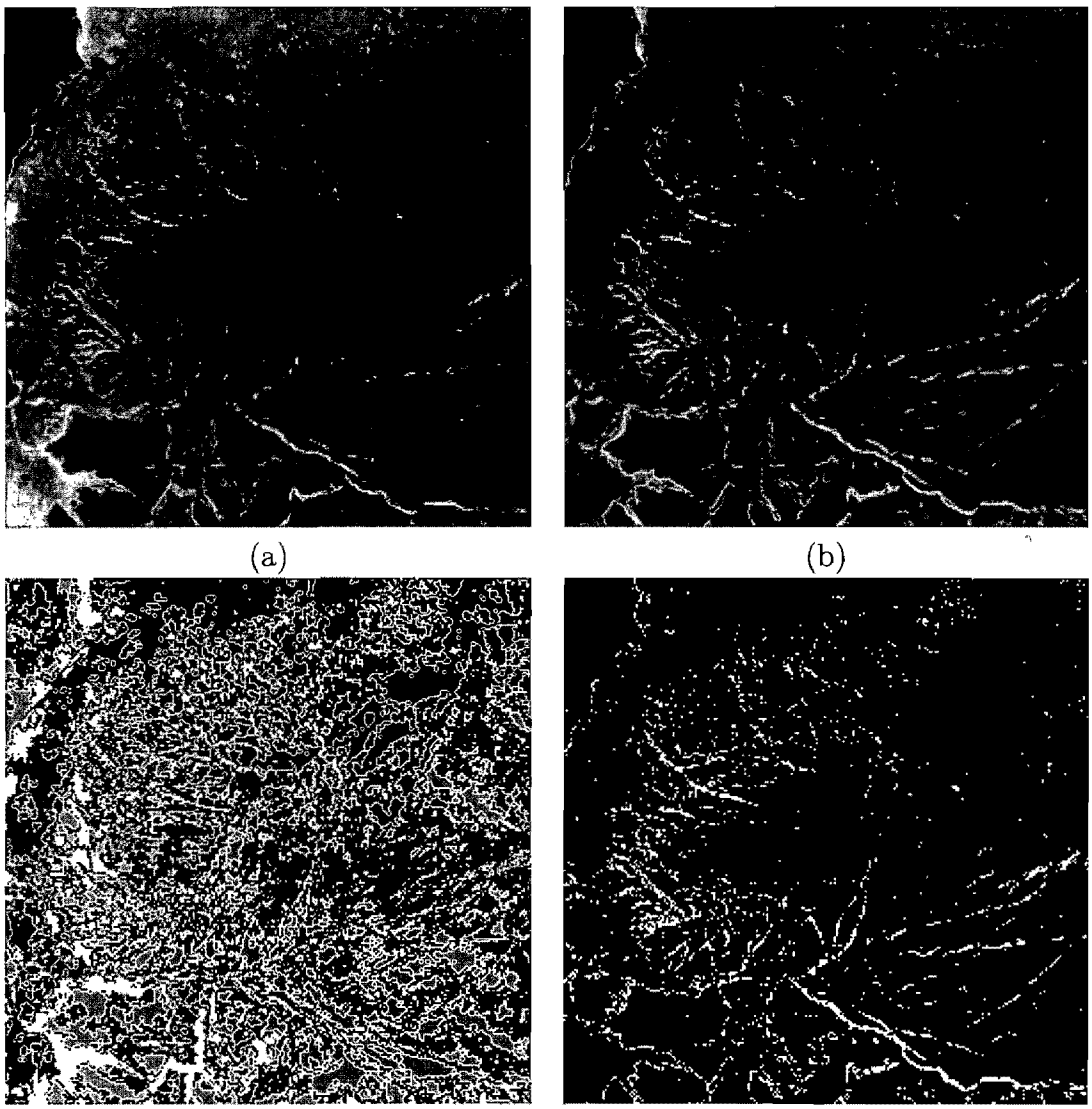
La performance de ce filtre est assez médiocre. Elle est meilleure certe que la performance du filtre identité, mais bien que des chenaux soient visibles, il y a aussi beaucoup de bruit, comme des sections de banquise reconnues comme étant



(a) (b)

(c) (d)

FIG. 5.6 – Le nettoyage du filtre spatial homomorphique avec une boîte carrée de 11 pixels et le classement avec la version originale du filtre des k-moyennes.



(a) (b)

(c) (d)

FIG. 5.7 – Le nettoyage du filtre spatial homomorphique avec une boîte carrée de 11 pixels et le classement avec la version contextuelle du filtre des k-moyennes.

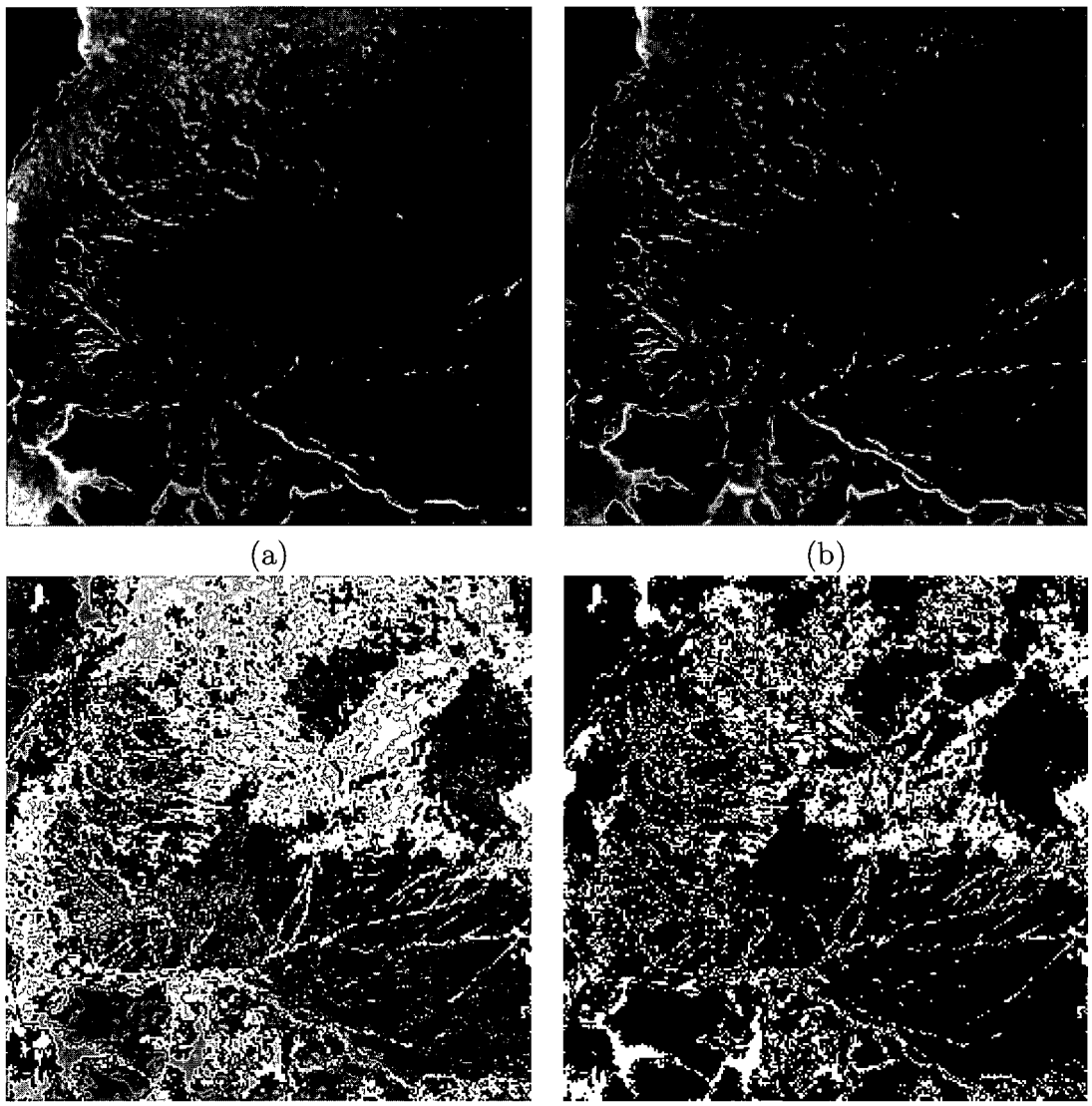
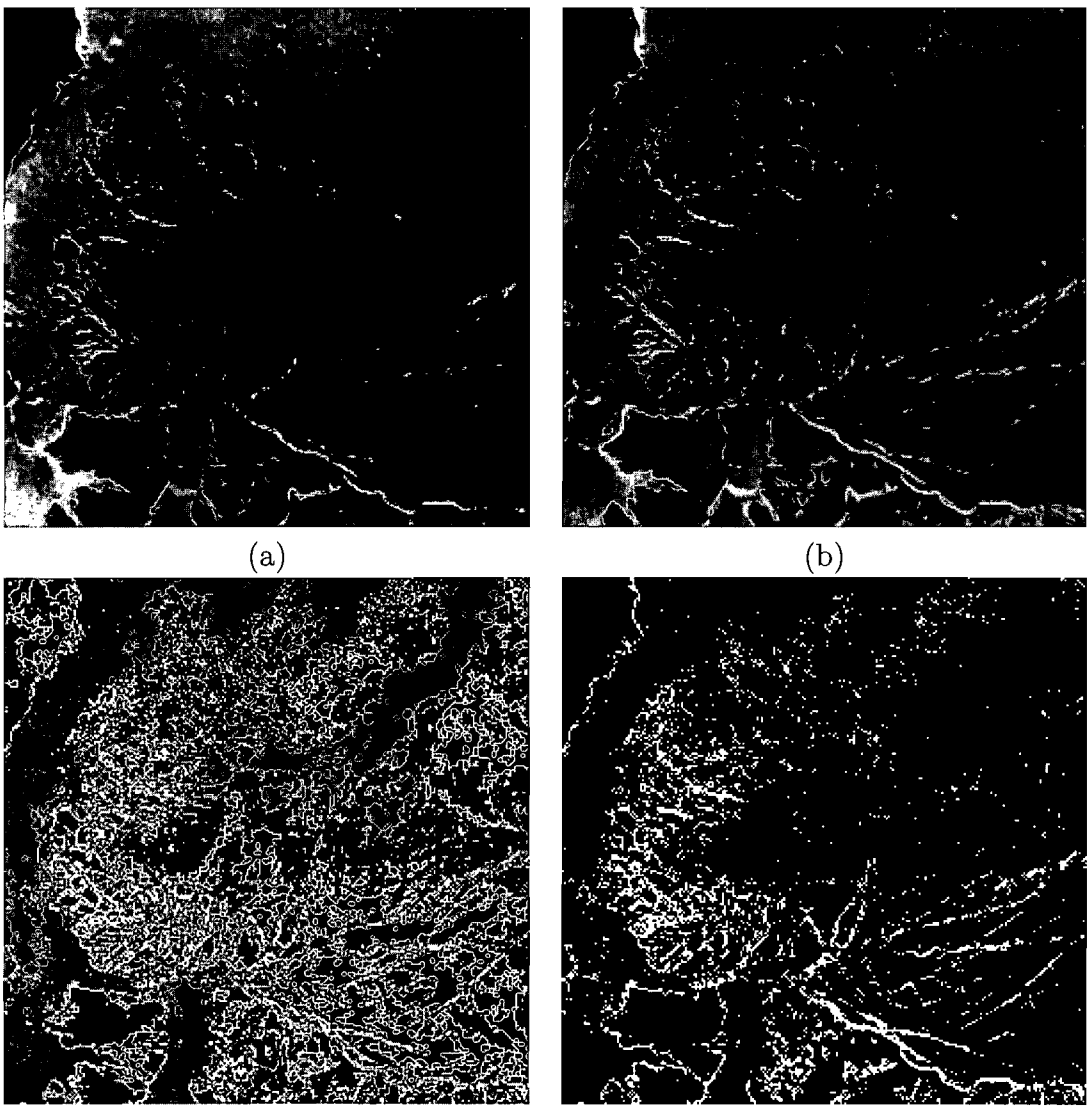


FIG. 5.8 – Le nettoyage du filtre spatial homomorphique avec une boîte carrée de 49 pixels et le classement avec la version originale du filtre des k-moyennes.



(a) (b)

(c) (d)

FIG. 5.9 – Le nettoyage du filtre spatial homomorphique avec une boîte carrée de 49 pixels et le classement avec la version contextuelle du filtre des k-moyennes.

F-measure	Algorithme	k-moyennes	Figure
0,237	passe-haut spatial	contextuel	figure 5.9
0,197	passe-haut spatial	contextuel	figure 5.7
0,175	passe-haut spatial	original	figure 5.6
0,135	passe-haut spatial	original	figure 5.8

TAB. 5.2 – Classement du filtre homomorphique passe-haut spatial selon la *F-measure*.

des chenaux.

Cependant, le classement du tableau 5.2 donne une assez bonne note à la version de la boîte de 49 pixels de côté tandis que les trois autres classifications se retrouvent toutes avec une valeur en bas de $0,2$.

Il est aussi à remarquer que plus la taille du filtre boîte est grande, plus l'image nettoyée ressemble à l'image de référence. En effet, les images 5.8(b) et 5.9(b) contiennent des traces beaucoup plus visibles des nuages en comparaison aux images 5.6(b) et 5.7(b).

L'image exemple de l'article décrivant ce filtre est une image urbaine provenant du satellite Terra. Le but du filtre dans [10] est d'enlever les nuages afin de rehausser les détails tout en produisant une image visiblement semblable à l'image initiale. D'ailleurs les images produites par ce filtre ressemblent à l'image de référence tout en ayant les chenaux plus visibles. Par contre, dans le cadre de ce travail, et ce sera encore plus visible avec les filtres spectraux, l'image nettoyée n'a pas à être similaire à l'image initiale. L'image nettoyée désirée est une image dans laquelle les chenaux sont très distinctifs, le reste de l'image n'étant pas important.

5.3 Les filtres spectraux

Cette section montre les résultats de nettoyage des filtres spectraux, c'est-à-dire les filtres utilisant la transformée de Fourier. Chacun des filtres a été testé avec deux jeux de paramètres, un premier qui ne retient que les basses fréquences et un deuxième qui retient les basses et les moyennes fréquences. Ces filtres sont

aussi tous des filtres rehausseur de contours tel que décrit au chapitre 4.

5.3.1 Filtre idéal

Le filtre spectral idéal, expliqué à la section 4.1.3, nettoie l'image en enlevant de manière inconditionnelle les basses fréquences.

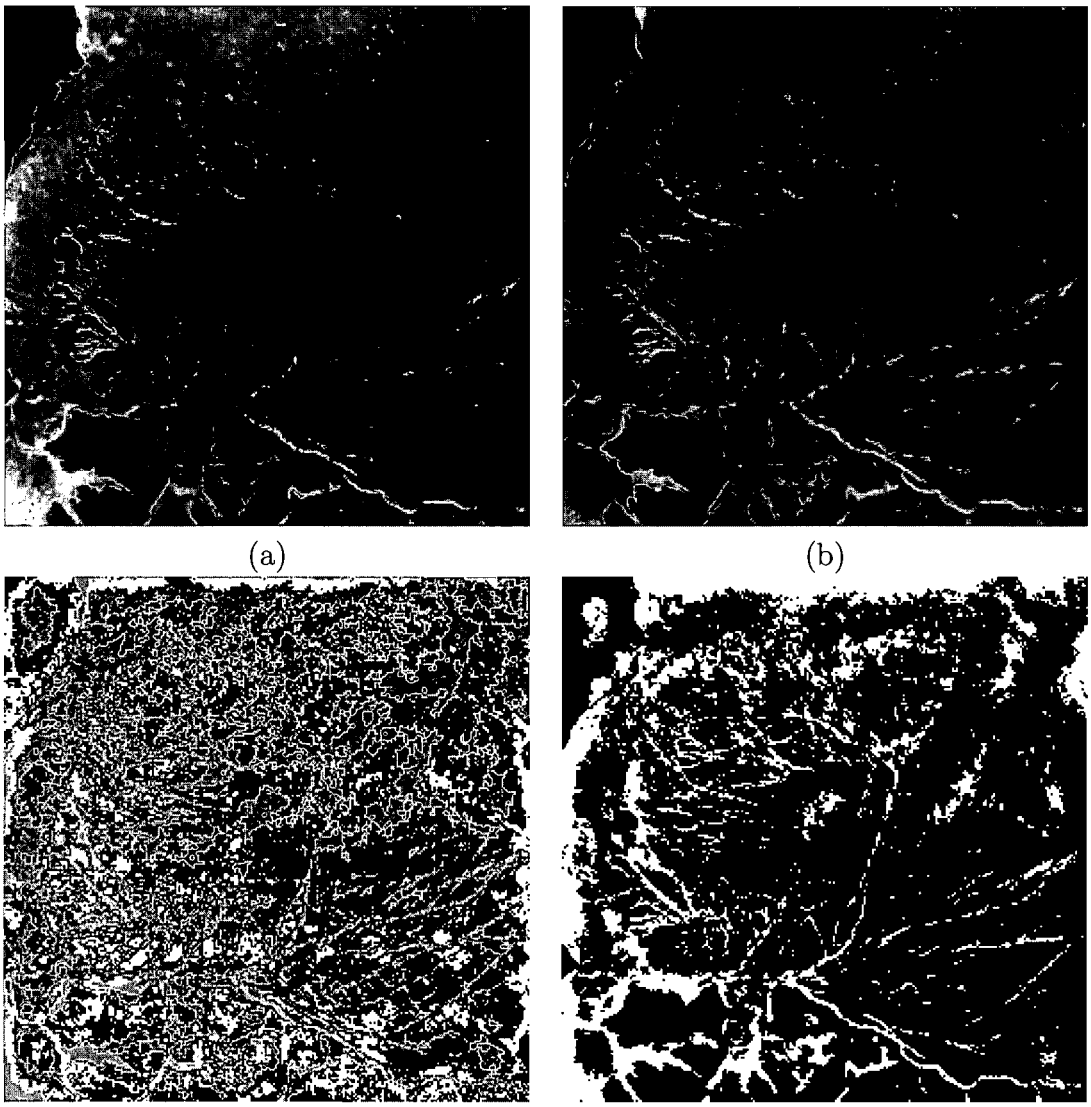
Les figures 5.10 et 5.11 illustrent le résultat de l'application de ce filtre avec une fréquence de coupure de 5%, une borne inférieure $\gamma_{min} = 0,13$ et une borne supérieure de $\gamma_{max} = 0,87$. La première figure présente la classification du filtre des k-moyennes original et la seconde présente la classification du filtre des k-moyennes contextuel.

Contrairement aux deux autres filtres spectraux, les figures 5.12 et 5.13 illustrent le résultat de l'application de ce filtre avec une fréquence de coupure de 32%, une borne inférieure $\gamma_{min} = 0,06$ et une borne supérieure de $\gamma_{max} = 0,94$. La première figure présente la classification du filtre des k-moyennes original et la seconde présente la classification du filtre des k-moyennes contextuel.

Les images nettoyées en n'enlevant que les basses fréquences ressemblent beaucoup aux images nettoyées du filtre spatial passe-haut. D'ailleurs, la classification de ces images est très similaire.

Par contre, les images nettoyées en ne laissant passer que les hautes fréquences sont beaucoup plus parlantes. Le phénomène des anneaux (*ringing* en anglais), propre au filtre idéal, y est très visible et s'amplifie quand la fréquence de coupure augmente et semble rendre l'image nettoyée totalement inutilisable. Par contre, ces images classifiées sont bien meilleures que toutes les images des filtres spatiaux, les chenaux principaux y étant très visibles. Enfin, même si l'image de classification de la version conventionnelle du filtre des k-moyennes semble très bruitée, une bonne partie des chenaux est visible.

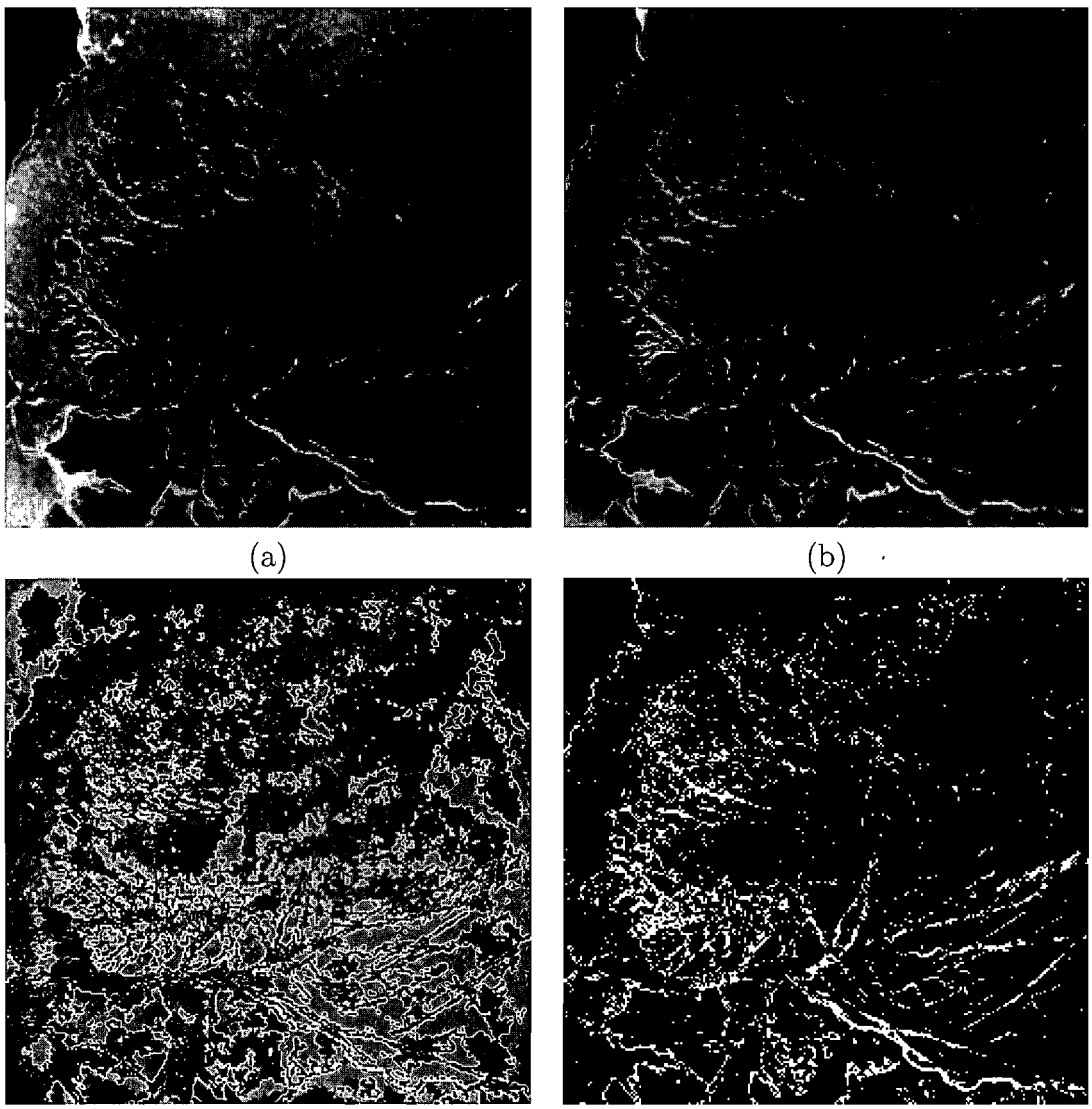
Les valeurs du tableau 5.3 corroborent ces observations en donnant d'assez bonnes notes à toutes les images, particulièrement à la figure 5.11.



(a) (b)

(c) (d)

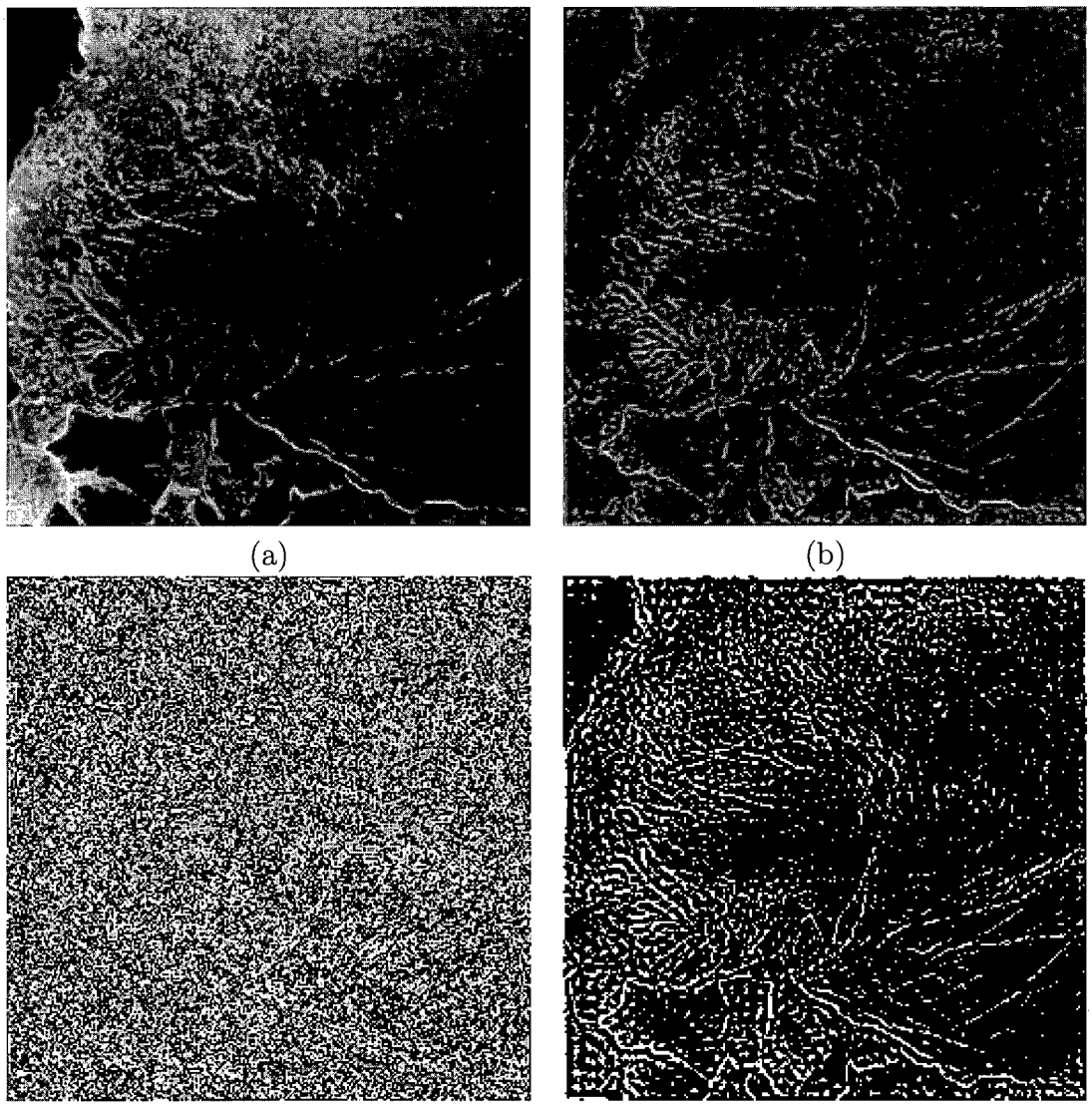
FIG. 5.10 – Le nettoyage du filtre spectral idéal 2D, avec la fréquence de coupure à 5%, $\gamma_{min} = 0,13$ et $\gamma_{max} = 0,87$ classifiée par le filtre des k-moyennes original.



(a) (b)

(c) (d)

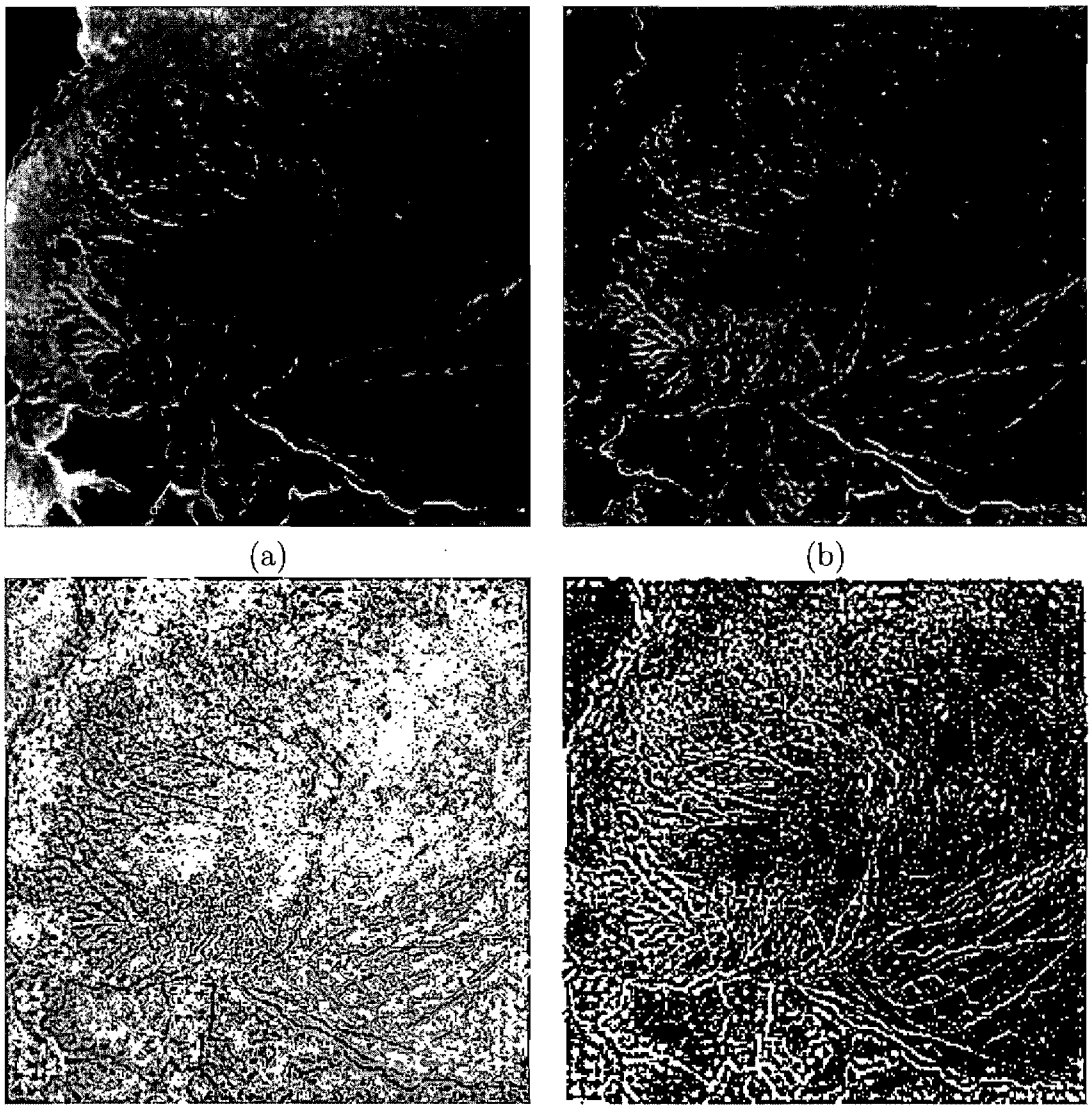
FIG. 5.11 – Le nettoyage du filtre spectral idéal 2D, avec la fréquence de coupure à 5%, $\gamma_{min} = 0,13$ et $\gamma_{max} = 0,87$ classifiée par le filtre des k-moyennes contextuel.



(a) (b)

(c) (d)

FIG. 5.12 – Le nettoyage du filtre spectral idéal 2D, avec la fréquence de coupure à 32%, $\gamma_{min} = 0,06$ et $\gamma_{max} = 0,94$ classifiée par le filtre des k-moyennes original.



(a) (b)

(c) (d)

FIG. 5.13 – Le nettoyage du filtre spectral idéal 2D, avec la fréquence de coupure à 32%, $\gamma_{min} = 0,06$ et $\gamma_{max} = 0,94$ classifiée par le filtre des k-moyennes contextuel.

F-measure	Algorithme	k-moyennes	Figure
0,249	idéal 2D	contextuel	figure 5.11
0,213	idéal 2D	contextuel	figure 5.13
0,202	idéal 2D	original	figure 5.10
0,195	idéal 2D	original	figure 5.12

TAB. 5.3 – Classement du filtre passe-haut idéal en deux dimensions selon la *F-measure*.

5.3.2 Filtre gaussien

Le filtre spectral gaussien, expliqué à la section 4.1.4, nettoie l'image en enlevant les basses fréquences en suivant une courbe gaussienne inversée.

Les figures 5.14 et 5.15 illustrent le résultat de l'application de ce filtre avec une fréquence de coupure de 5%, une borne inférieure $\gamma_{min} = 0,13$, une borne supérieure $\gamma_{max} = 0,87$ et une pente $c = 1$. La première figure présente la classification du filtre des k-moyennes original et la seconde présente la classification du filtre des k-moyennes contextuel.

Les figures 5.16 et 5.17 illustrent le résultat de l'application de ce filtre avec une fréquence de coupure de 60%, une borne inférieure $\gamma_{min} = 0,10$, une borne supérieure $\gamma_{max} = 0,90$ et une pente $c = 2$. La première figure présente la classification du filtre des k-moyennes original et la seconde présente la classification du filtre des k-moyennes contextuel.

Les images produites par ce filtre sont très similaires aux images produites par le filtre idéal, si ce n'est du phénomème de *ringing*. L'image produite par la coupure des basses fréquences produit une image similaire à l'image originale sans l'interférence des nuages tandis que l'image produite par la coupure des basses et moyennes fréquences produit une image avec des chenaux très distincts.

Comme le filtre idéal passe-haut, la meilleure classification a été faite avec le filtre qui ne retient que les basses fréquences. Le filtre lui-même dans son ensemble est un très bon candidat pour d'autres filtres de classification.

Les valeurs du tableau 5.4 corroborent ces observations en donnant d'assez

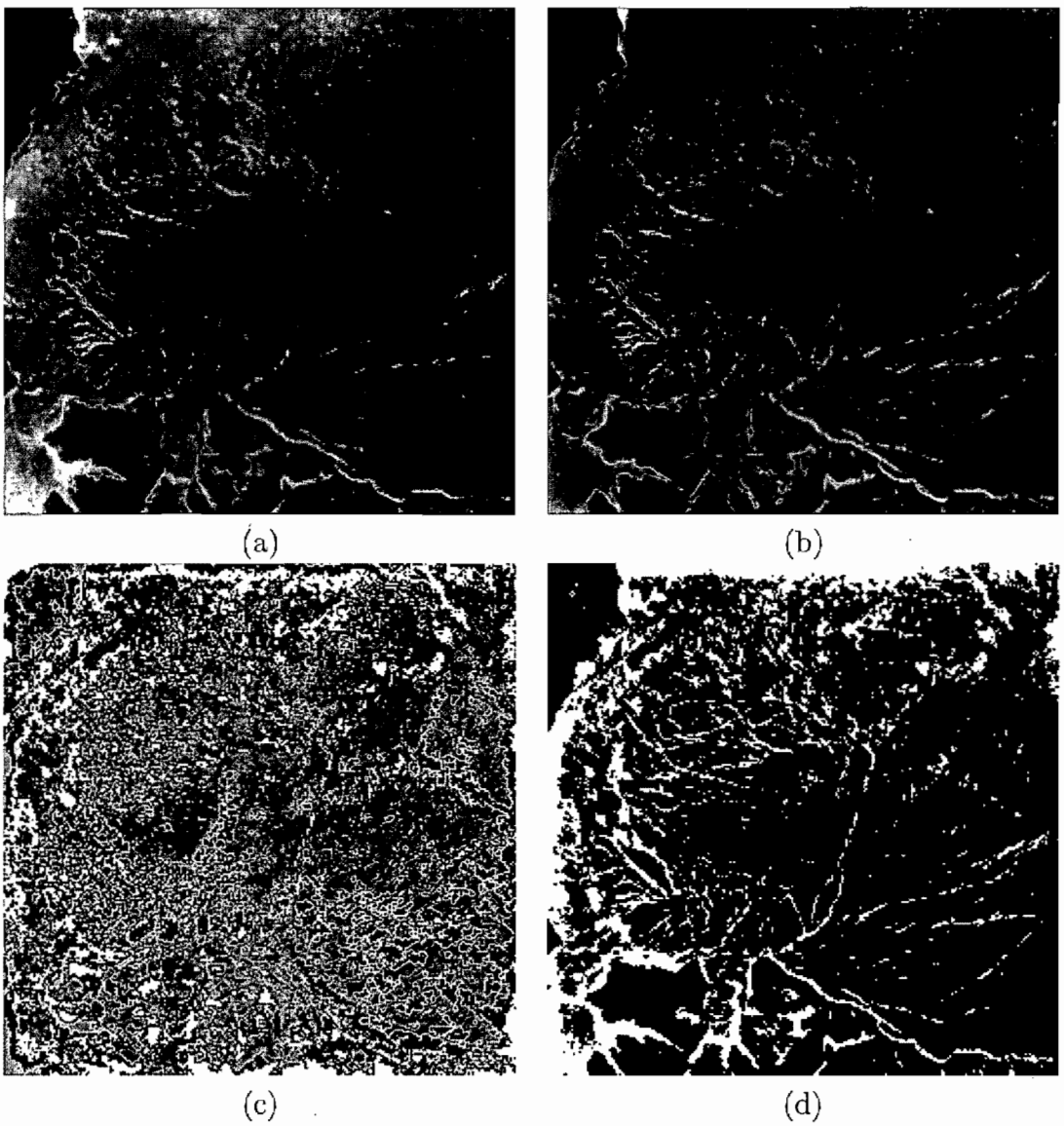
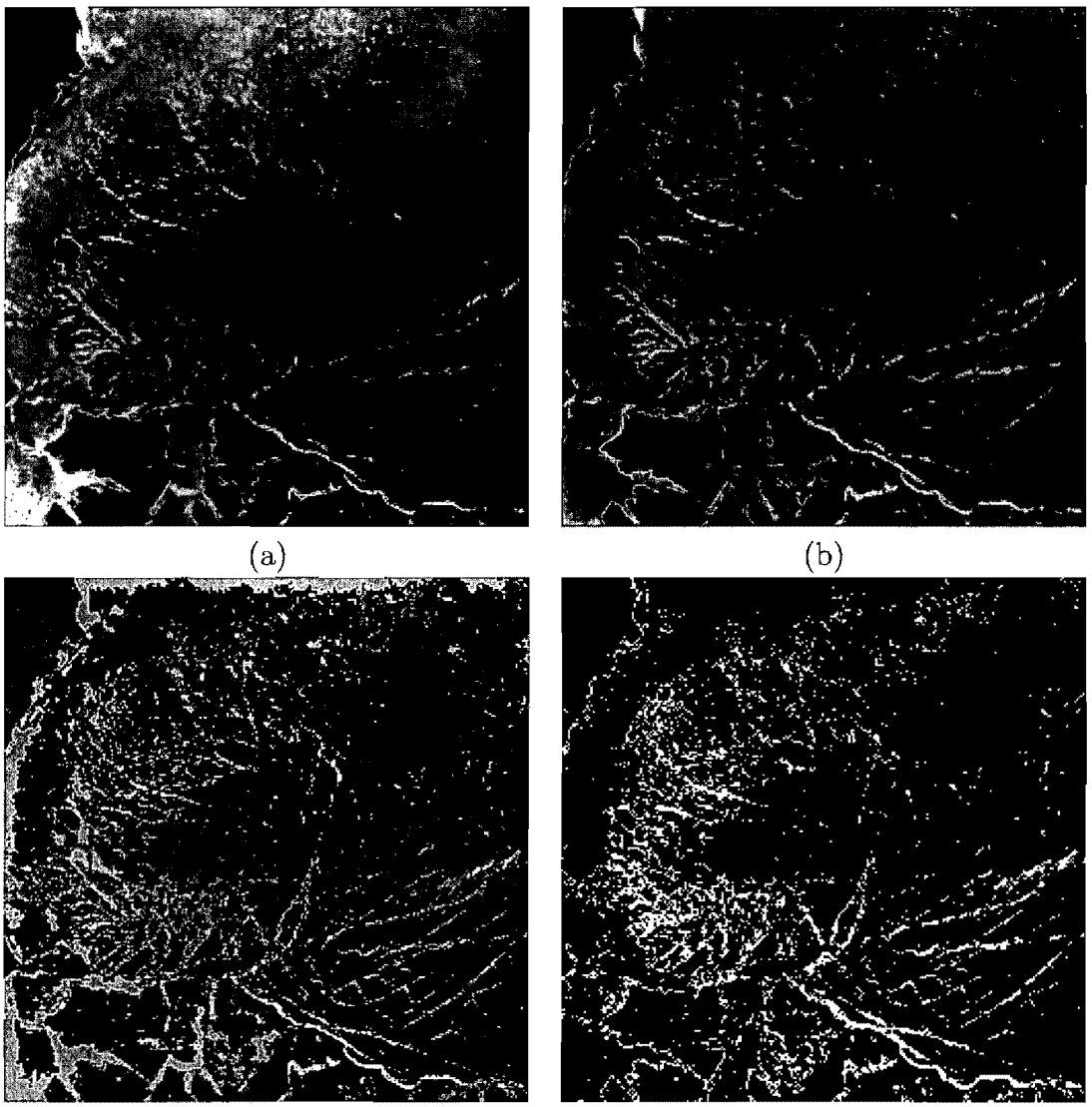


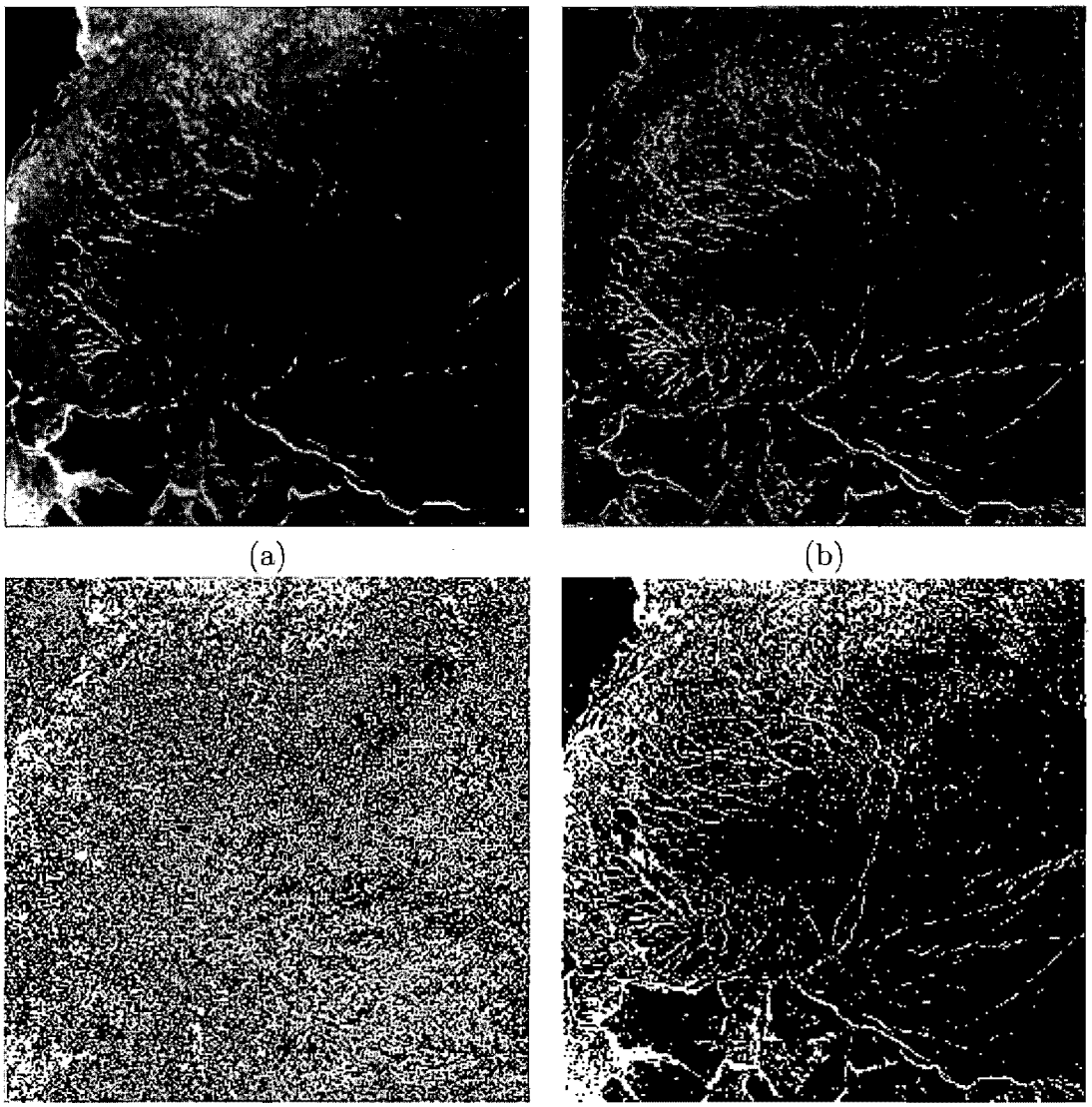
FIG. 5.14 – Le nettoyage du filtre spectral gaussien 2D, avec la fréquence de coupure à 5%, $\gamma_{min} = 0,13$, $\gamma_{max} = 0,87$ et une pente de 1, classifiée par le filtre des k-moyennes original.



(a) (b)

(c) (d)

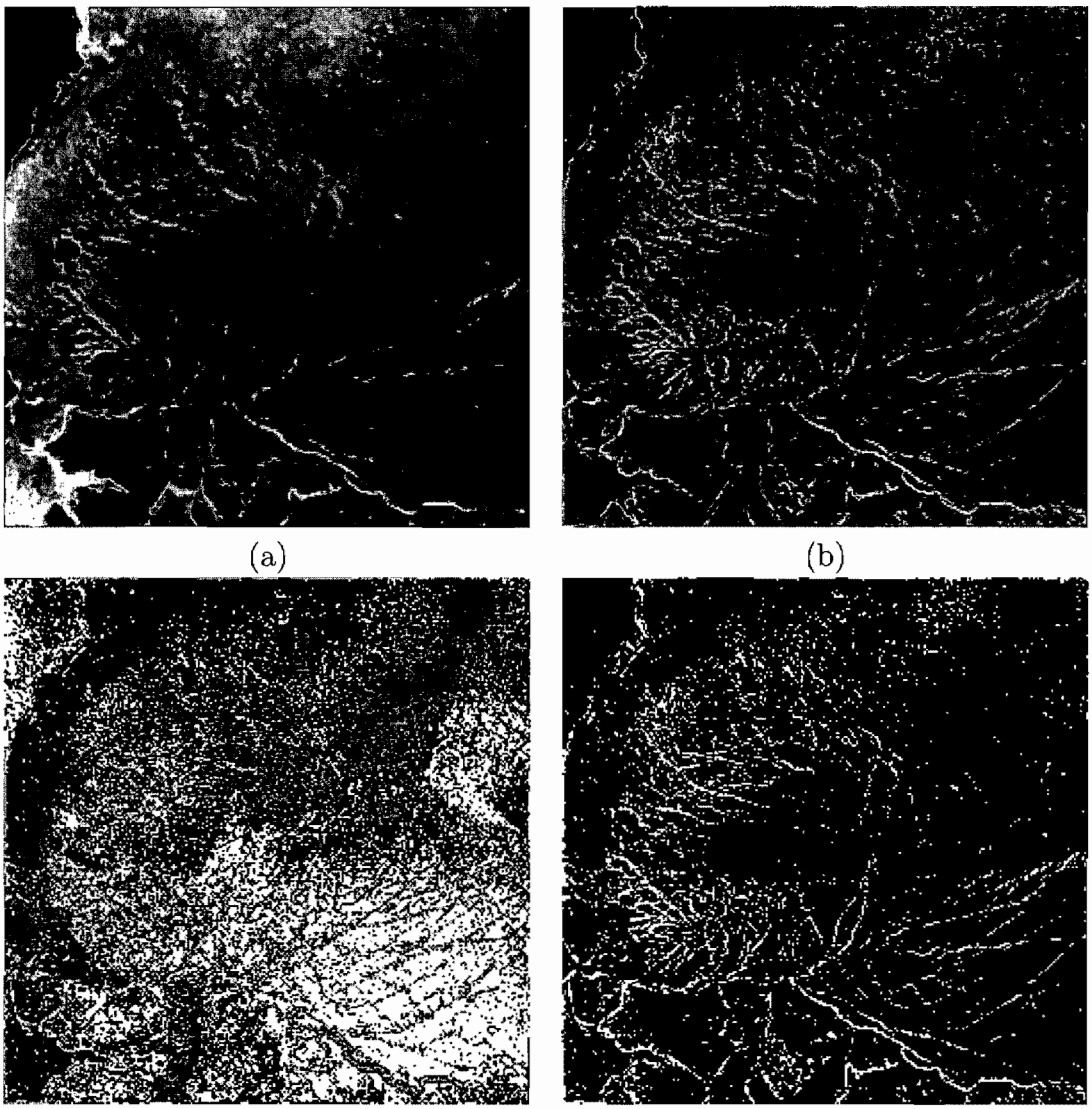
FIG. 5.15 – Le nettoyage du filtre spectral gaussien 2D, avec la fréquence de coupure à 5%, $\gamma_{min} = 0,13$, $\gamma_{max} = 0,87$ et une pente de 1, classifiée par le filtre des k-moyennes contextuel.



(a) (b)

(c) (d)

FIG. 5.16 – Le nettoyage du filtre spectral gaussien 2D, avec la fréquence de coupure à 60%, $\gamma_{min} = 0,10$, $\gamma_{max} = 0,90$ et une pente de 2, classifiée par le filtre des k-moyennes original.



(a) (b)

(c) (d)

FIG. 5.17 – Le nettoyage du filtre spectral gaussien 2D, avec la fréquence de coupure à 60%, $\gamma_{min} = 0,10$, $\gamma_{max} = 0,90$ et une pente de 2, classifiée par le filtre des k-moyennes contextuel.

F-measure	Algorithme	k-moyennes	Figure
0,257	gaussien 2D	contextuel	figure 5.15
0,213	gaussien 2D	contextuel	figure 5.17
0,192	gaussien 2D	original	figure 5.14
0,170	gaussien 2D	original	figure 5.16

TAB. 5.4 – Classement du filtre passe-haut gaussien en deux dimensions selon la *F-measure*.

bonnes notes à toutes les images, particulièrement à la figure 5.15.

5.3.3 Filtre de Butterworth

Le filtre spectral de Butterworth, expliqué à la section 4.1.5, nettoie l'image en enlevant les basses fréquences en suivant une courbe de Butterworth inversée.

Les figures 5.18 et 5.19 illustrent le résultat de l'application de ce filtre avec une fréquence de coupure de 5%, une borne inférieure $\gamma_{min} = 0,13$, une borne supérieure de $\gamma_{max} = 0,87$ et un ordre de 1. La première figure présente la classification du filtre des k-moyennes original et la seconde présente la classification du filtre des k-moyennes contextuel.

Les figures 5.20 et 5.21 illustrent le résultat de l'application de ce filtre avec une fréquence de coupure de 60%, une borne inférieure $\gamma_{min} = 0,10$, une borne supérieure de $\gamma_{max} = 0,90$ et un ordre de 2. La première figure présente la classification du filtre des k-moyennes original et la seconde présente la classification du filtre des k-moyennes contextuel.

Les images produites par ce filtre sont très similaires aux images produites par les deux autres filtres spectraux, les filtres idéaux et gaussiens. L'image produite par la coupure des basses fréquences produit une image similaire à l'image originale sans l'interférence des nuages tandis que l'image produite par la coupure des basses et moyennes fréquences produit une image avec des chenaux très distincts.

Comme le filtre idéal passe-haut et le filtre gaussien, la meilleure classification a été faite avec le filtre qui ne retient que les basses fréquences. Le filtre lui-même

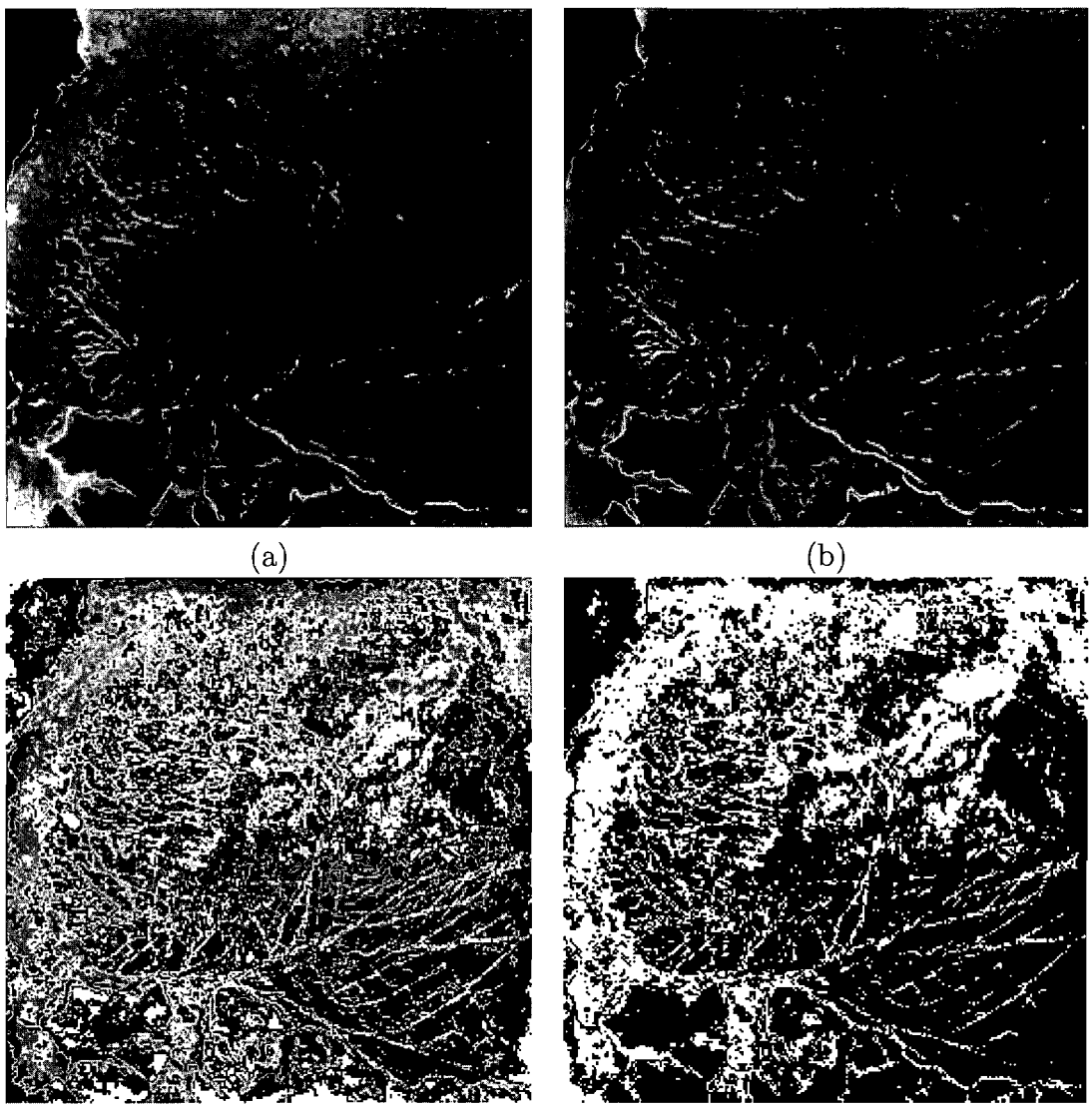
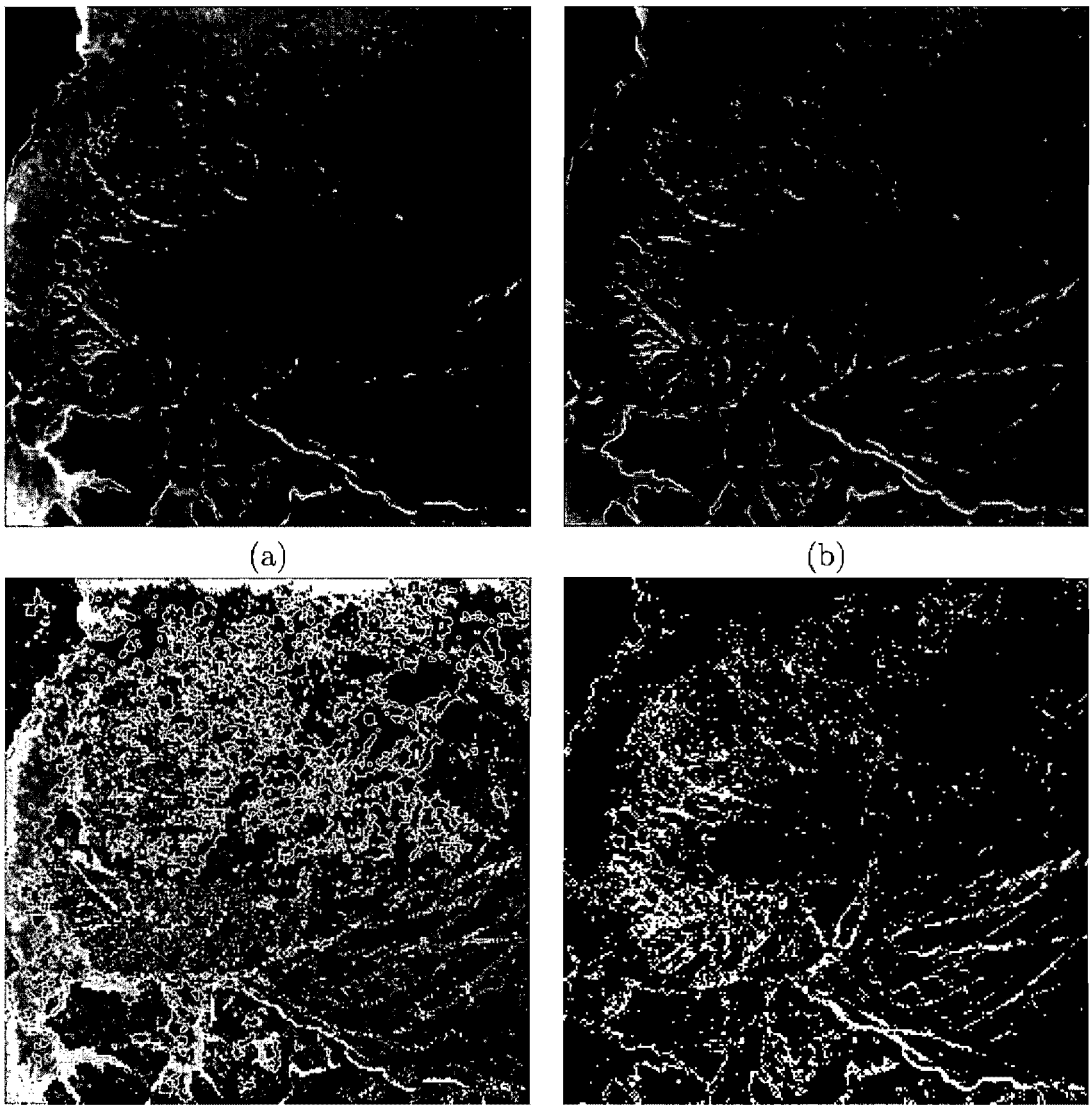


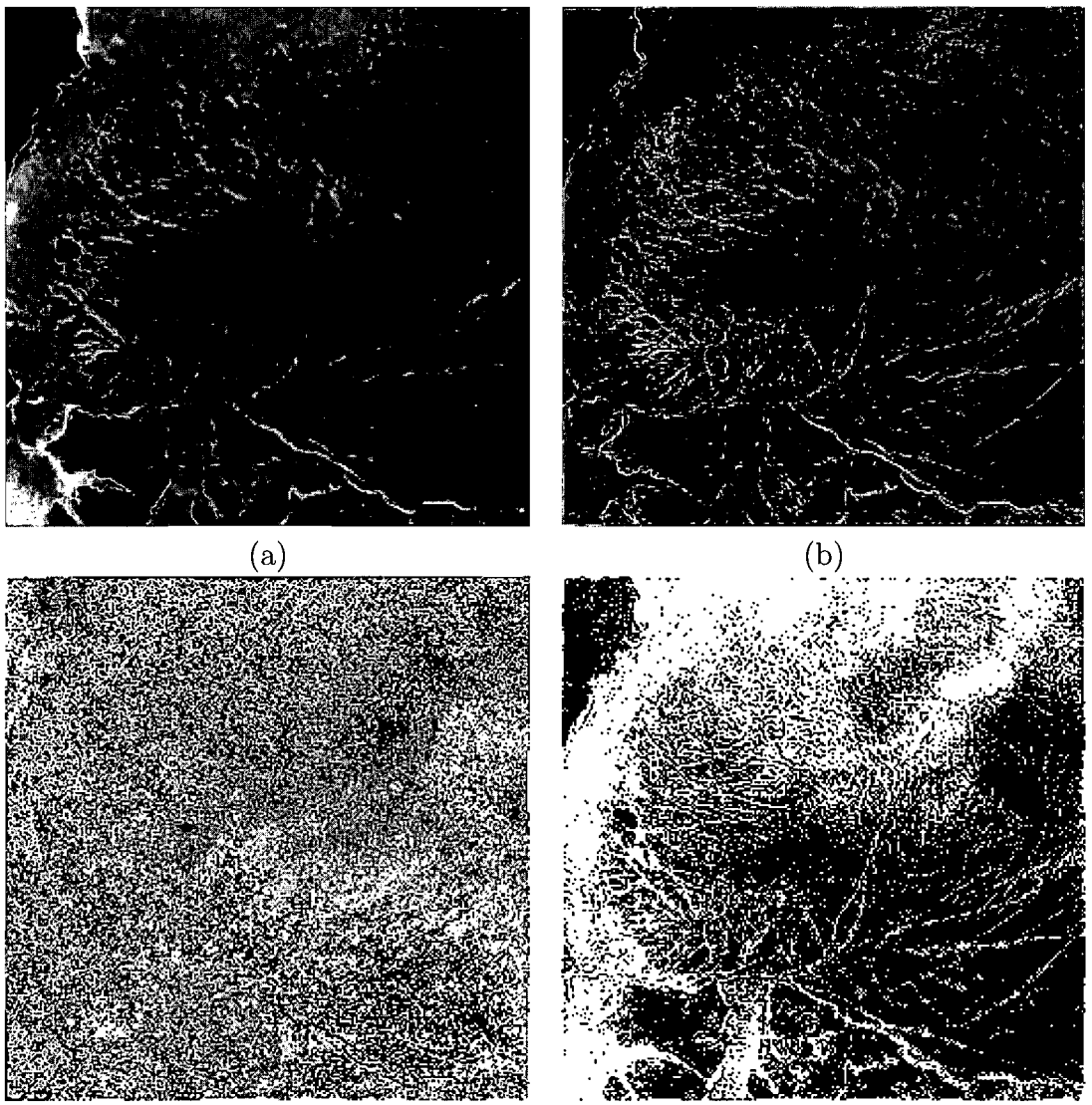
FIG. 5.18 – Le nettoyage du filtre spectral de Butterworth 2D, avec la fréquence de coupure à 5%, $\gamma_{min} = 0,13$, $\gamma_{max} = 0,87$ et un ordre de 1, classifiée par le filtre des k-moyennes original.



(a) (b)

(c) (d)

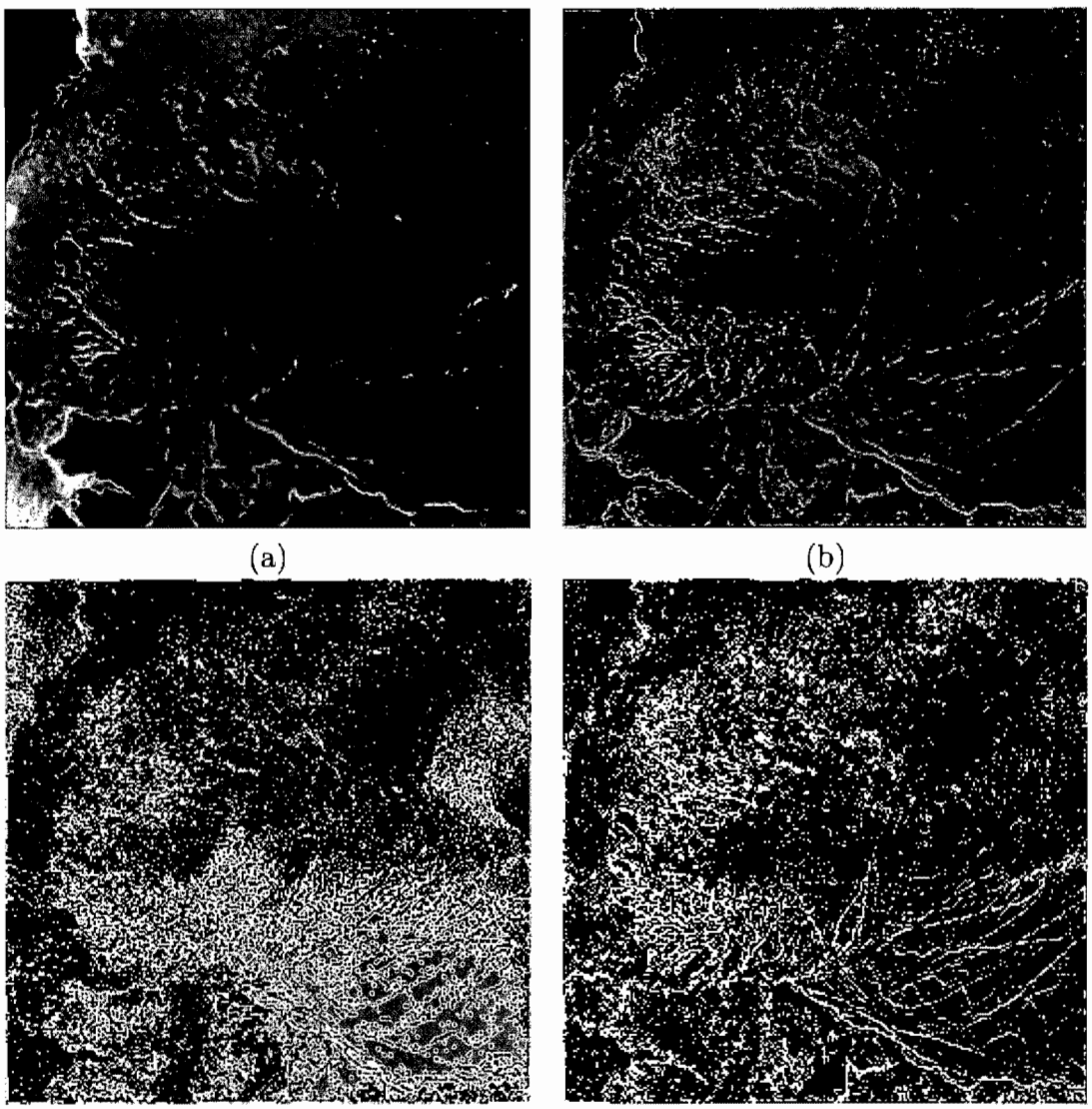
FIG. 5.19 – Le nettoyage du filtre spectral de Butterworth 2D, avec la fréquence de coupure à 5%, $\gamma_{min} = 0,13$, $\gamma_{max} = 0,87$ et un ordre de 1, classifiée par le filtre des k-moyennes contextuel.



(a) (b)

(c) (d)

FIG. 5.20 – Le nettoyage du filtre spectral de Butterworth 2D, avec la fréquence de coupure à 60%, $\gamma_{min} = 0,10$, $\gamma_{max} = 0,90$ et un ordre de 2, classifiée par le filtre des k-moyennes original.



(a) (b)

(c) (d)

FIG. 5.21 – Le nettoyage du filtre spectral de Butterworth 2D, avec la fréquence de coupure à 60%, $\gamma_{min} = 0,10$, $\gamma_{max} = 0,90$ et un ordre de 2, classifiée par le filtre des k-moyennes contextuel.

F-measure	Algorithme	k-moyennes	Figure
0,254	butterworth 2D	contextuel	figure 5.19
0,235	butterworth 2D	original	figure 5.18
0,221	butterworth 2D	contextuel	figure 5.21
0,190	butterworth 2D	original	figure 5.20

TAB. 5.5 – Classement du filtre passe-haut de Butterworth en deux dimensions selon la *F-measure*.

dans son ensemble est un très bon candidat pour d'autres filtres de classification.

Les valeurs du tableau 5.5 corroborent ces observations en donnant d'assez bonnes notes à toutes les images, particulièrement à la figure 5.19.

5.3.4 Filtre idéal 3D

Le filtre spectral idéal 3D, expliqué à la section 4.1.3.2, nettoie l'image en enlevant de façon inconditionnelle les basses fréquences tant au niveau spatial qu'au niveau temporel.

Les figures 5.22 et 5.23 illustrent le résultat de l'application de ce filtre sur une séquence de quatre images avec l'image de référence en troisième position, c'est-à-dire que deux voisins précédents ainsi qu'un voisin suivant sont utilisés. La fréquence de coupure au niveau spatial est de 30% et la fréquence de coupure au niveau temporel est de 75%. Les bornes du rehaussement de coutours sont $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$. La première figure présente la classification du filtre des k-moyennes original et la seconde présente la classification du filtre des k-moyennes contextuel.

Les figures 5.24 et 5.25 illustrent le résultat de l'application de ce filtre sur une séquence de seize images avec l'image de référence en huitième position, c'est-à-dire que la séquence entière est utilisée. La fréquence de coupure au niveau spatial est de 30% et la fréquence de coupure au niveau temporel est de 75%. Les bornes du rehaussement de coutours sont $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$. La première figure présente la classification du filtre des k-moyennes original et la seconde présente la

F-measure	Algorithme	k-moyennes	Figure
0,245	idéal 3D	contextuel	figure 5.23
0,213	idéal 3D	contextuel	figure 5.25
0,209	idéal 3D	original	figure 5.24
0,205	idéal 3D	original	figure 5.23

TAB. 5.6 – Classement du filtre passe-haut idéal en trois dimensions selon la *F-measure*.

classification du filtre des k-moyennes contextuel.

Le choix de la fréquence de coupure spatiale de 30% a été fait puisque une fréquences retenant les basses et les moyennes fréquences donnait de bons résultats pour les filtres spectraux classiques.

La performance de ce filtre spectral en trois dimensions est assez spéciale puisque les deux images nettoyées, les images 5.22(b) et 5.24(b), sont très similaires malgré la grande différence d'images dans la séquence. De plus, les images nettoyées sont similaires à celle de la version 2D du même filtre et il en va de même pour leur classification avec les deux filtres des k-moyennes, les valeurs de la *F-measure* présentées au tableau 5.6 étant similaires.

5.3.5 Filtre gaussien 3D

Le filtre spectral gaussien 3D, expliqué à la section 4.1.4.2, nettoie l'image en enlevant les basses fréquences tant au niveau spatial qu'au niveau temporel selon une bulle gaussienne inversée.

Les figures 5.26 et 5.27 illustrent le résultat de l'application de ce filtre sur une séquence de quatre images avec l'image de référence en troisième position, c'est-à-dire que deux voisins précédents ainsi qu'un voisin suivant sont utilisés. La fréquence de coupure au niveau spatial est de 30%, la fréquence de coupure au niveau temporel est de 75% et la pente est $c = 2$. Les bornes du rehaussement de coutours sont $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$. La première figure présente la classification du filtre des k-moyennes original et la seconde présente la classification

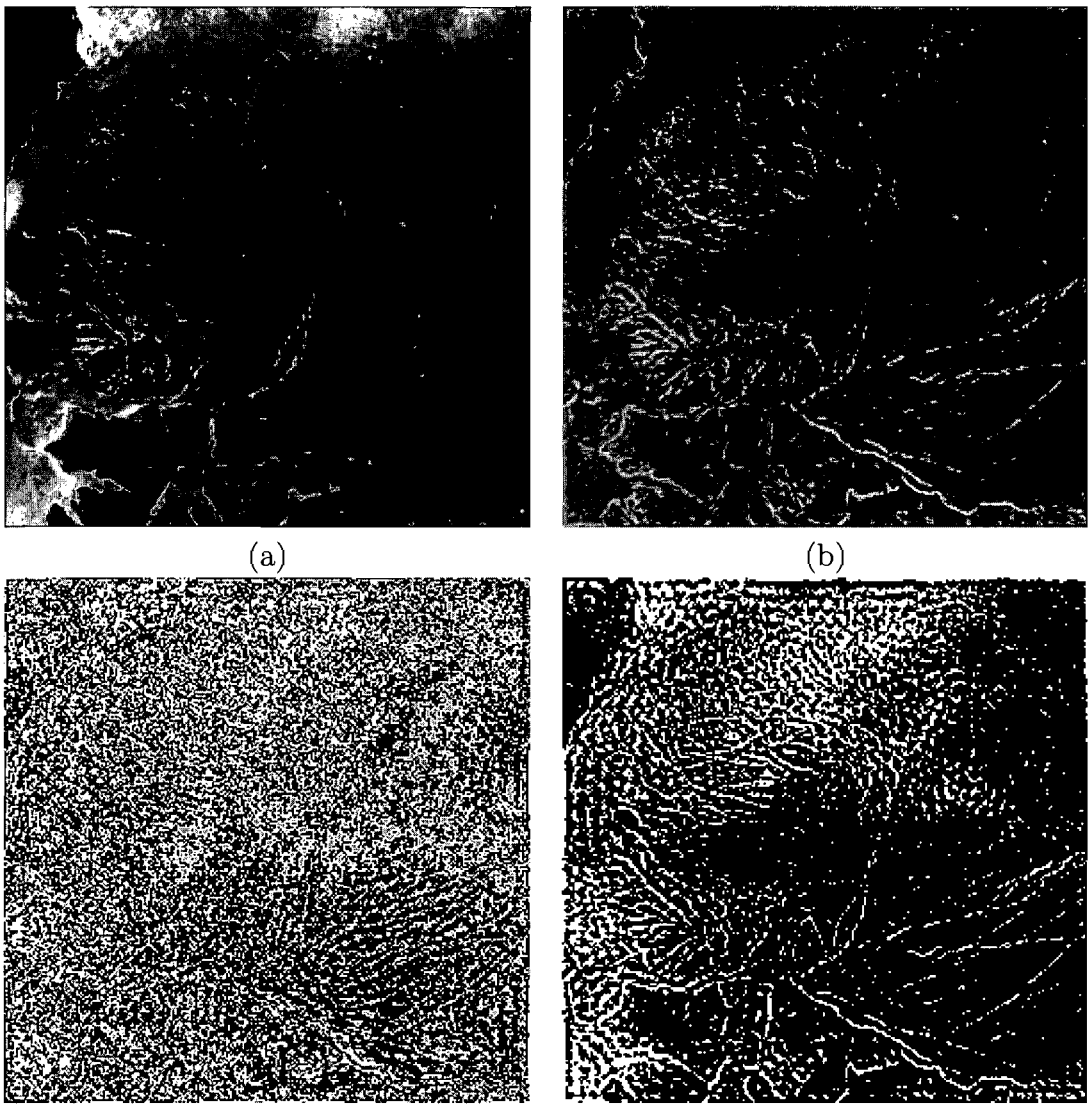
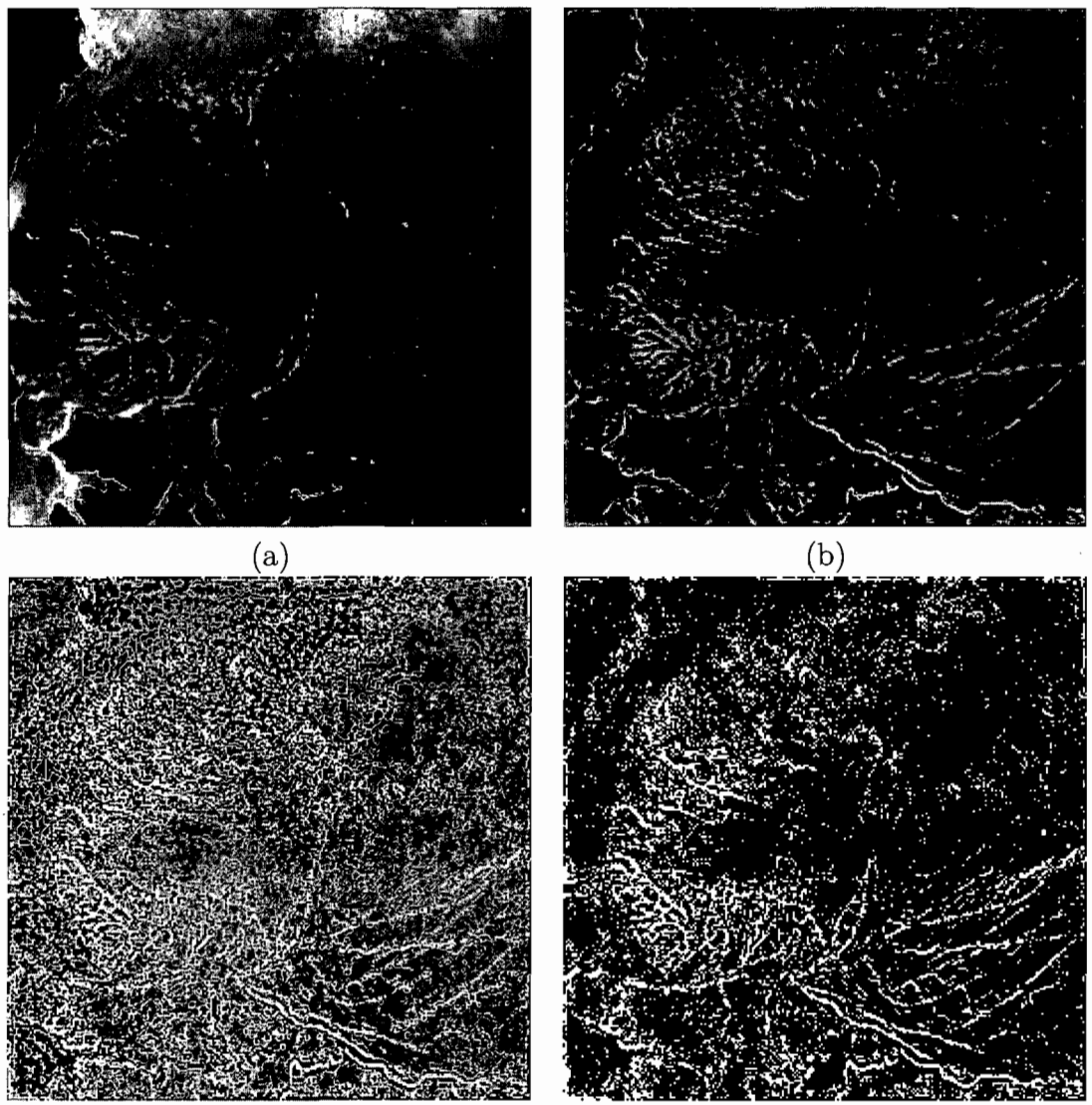


FIG. 5.22 – Le nettoyage du filtre spectral idéal 3D, avec quatre images, une fréquence de coupure spatiale à 30%, une fréquence de coupure temporelle à 75%, $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$, classifiée par le filtre des k-moyennes original.



(a) (b)

(c) (d)

FIG. 5.23 – Le nettoyage du filtre spectral idéal 3D, avec quatre images, une fréquence de coupure spatiale à 30%, une fréquence de coupure temporelle à 75%, $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$, classifiée par le filtre des k-moyennes contextuel.

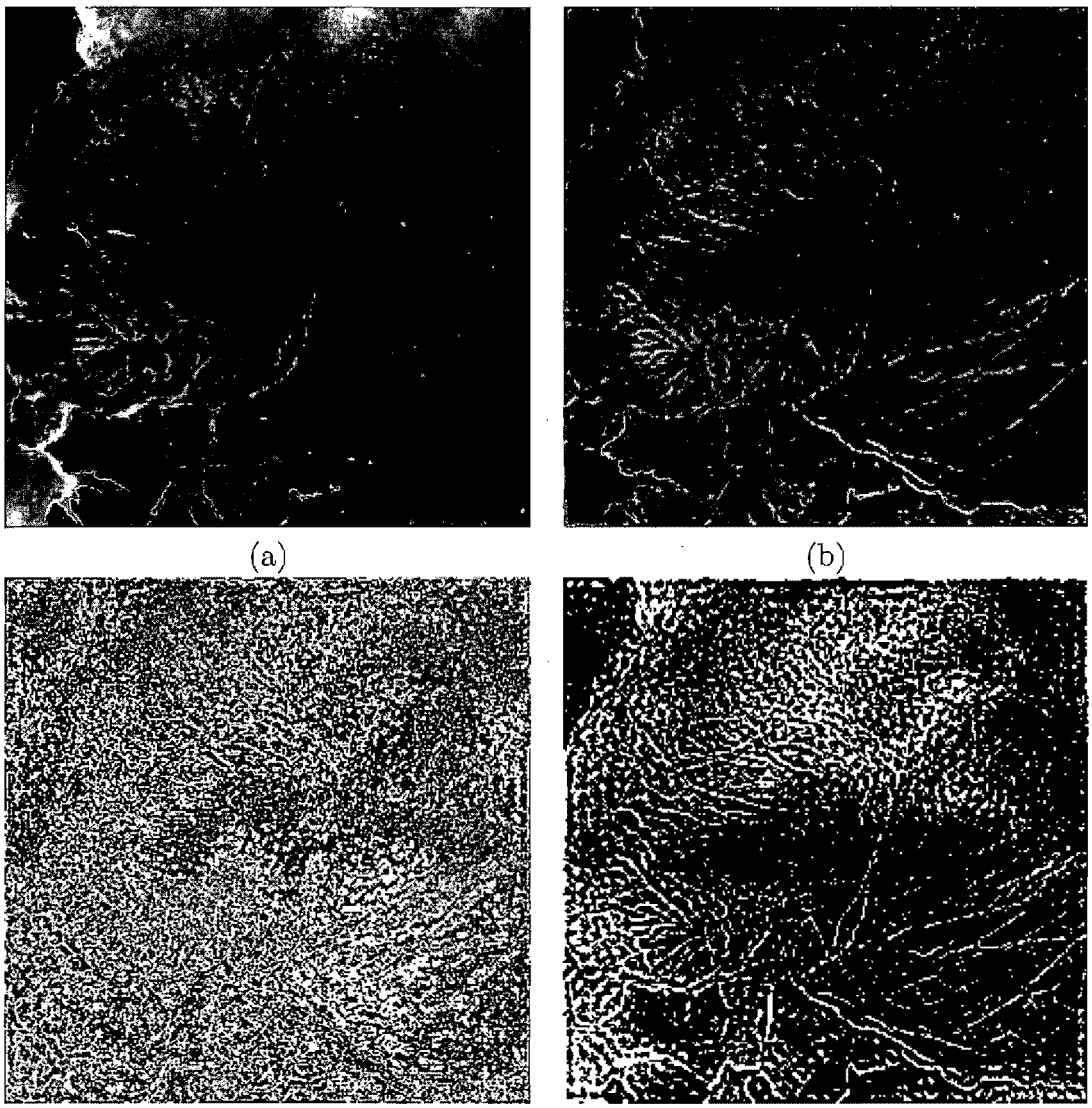


FIG. 5.24 – Le nettoyage du filtre spectral idéal 3D, avec seize images, une fréquence de coupure spatiale à 30%, une fréquence de coupure temporelle à 75%, $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$, classifiée par le filtre des k-moyennes original.

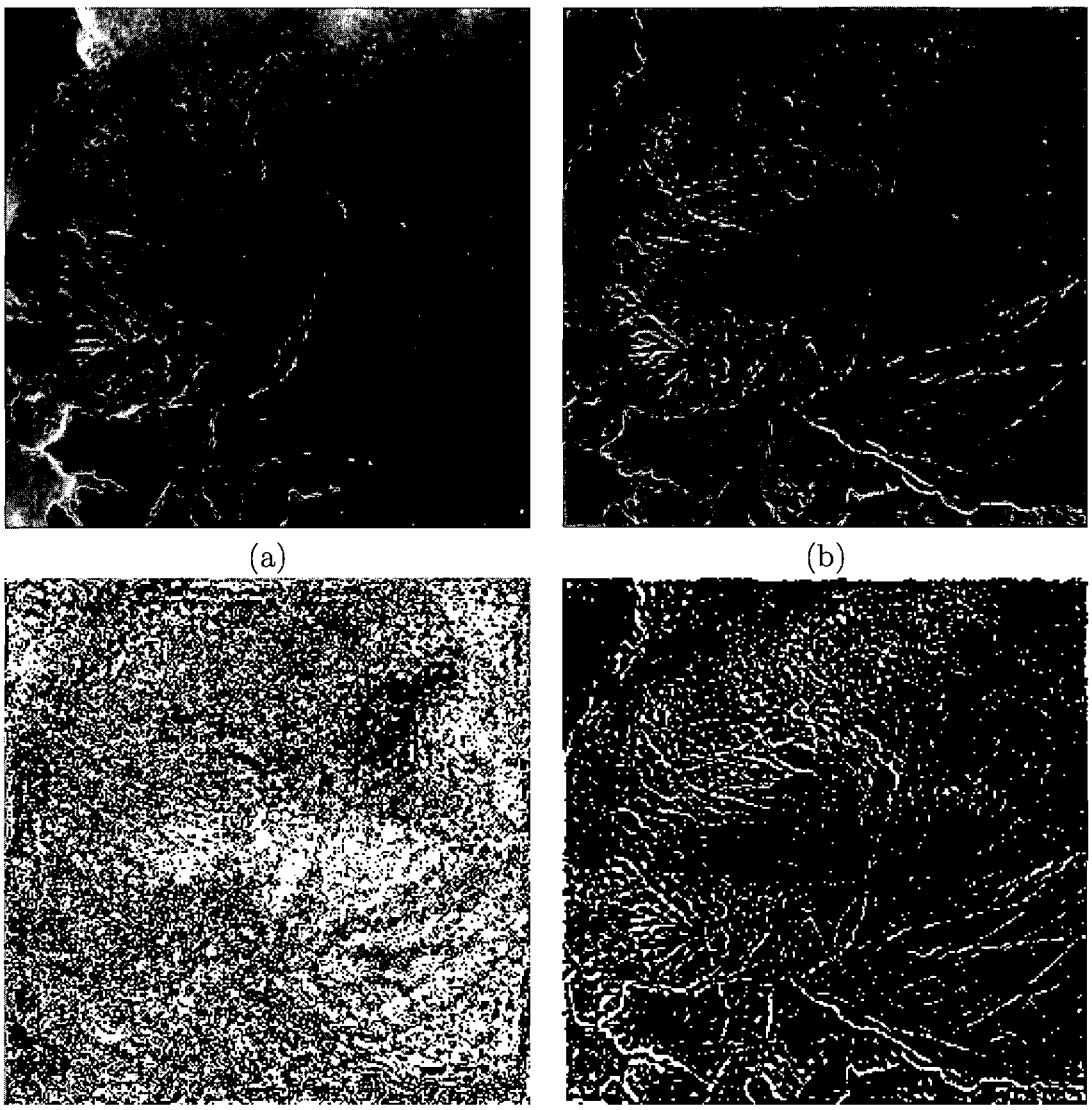


FIG. 5.25 – Le nettoyage du filtre spectral idéal 3D, avec seize images, une fréquence de coupure spatiale à 30%, une fréquence de coupure temporelle à 75%, $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$, classifiée par le filtre des k-moyennes contextuel.

F-measure	Algorithme	k-moyennes	Figure
0,262	gaussien 3D	contextuel	figure 5.27
0,261	gaussien 3D	contextuel	figure 5.29
0,252	gaussien 3D	original	figure 5.26
0,245	gaussien 3D	original	figure 5.28

TAB. 5.7 – Classement du filtre passe-haut gaussien en trois dimensions selon la *F-measure*.

du filtre des k-moyennes contextuel.

Les figures 5.28 et 5.29 illustrent le résultat de l'application de ce filtre sur une séquence de seize images avec l'image de référence en huitième position, c'est-à-dire que la séquence entière est utilisée sans répétition ni mirroir. La fréquence de coupure au niveau spatial est de 30%, la fréquence de coupure au niveau temporel est de 75% et la pente est $c = 2$. Les bornes du rehaussement de coutours sont $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$. La première figure présente la classification du filtre des k-moyennes original et la seconde présente la classification du filtre des k-moyennes contextuel.

Le choix de la fréquence de coupure spatiale de 30% a été faite puisque une fréquence retenant les basses et les moyennes fréquences donnait de bons résultats pour les filtres spectraux classiques.

La première observation des résultats de ce filtre est que les deux images nettoyées, 5.27(b) et 5.29(b) sont presque identiques malgré le fait que le nombre d'images diffère grandement, quatre images et seize images.

Malgré le fait que les images nettoyées semblent similaires à celles du filtre passe-haut gaussien en deux dimensions, les résultats de ce filtre de classification sont très bons (comparativement aux résultats précédents) et sont présentés au tableau 5.7.

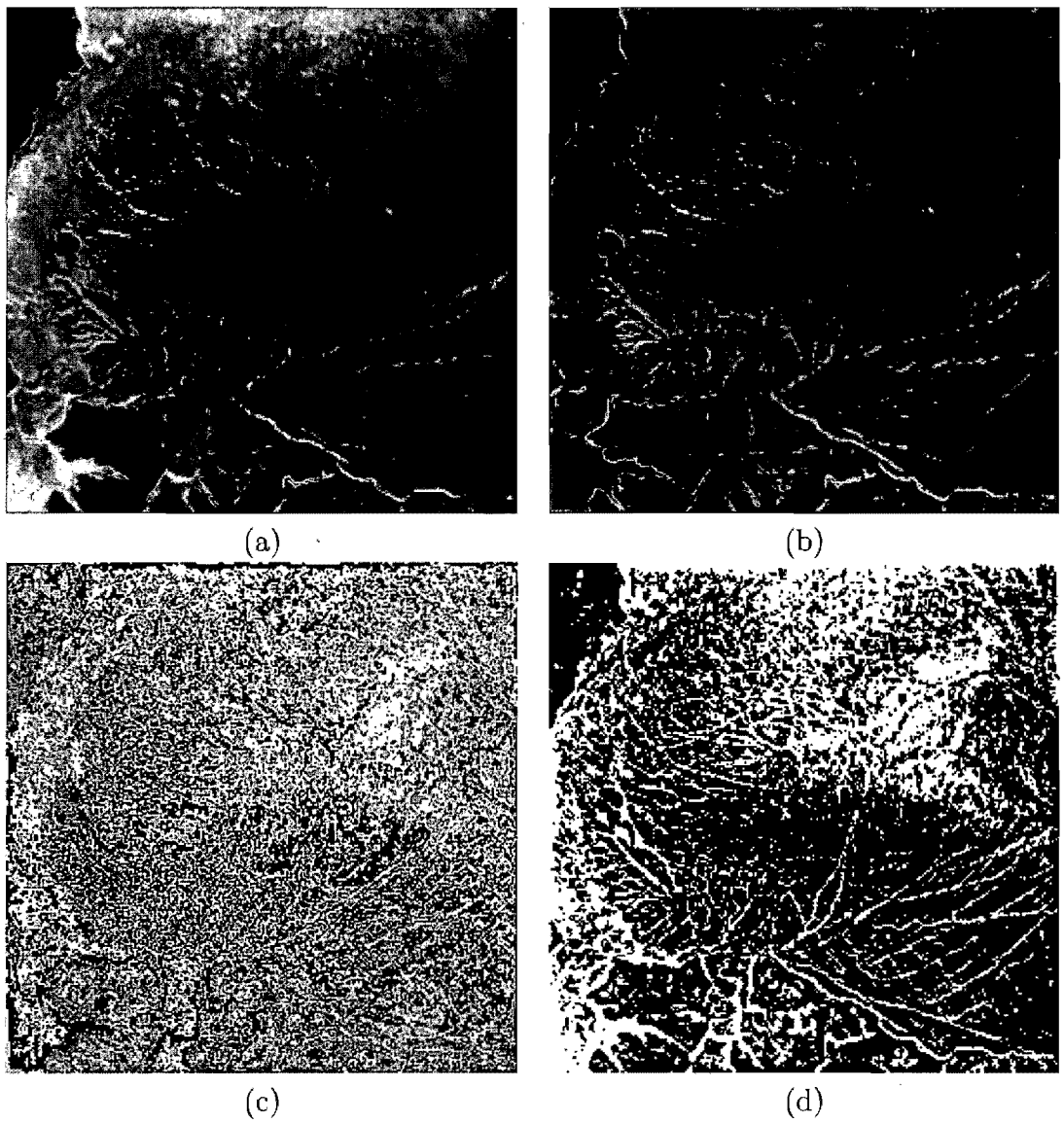


FIG. 5.26 – Le nettoyage du filtre spectral gaussien 3D, avec quatre images, une fréquence de coupure spatiale à 30%, une fréquence de coupure temporelle à 75%, une pente de $c = 2$, $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$, classifiée par le filtre des k-moyennes original.

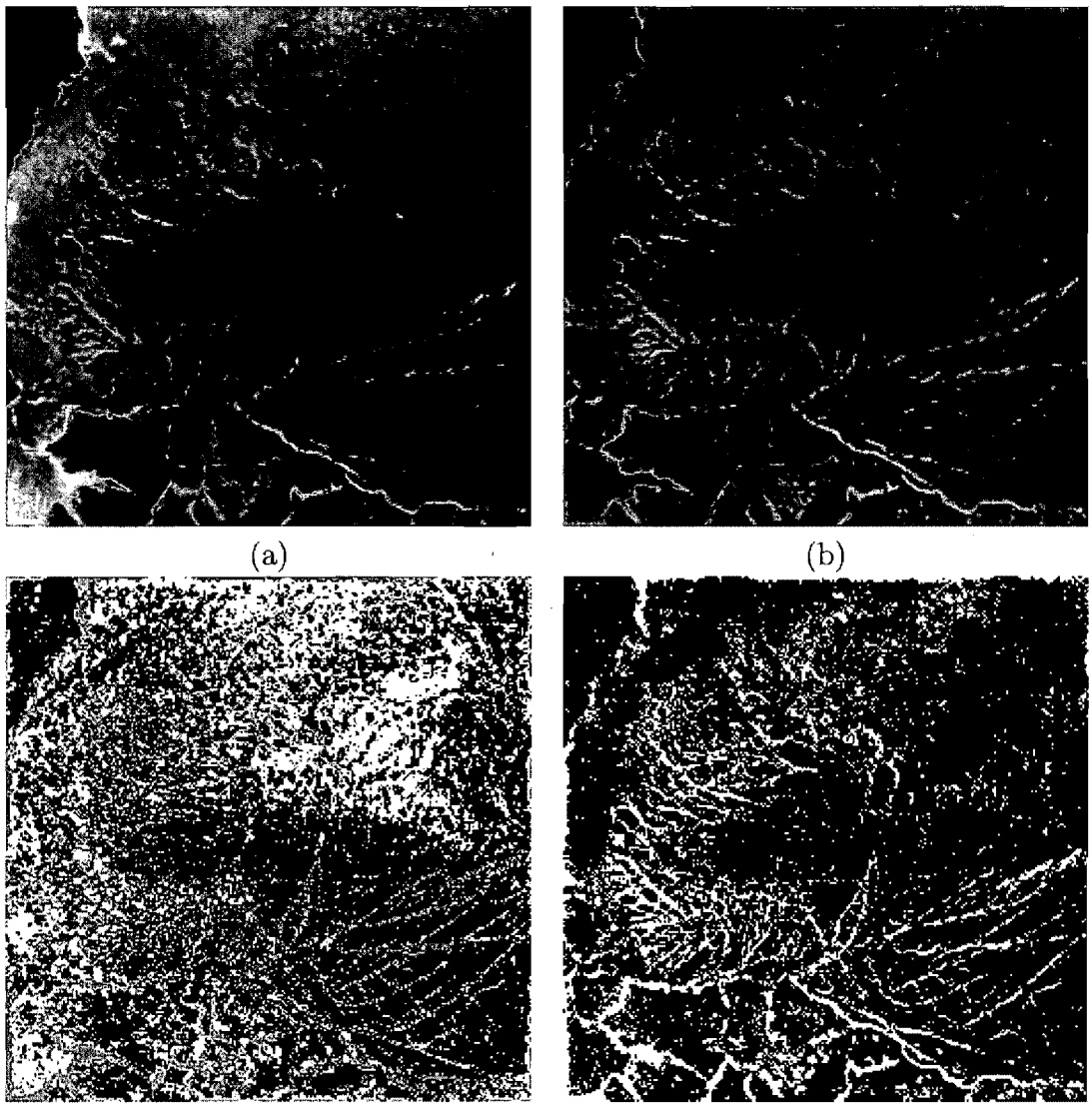


FIG. 5.27 – Le nettoyage du filtre spectral gaussien 3D, avec quatre images, une fréquence de coupure spatiale à 30%, une fréquence de coupure temporelle à 75%, une pente de $c = 2$, $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$, classifiée par le filtre des k-moyennes contextuel.

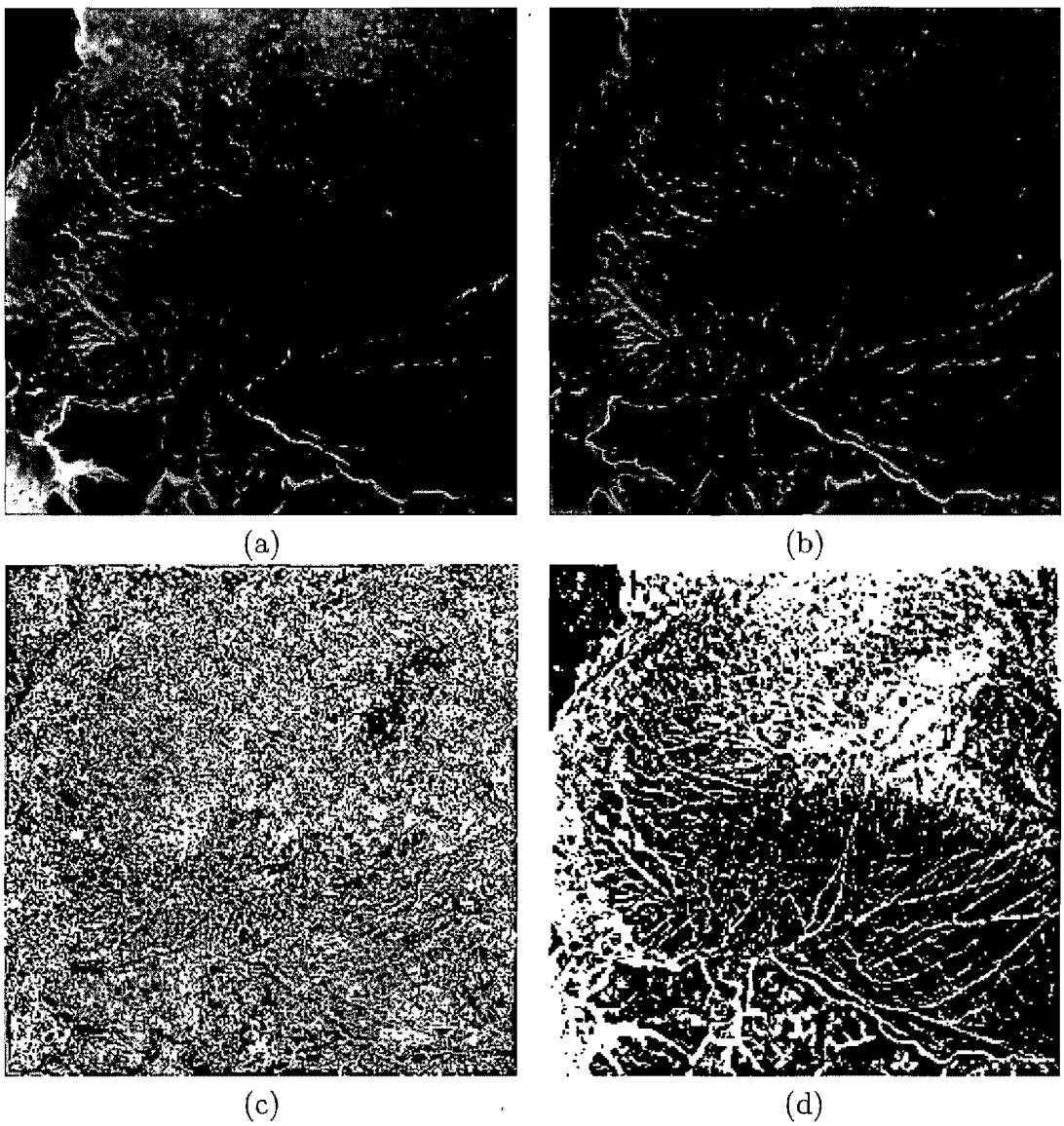
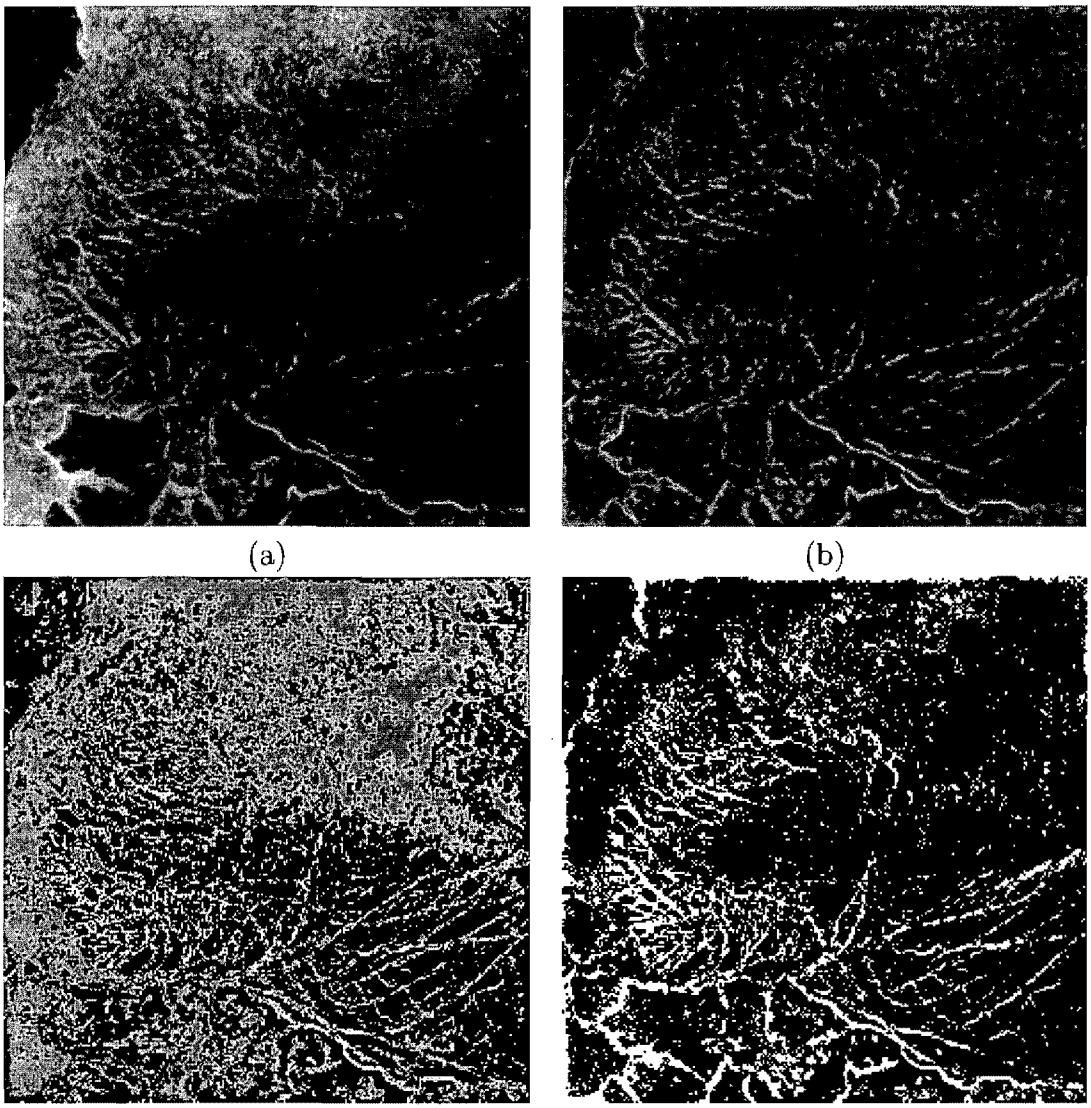


FIG. 5.28 – Le nettoyage du filtre spectral gaussien 3D, avec seize images, une fréquence de coupure spatiale à 30%, une fréquence de coupure temporelle à 75%, une pente de $c = 2$, $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$, classifiée par le filtre des k-moyennes original.



(a) (b)

(c) (d)

FIG. 5.29 – Le nettoyage du filtre spectral gaussien 3D, avec seize images, une fréquence de coupure spatiale à 30%, une fréquence de coupure temporelle à 75%, une pente de $c = 2$, $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$, classifiée par le filtre des k-moyennes contextuel.

5.4 Classement des images résultats

L'algorithme de la *F-measure* a été appliqué sur chacune des images résultats présentées dans la section précédente et le classement général est présenté au tableau 5.8. La figure 5.30 illustre les huit meilleures images.

En regardant ce classement, il est visible que les algorithmes spectraux donnent les meilleurs résultats, surtout les filtres gaussiens et en particulier le filtre gaussien en trois dimensions. En effet, les première, deuxième et cinquième positions sont tenues par le même filtre gaussien : la différence entre la première et la deuxième position étant le nombre d'images (respectivement quatre et seize) et la différence entre la première et la cinquième position étant l'algorithme de classification (respectivement les k-moyennes contextuel et original). Il est aussi sans équivoque que le filtre de classification des k-moyennes contextuel l'emporte sur le filtre de classification des k-moyennes original.

Ce filtre gaussien en trois dimensions a été exploré plus en profondeur et les résultats sont affichés dans les tableaux 5.9 et 5.10. Les valeurs dans ces tableaux pour chacune des rangées correspondent au nombre d'images : quatre, huit et seize tandis que les valeurs pour chacune des colonnes correspondent aux fréquences de coupure au niveau spatial : 20%, 30% et 40%. Le tableau 5.9 présente les résultats pour une fréquence de coupure temporelle à 75% et le tableau 5.10 pour une fréquence de coupure à 50%. Les autres paramètres sont restés les mêmes : la pente est $c = 2$ et les bornes du rehaussement de coutours sont $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$. Le filtre de classification est le filtre des k-moyennes contextuel. Les valeurs en gras sont les maxima de chacun des tableaux et les valeurs en italiques sont les valeurs du tableau 5.8.

De ces deux tableaux une seule conclusion peut être tirée, les images nettoyées avec une fréquence de coupure de 40% semblent performer moins bien que toutes les autres.

F-measure	**Algorithme**	***k*-moyennes**	**Figure**
0,262	gaussien 3D	contextuel	figure 5.27
0,261	gaussien 3D	contextuel	figure 5.29
0,257	gaussien 2D	contextuel	figure 5.15
0,254	butterworth 2D	contextuel	figure 5.19
0,252	gaussien 3D	original	figure 5.26
0,249	idéal 2D	contextuel	figure 5.11
0,245	idéal 3D	contextuel	figure 5.23
0,245	gaussien 3D	original	figure 5.28
0,237	passe-haut spatial	contextuel	figure 5.9
0,235	butterworth 2D	original	figure 5.18
0,221	butterworth 2D	contextuel	figure 5.21
0,213	idéal 3D	contextuel	figure 5.25
0,213	idéal 2D	contextuel	figure 5.13
0,213	gaussien 2D	contextuel	figure 5.17
0,209	idéal 3D	original	figure 5.24
0,205	idéal 3D	original	figure 5.23
0,202	idéal 2D	original	figure 5.10
0,197	passe-haut spatial	contextuel	figure 5.7
0,195	idéal 2D	original	figure 5.12
0,192	gaussien 2D	original	figure 5.14
0,190	butterworth 2D	original	figure 5.20
0,175	passe-haut spatial	original	figure 5.6
0,170	gaussien 2D	original	figure 5.16
0,154	identité	contextuel	figure 5.3
0,146	identité	original	figure 5.2
0,135	passe-haut spatial	original	figure 5.8

TAB. 5.8 – Classement des images résultats selon la *F-measure*.

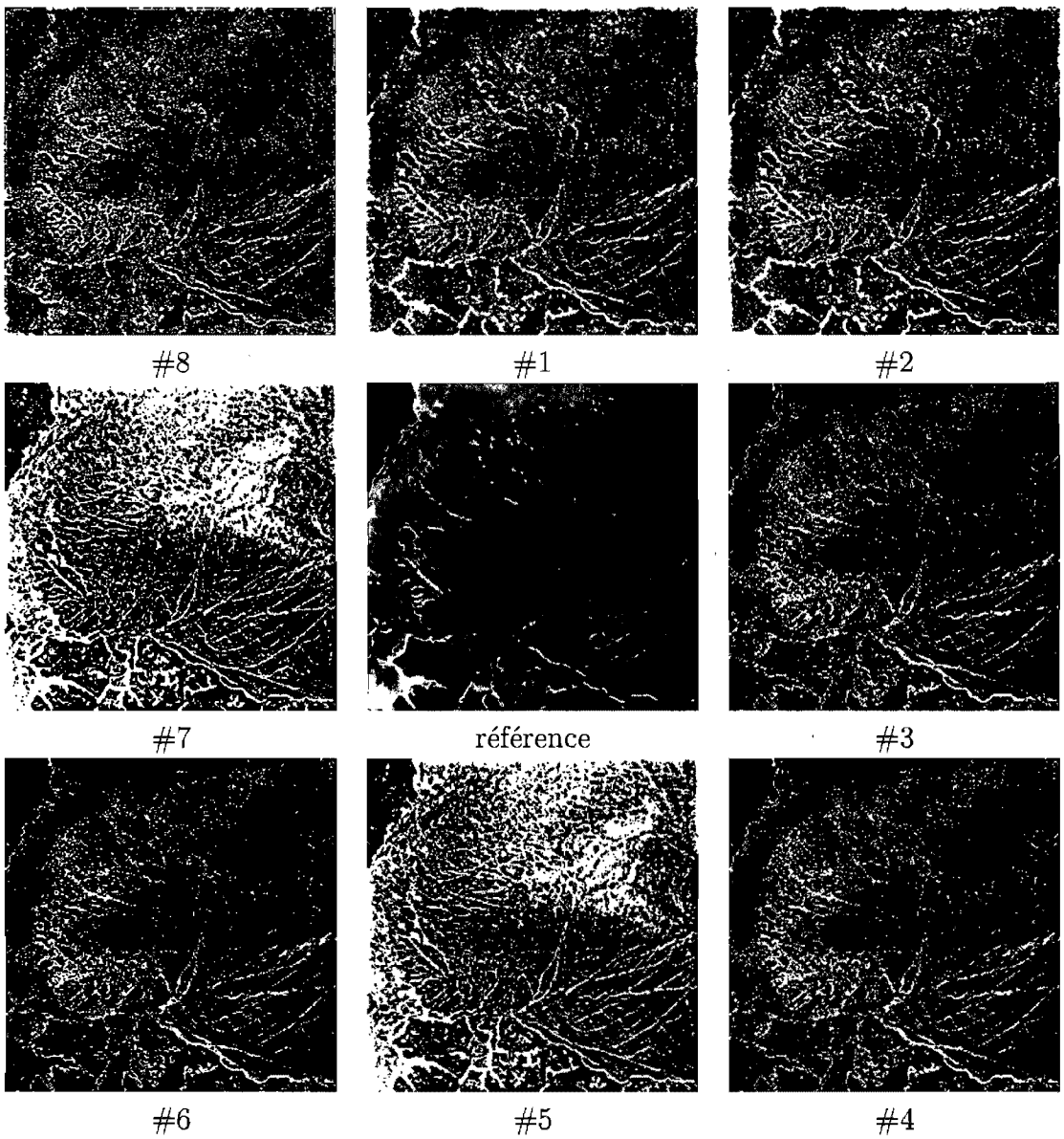


FIG. 5.30 – Présentation des huit meilleures classifications, en ordre horaire en commençant à midi. L'image de référence est au centre et la figure correspondante est listée au tableau 5.8

	20%	30%	40%
4	0.238	*0.262*	0.224
8	0.248	0.223	0.249
16	**0.266**	*0.261*	0.198

TAB. 5.9 – Valeur de la *F-measure* pour le filtre gaussien en trois dimensions avec la fréquence de coupure temporelle à 75%, la pente $c = 2$, les bornes du rehaussement de coutours $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$ et classifié avec le filtre des k-moyennes contextuel. Les rangées changent le nombre d'images et les colonnes changent la fréquence de coupure spatiale.

	20%	30%	40%
4	**0.263**	0.219	0.254
8	0.240	0.235	0.190
16	0.240	0.235	0.193

TAB. 5.10 – Valeur de la *F-measure* pour le filtre gaussien en trois dimensions avec la fréquence de coupure temporelle à 50%, la pente $c = 2$, les bornes du rehaussement de coutours $\gamma_{min} = 0,10$ et $\gamma_{max} = 0,90$ et classifié avec le filtre des k-moyennes contextuel. Les rangées changent le nombre d'images et les colonnes changent la fréquence de coupure spatiale.

5.5 Commentaires généraux

D'après les résultats présentés par les tableaux des sections précédentes, le meilleur filtre de nettoyage semble être le filtre passe-haut gaussien en trois dimensions. Ce filtre est suivi de près par tous les autres filtres spectraux, en deux et en trois dimensions.

Il est aussi étonnant que, pour les filtres spectraux, les petites fréquences de coupure, (i.e. de l'ordre de 5%), donnent de meilleures classifications que les plus grandes fréquences de coupure (i.e. de l'ordre de 60%) malgré l'apparence plus claire des images nettoyées.

Quant aux filtres de classification, le problème est beaucoup plus simple. Le filtre des k-moyennes classique donne de piètres résultats puisqu'il ne tient compte que de la couleur. Ceci est inadéquat pour les images de ce projet car celles-ci ne sont pas stationnaires, c'est-à-dire qu'un pixel identifié par un expert comme représentant un chenal peut être de la même luminosité qu'un autre pixel ailleurs dans l'image identifié par le même expert comme faisant partie de la banquise.

La version contextuelle du filtre des k-moyennes tente de résoudre ce problème de non-stationnarité en ajoutant à la couleur du pixel une valeur correspondant à la différence de couleur de ce pixel avec son voisinage. Les résultats de cette version sont vraiment plus proches de ce qui est recherché.

Il faut rappeler finalement que le choix des centres des deux versions de l'algorithme de k-moyennes est laissé au hasard et que plusieurs classifications d'une même image peuvent donner des résultats différents, ce qui signifie que l'algorithme des k-moyennes tombe sûrement dans un minimum local, un signe que le problème de classification est complexe et/ou que les classes sont très mélangées.

CHAPITRE 6

LES LIVRABLES

Ce chapitre traite des livrables, les logiciels et composantes conçus pour le nettoyage et la classification des images. Tout le code source, les logiciels et les composantes sont disponibles dans la distribution fournis avec ce mémoire. Le contenu de la distribution est disponible à l'annexe IV.

6.1 Le choix de Java

Le langage de programmation utilisé pour réaliser les logiciels utilisés durant ce projet de maîtrise est le langage Java. Ce langage a été utilisé à cause de la disponibilité et de la puissance des bibliothèques externes concernant le traitement d'image et la création de logiciels avec une interface graphique.

6.1.1 La bibliothèque `JAI` et `JAI-ImageIO`

Le coeur du langage de programmation Java ne contient pas vraiment de possibilités pour le traitement des images. Il y a des classes pour faire des manipulations simples dans le *package* `java.awt`, mais celles-ci sont surtout limitées aux transformations linéaires et à l'affichage des images dans un contexte graphique (écran et impression).

Par contre, Sun Microsystems inc., la compagnie qui a créé le langage Java, a aussi créé une bibliothèque d'extension pour la manipulation d'images. Cette bibliothèque, la *Java Advanced Imaging* (`JAI`), offre une interface orientée-objet simple qui permet de manipuler facilement des images [27].

Cette bibliothèque a été choisie parce qu'elle offre un grand choix d'opérations de base (entre autres les transformées de Fourier) et aussi parce qu'il est très facile d'étendre cette bibliothèque pour y ajouter de nouvelles opérations de manipulation. D'ailleurs, plusieurs opérations des logiciels créés pour ce projet sont mainte-

nant des opérateurs utilisant l'architecture de cette bibliothèque et distribués dans leur propre bibliothèque, la bibliothèque `jai-operators` décrite à la section 6.8.

Une autre bibliothèque reliée à `JAI` a été utilisée, il s'agit de la bibliothèque `JAI-ImageIO` [29]. Cette bibliothèque offre de nouveaux opérateurs `JAI` qui permettent d'adapter les encodeurs et décodeurs d'images offerts par les bibliothèques principales du langage Java qui sont devenus disponibles à partir de la version 1.4 du langage Java. Cette bibliothèque est destinée à remplacer les encodeurs et décodeurs d'images désuets inclus dans `JAI`.

6.1.2 Les bibliothèques de JGoodies

La compagnie allemande *JGoodies* [30, 31] se spécialise dans les interfaces graphiques, en particulier comment les rendre fonctionnelles et agréables à utiliser. En plus de ses services payants, elle offre gratuitement des bibliothèques Java pour facilement construire et manipuler les interfaces graphiques ainsi que leurs données sous-jacentes.

Trois bibliothèques gratuites ont été utilisées : *JGoodies Binding* [32] qui lie les composantes graphiques au modèle de données ; *JGoodies Forms* [33] qui permet de construire une interface fonctionnelle et ; *JGoodies Looks* [34] qui permet de rendre constantes et élégantes chacune des composantes de l'interface graphique.

Ces trois bibliothèques ont été utilisées dans la réalisation de toutes les applications graphiques décrites au chapitre 6.

6.1.3 La bibliothèque JUnit

La bibliothèque JUnit [35] offre une architecture servant à créer, garder et utiliser des tests unitaires [36].

Cette bibliothèque a été utilisée pour tester, entre autres, les filtres utilisés pour nettoyer les images. Elle a été aussi utilisée pour tester, de manière exhaustive, la bibliothèque `jai-operators`.

6.1.4 Les bibliothèques *Forklabs*

Les dernières bibliothèques utilisées, `baselib` [38] et `hdr-codec` [39], sont les bibliothèques personnelles de l'auteur écrites au fil des ans. Ces bibliothèques sont des bibliothèques *Open Source* distribuées sous le nom de *Forklabs* [37].

Elles contiennent pour la première du code utilitaire complémentant les bibliothèques coeurs du langage Java et pour la seconde des encodeurs et décodeurs pour des images à haut contraste [40].

6.2 La classe `ImageCanvas`

Les bibliothèques de base du langage Java n'ont pas vraiment de composantes pour afficher simplement des images, si ce n'est la classe `JLabel`[1] en utilisant une `Icon`[2]. De plus, certains comportements étaient souhaités, comme facilement déplacer l'image ainsi que l'agrandir ou la rapetisser.

La classe `ImageCanvas`[3] est la composante principale pour l'affichage des images dans toutes les applications réalisées dans le cadre de ce travail de maîtrise. La figure 6.1 illustre le logiciel `ClusterViewer` qui, dans sa partie principale, utilise une instance de la classe `ImageCanvas` pour afficher l'image.

Pour déplacer l'image, il suffit simplement de *dragger* avec le bouton principal de la souris. Une méthode est exposée pour déplacer l'image en utilisant d'autres moyens.

Pour changer l'agrandissement, trois manières équivalentes sont possibles : l'utilisation des boutons dans le coin supérieur droit ; l'utilisation de la roulette de la souris et ; l'utilisation du menu contextuel. Ces trois manières naviguent une liste de facteurs d'aggrandissement fournis à la création du composant. Deux `Action`[4] ainsi que deux méthodes sont disponibles pour naviguer les facteurs d'agrandissement par d'autres moyens.

---

[1] `javax.swing.JLabel`
[2] `javax.swing.Icon`
[3] `ca.umontreal.iro.image.arcticice.analyser.swing.ImageCanvas`
[4] `javax.swing.Action`

Finalement, il est possible d'ajouter des items au menu contextuel. Par exemple, les applications `GUIAnalyser` et `ClusterViewer` permettent de sauvegarder n'importe quelle image en utilisant le menu contextuel.

6.3 `Cropper`

L'application `Cropper` est utilisée pour extraire la région d'intérêt des images brutes fournies par la NASA. Ce sont les images créées par cette petite application qui seront utilisées par le logiciel `GUIAnalyser`. Cette application est divisée en deux programmes distincts.

Le premier programme est le script d'extraction des images. Ce script extrait des fichiers HDF les images pertinentes, décrites à la section 3.4, et son code est disponible à l'annexe I. Ce script est très limité, il extrait les images de tous les fichiers dans un même répertoire.

Le second programme est une petite application Java qui s'occupe de charger les images extraites, de les *cropper*, de remplir les trous noirs et d'unifier les images des senseurs de polarité horizontale et verticale ; les étapes des sections 3.5, 3.6 et 3.7.

Comme le script, ce programme est très limité et existe en deux versions, une première version pour générer les images de format 512 pixels par 512 pixels, `HighResCropper`[5], et une seconde version pour les images de format 256 pixels par 256 pixels, `HighResCropper2`[6]. Si d'autres dimensions ou d'autres régions d'intérêt sont désirées, un nouveau programme devra être écrit.

Si un programme général est aussi désiré, il devra tenir compte des dimensions et de la position de la zone d'intérêt. Il devra aussi tenir compte des noms des fichiers originaux et des noms des fichiers pour les images générées.

---

[5] `ca.umontreal.iro.image.arcticice.launcher.HighResCropper`

[6] `ca.umontreal.iro.image.arcticice.launcher.HighResCropper2`

```
$ java -jar diro-arctic-ice-quick.hdf.viewer-0.0-M0.jar [filename width height]
```

Listing 6.1 – La commande pour lancer `QuickHDFViewer`

6.4 `QuickHDFViewer`

Les images utilisées proviennent de fichiers archives fournis par la NASA (une description complète des images est présentée au chapitre 3). Il existe plusieurs logiciels pour visionner et extraire le contenu de ces fichiers archives, dont `HDFView`[7], un logiciel écrit en Java par la compagnie *The HDF Group*[8]. Ce logiciel permet entre autres, de visualiser les images incluses dans l'archive HDF.

Le logiciel `HDFView` est utile pour visionner l'image originale à l'intérieur d'un fichier archive HDF. Par contre, puisque les images sont extraites des archives et ensuite manipulées par les étapes décrites au chapitre 3, le logiciel `HDFView` est incapable de visionner ces nouvelles images. Ce travail de visionnement est celui du logiciel `QuickHDFViewer`.

Ce logiciel très simple est lancé par la commande illustrée au listing 6.1 et ne possède que trois commandes : charger une image, sauvegarder l'image courante et quitter l'application.

Une note cependant, il ne faut pas oublier de spécifier les dimensions de l'image brute chargée puisqu'aucune indication n'est disponible dans les données contenues dans le fichier lui-même, il faut les connaître.

6.5 `ClusterViewer`

Le logiciel `ClusterViewer` est un petit logiciel qui permet de visionner seulement une partie des niveaux de gris d'une image. La figure 6.1 illustre le logiciel `ClusterViewer` sur une image filtrée par un filtre k-moyennes. Le logiciel affiche dans sa partie gauche en premier l'image originale. La partie droite contient les contrôles qui permettent d'afficher ou de cacher chacun des amas.

[7] `http://www.hdfgroup.org/hdf-java-html/hdfview/`
[8] `http://www.hdfgroup.org/`

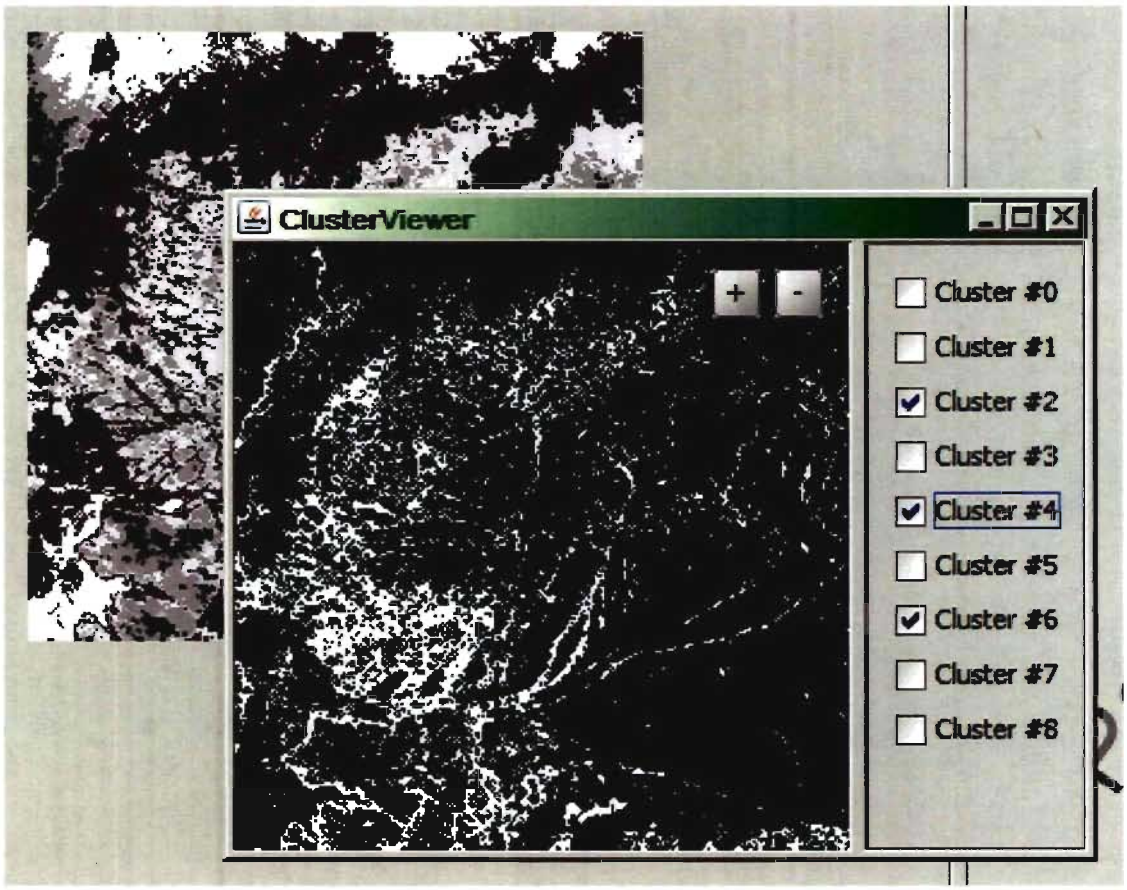


FIG. 6.1 – Le logiciel `ClusterViewer` avec le logiciel `GUIAnalyser` dans le fond.

```
$ java -jar diro-arctic-ice-quick.hdf.viewer-0.0-M0.jar <filename>
```

Listing 6.2 – La commande pour lancer `ClusterViewer`

L'utilisation la plus fréquente de ce logiciel est par son intégration au logiciel `GUIAnalyser` (dans le menu contextuel des images classifiées par une version de l'algorithme k-moyennes), mais il peut être lancé par lui-même pour examiner des images externes, comme illustré au listing 6.2.

Il est aussi possible de sauvegarder l'image présentée en faisant apparaître le menu contextuel, celui-ci contient une option pour sauvegarder l'image.

6.6 Les filtres

Tous les filtres qui ont été essayés ont été réalisés avec la bibliothèque `JAI`. Certains ont été réalisés comme opérateurs complets tandis que d'autres l'ont été comme paramètres à des opérateurs déjà existants.

Filtre identité Le filtre identité utilise l'opérateur `JAI null` pour ne pas altérer l'image de base lors d'opérations subséquentes.

Filtre homomorphique spatial Le filtre passe-haut homomorphique spatial a été réalisé comme opérateur `JAI`, l'opérateur `spatialhighpass`.

Filtre idéal 2D La classe `IdealHighPassFilterImageFunction`[9] réalise le filtre passe-haut idéal 2D. Une instance de cette classe adaptée par une instance de la classe `ImageFunctionSpectralHomomorphicFilter`[10] est utilisée comme paramètre pour l'opérateur `spectralfilter` et l'opérateur `spectralhomomorphic`.

Filtre idéal 3D La classe `IdealHighPassFilterImageFunction3D`[11] réalise le filtre passe-haut idéal 3D. Une instance de cette classe adaptée par une instance de la classe `ImageFunctionSpectralHomomorphicFilter3D`[12] est utilisée comme paramètre pour l'opérateur `spectralfilter` et l'opérateur `spectralhomomorphic`.

Filtre gaussien 2D La classe `GaussianHighPassFilterImageFunction`[13] réalise le filtre passe-haut gaussien 2D. Une instance de cette classe adaptée par une instance de la classe `ImageFunctionSpectralHomomorphicFilter` est utilisée comme paramètre pour l'opérateur `spectralfilter` et l'opérateur `spectralhomomorphic`.

Filtre gaussien 3D La classe `GaussianHighPassFilterImageFunction3D`[14] réa-

[9] `c.u.i.i.a.media.jai.function.IdealHighPassFilterImageFunction`
[10] `c.u.i.i.a.media.jai.spectralfilter.ImageFunctionSpectralHomomorphicFilter`
[11] `c.u.i.i.a.arcticice.media.jai.function.IdealHighPassFilterImageFunction3D`
[12] `c.u.i.i.a.media.jai.spectralfilter.ImageFunctionSpectralHomomorphicFilter3D`
[13] `c.u.i.i.a.media.jai.function.GaussianHighPassFilterImageFunction`
[14] `c.u.i.i.a.media.jai.function.GaussianHighPassFilterImageFunction3D`

lise le filtre passe-haut gaussien 3D. Une instance de cette classe adaptée par une instance de la classe `ImageFunctionSpectralHomomorphicFilter3D` est utilisée comme paramètre pour l'opérateur `spectralfilter` et l'opérateur `spectralhomomorphic`.

Filtre de Butterworth La classe `ButterworthHighPassFilterImageFunction`[15] réalise le filtre de Butterworth. Une instance de cette classe adaptée par une instance de la classe `ImageFunctionSpectralHomomorphicFilter` est utilisée comme paramètre pour l'opérateur `spectralfilter` et l'opérateur `spectralhomomorphic`.

Filtre des k-moyennes Le filtre de classification des k-moyennes a été réalisé comme opérateur `JAI`, l'opérateur `kmeans` de la bibliothèque `jai-operators`. La version originale du filtre utilise la fonction d'évaluation comprise dans l'opérateur tandis que la version contextuelle utilise une instance de la classe `RobustKMeansEvaluationFunction`[16] comme fonction d'évaluation.

Au moment de l'écriture de ce mémoire, sauf le filtre des k-moyennes, aucun de ces filtres ne faisaient parties de la bibliothèque `jai-operators`.

6.6.1 Filtre spectraux

Les filtres spectraux : idéal, gaussien et de Butterworth, ont été réalisés comme spécialisation de l'interface `ImageFunction`[17] pour les filtres en deux dimensions et de l'interface `ImageFunction3D`[18] pour les filtres en trois dimensions.

Il est à noter qu'une façade pourrait être réalisée en tant qu'opérateur `JAI` pour cacher la complexité de configuration de ces filtres.

---

[15] `c.u.i.i.a.media.jai.function.ButterworthHighPassFilterImageFunction`
[16] `c.u.i.i.a.analyser.RobustKMeansEvaluationFunction`
[17] `javax.media.jai.ImageFunction`
[18] `ca.forklabs.media.jai.ImageFunction3D`

6.6.2 Les filtres 3D

Comme mentionné au début du chapitre 3, les images chargées dans l'application `GUIAnalyser` forment une séquence, la première image est supposée être plus vieille que la seconde, qui elle-même est supposée être plus vieille que la troisième et ainsi de suite.

Quand un filtre a besoin de plusieurs images en entrée, entre autres les filtres spectraux 3D et la technique de la médiane temporelle, et que ce filtre est appliqué à une image à l'extrémité de la séquence, il est possible que l'intervalle contienne des images manquantes.

La figure 6.2 illustre cette situation, l'image 1 est l'image qui sera filtrée par un filtre ayant besoin d'une séquence de sept images, l'image centrale et trois images des deux côtés.

Il y a deux solutions pour résoudre ce problème, illustrées aux figures 6.3 et 6.4. Il s'agit de faire un mirroir de la séquence ou encore copier l'image frontière. Les algorithmes spectraux utilisent la technique du mirroir tandis que la technique de la médiane temporelle utilise la technique de la copie.

6.7 `GUIAnalyser`

Le logiciel `GUIAnalyser` est l'application principale qui permet de nettoyer les images de base et de classifier les chenaux. Le mode d'emploi de ce logiciel est décrit à l'annexe II et son architecture, spécifiquement l'intégration des filtres à l'interface graphique, est décrit à l'annexe III.

6.8 Projet `jai-operators`

Le langage Java et sa bibliothèque `JAI` ont été choisis pour la réalisation des outils de recherche et des algorithmes de nettoyage et d'analyse des images. La bibliothèque `JAI` est facilement extensible et les algorithmes de nettoyages et d'analyse ont été architecturés comme nouveaux opérateurs pour cette bibliothèque. Une grande partie de ceux-ci ont été regroupés dans la bibliothèque `jai-operators`,

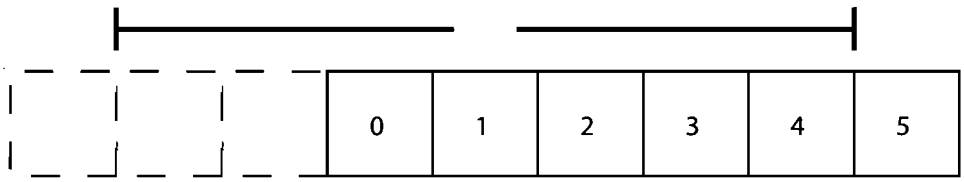


FIG. 6.2 – Séquence de sept images centrées sur l'image 1, dont deux images inexistantes faisant partie de cet intervalle.

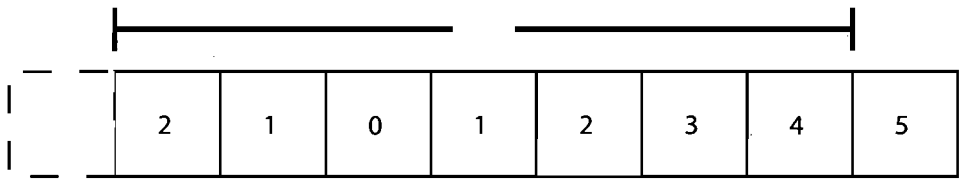


FIG. 6.3 – Séquence de sept images centrées sur l'image 1 utilisant la technique du miroir pour donner vie aux images inexistantes.

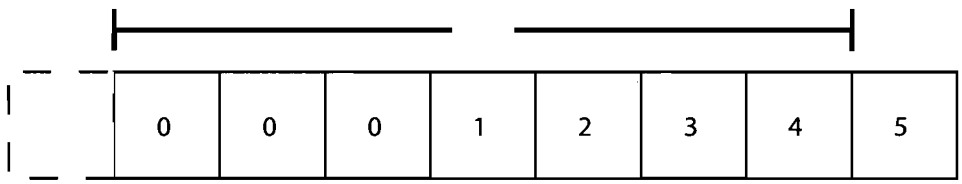


FIG. 6.4 – Séquence de sept images centrées sur l'image 1 utilisant la technique de la copie pour donner vie aux images inexistantes.

un projet indépendant. La bibliothèque `jai-operators` est maintenant un projet *Open Source* résidant sur le site communautaire de Java, le site `java.net`[19]. De plus, la bibliothèque `jai-operators` est sous l'ombrelle du projet maître `JAI`[20] dont le propriétaire est Sun Microsystems, Inc., la compagnie à l'origine du langage de programmation Java et de la bibliotheque `JAI`. Au moment de l'écriture de ce mémoire, la bibliothèque `jai-operators` comprend treize opérateurs, certains sont originaux et d'autres sont des extensions d'opérateurs déjà existants. La description complète de ces opérateurs est disponible à l'annexe V

[19] `https://jai-operators.dev.java.net/`
[20] `https://jai.dev.java.net/`

CHAPITRE 7

LES TRAVAUX FUTURS

Le but ultime pour lequel ce projet est un prototype est de pouvoir détecter automatiquement les chenaux et ensuite pouvoir les quantifier et aider les scientifiques à estimer le moment fatidique où l'océan Arctique sera libre de glace durant son été. Les résultats et livrables de ce projet de maîtrise ne sont que les premiers pas. Les autres pas peuvent prendre plusieurs directions plus ou moins indépendantes et peuvent être faits autant du côté de la recherche scientifique que du côté des procédés.

7.1 Filtres de classification

Les deux versions du filtre des k-moyennes ont de la difficulté à classifier l'image parce que celle-ci est non-stationnaire. De plus les régions recherchées sont des zones linéiques. Il faudrait alors rechercher un filtre de classification globale qui soit plus robuste à la détection de ces zones linéiques.

7.2 Filtres normaux vs filtres homomorphiques

Le logiciel `GUIAnalyser` présente deux images candidates lorsqu'une image est nettoyée par un filtre spectral (cf. section III.1.3). Le premier candidat est celui du filtre normal et le second candidat est celui du filtre homomorphique. Le logiciel présente aussi une troisième image, l'image de la différence entre ces deux versions. Une telle différence est illustrée à la figure 7.1 et les chenaux y sont assez clairement visibles. Cette image n'est pas du tout exploitée dans le cadre de ce travail.

7.3 Estimation de la largeur et de la longueur

Les images utilisées dans ce travail ont une résolution de 6,25 kilomètres. D'après les résultats du *Arctic Lead Experiment*, il y a très peu de chenaux dont la largeur

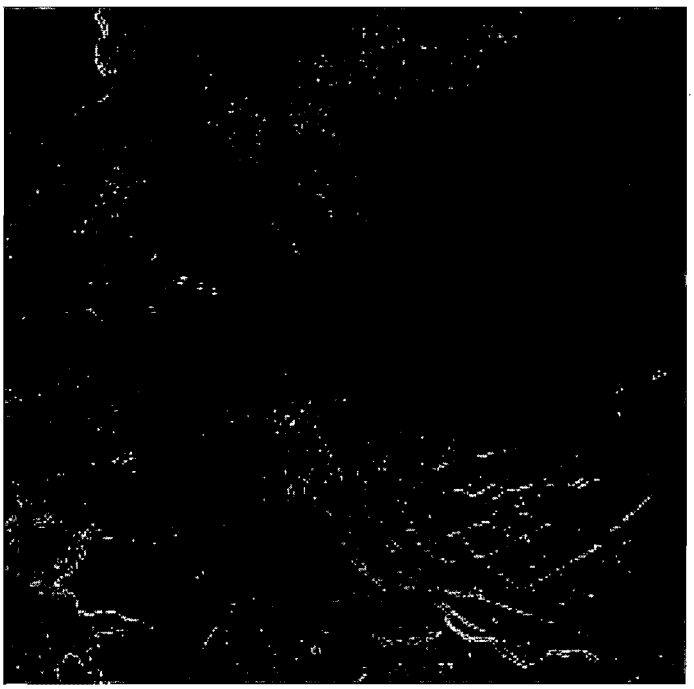


FIG. 7.1 – Image de la distance L1 entre la version normale et la version homomorphique d'un même filtre.

est de l'ordre du kilomètre, pourtant les chenaux sont visibles sur ces images.

Il existe des routines utilisées dans la IDL[1] pour visualiser les images de ce projet et déterminer la température de chacun des niveaux de gris. Connaissant la température d'un pixel classé comme un chenal, la température de l'océan et la température des pixels voisins, il serait alors possible d'estimer la largeur du chenal à l'endroit de ce pixel. Ce travail est du domaine de l'estimation sub-pixelique.

L'estimation de la longueur sera beaucoup plus facile, il ne suffirait que de compter le nombre de pixels représentant le chenal.

[1] `http://www.ittvis.com/idl/`

7.4 Détection automatique des chenaux

Afin de pouvoir détecter automatiquement les chenaux, il faut pouvoir détecter la ou les classes les représentant. Cette détection en elle-même est un travail complexe.

Cependant, une adaptation des travaux de [11] pourrait être utilisée lorsqu'un bon filtre de nettoyage sera trouvé. L'expert n'aura qu'à classer que quelques images et, en supposant une application constante du même filtre de nettoyage, ce filtre de classification pourra être utilisé pour classifier les images de plusieurs années.

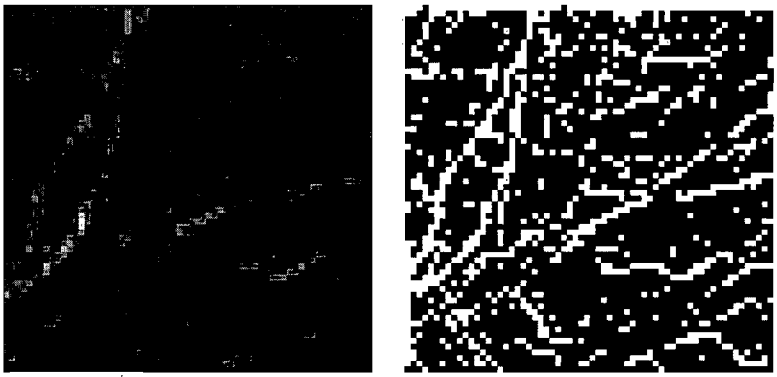
Une autre idée pour aider la détection serait de dessiner à main levée, un peu comme dans les logiciels de dessin tels que Photoshop™ ou MSPaint™, une zone d'intérêt pour la classification. L'avantage de cette idée est qu'il serait alors facile de rabouter des chenaux apparaissant comme discontinus.

7.5 Vignettes

Au lieu d'utiliser au complet l'image présentée et d'essayer de contrer la propriété de non-stationnarité, le logiciel `GUIAnalyser` pourrait permettre le nettoyage et la classification de plusieurs vignettes d'une même image, en d'autres mots de travailler sur de petites zones d'intérêt de l'image. Il faudrait alors modifier l'interface du logiciel pour permettre la sélection d'une vignette pour la nettoyer et ensuite la classifier. La figure 7.2 illustre le classement sur une vignette carrée de 64 pixels de côté.

7.6 Carte de nuages

Le filtre spatial passe-haut homomorphique de la section 4.1.2 a été créé à l'origine pour retirer les nuages des images urbaines prises par le satellite Terra. Les filtres spectraux classiques calibrés pour ne retenir que les basses fréquences donnent des résultats similaires. La différence entre l'image originale et l'image filtrée pourrait être utilisée pour dresser une carte de nuage pour la journée en



(a) (b)

FIG. 7.2 – Classification d'une vignette carrée de 64 pixels de côté, (a) la vignette et (b) la classification.

question.

7.7 `GUIAnalyser`

Le manuel d'utilisation du logiciel `GUIAnalyser` à la fin de l'annexe II liste des améliorations plus immédiates pour le logiciel lui-même.

7.8 `jai-operators`

Plusieurs opérateurs `JAI` font encore partie du code propre à ce projet. Certains de ces opérateurs pourraient être déplacés dans le projet `jai-operators`. Ce projet pourrait aussi être continué en augmentant le nombre d'opérateurs, mais cette augmentation est hors de la portée de ce projet.

CONCLUSION

Ce travail de maîtrise a présenté des travaux faits dans le cadre d'une recherche dans le domaine de la segmentation et de la classification du contenu d'images satellite de l'océan Arctique dans le but de reconnaître, de quantifier les chenaux créés par le mouvement de l'océan et des vents polaires et enfin de pouvoir quantifier le réchauffement global de la planète.

Plusieurs filtres passe-hauts ont été réalisés et testés pour d'abord nettoyer les images et ainsi enlever les artéfacts indésirables comme les nuages et rehausser les chenaux. Plusieurs de ces filtres sont des filtres classiques et bien connus mais d'autres ont été créés pour exploiter le fait que les images font partie d'une séquence.

De tous les filtres de nettoyage, la palme va au filtre passe-haut gaussien en trois dimensions. Cependant les autres filtres spectraux classiques performent bien, toutes les versions produisant des images dans lesquelles les contours sont bien rehaussés et les chenaux très bien visibles. En dernières positions après ces filtres spectraux sont les filtres passe-haut spatial, identité et la médiane temporelle.

D'autres filtres ont été réalisés et testés pour segmenter les images. La pauvre performance de la version originale du filtre des k-moyennes est due à la non-stationnarité de l'image mais la version contextuelle performe un peu mieux. Ce travail n'a pas réussi à segmenter hors de tout doute les chenaux à cause de leur épaisseur et intensité variables, caractéristiques très difficiles à reconnaître.

Finalement ce travail laisse comme héritage un procédé pour extraire et préparer les images pour leur analyse. Ce procédé utilise un script d'extraction et un programme pour la préparation. Il laisse aussi un programme facilement extensible, le logiciel `GUIAnalyser`, et des utilitaires comme les programmes `QuickHDFViewer` et `ClusterViewer` qui permettent de nettoyer et d'identifier les chenaux. En dernier lieu, il laisse aussi une bibliothèque de code *Open Source*, la bibliothèque `jai-operators`, qui étend les capacités de la bibliothèque officielle d'extension `JAI`.

BIBLIOGRAPHIE

[1] *Aqua Project Science*, http://aqua.nasa.gov/ le 29 novembre 2007.

[2] *NASA's Earth Observing System*, http://eospso.gsfc.nasa.gov/ le 29 novembre 2007.

[3] *A-Train Constellation*, http://www-calipso.larc.nasa.gov/about/atrain.php le 29 novembre 2007.

[4] *What is EOSDIS ?*, http://spsosun.gsfc.nasa.gov/eosinfo/EOSDIS_Site/index.html le 29 novembre 2007.

[5] *Data Products*, http://www.ghcc.msfc.nasa.gov/AMSR/data_products.html le 20 novembre 2007.

[6] *Instrument Description*, http://www.ghcc.msfc.nasa.gov/AMSR/instrument_descrip.html le 29 novembre 2007.

[7] *Why HDF ?*, http://hdf.ncsa.uiuc.edu/why_hdf/index.html le 29 novembre 2007

[8] *HDP – the HDF dumper*, http://hdf.ncsa.uiuc.edu/why_hdf/index.html le 29 novembre 2007

[9] texititHDF5 File Format Specification Version 2.0, http://hdf.ncsa.uiuc.edu/HDF5/doc/H5.format.html le 17 juin 2008

[10] *An improved method for cloud removal in ASTER data change detection* Feng Chun; Ma Jian-wen; Dai Qin; Chen Xue Geoscience and Remote Sensing Symposium, 2004. IGARSS apos;04. Proceedings. 2004 IEEE International Digital Object Identifier 10.1109/IGARSS.2004.1370431

[11] *Arctic sea ice, cloud, water, and lead classification using neural networks and 1.6-μm data* McIntire, T.J; Simpson, J.J. IEEE Transactions on Geoscience and Remote Sensing, 2002. IGARSS apos;04. Proceedings. 2004 IEEE International Digital Object Identifier 10.1109/TGRS.2002.803728

[12] *Enhancement of Low-Contrast Curvilinear Feature in Imagery* Mark J. Carlotto. IEEE Transactions on Image Processing, Vol. 16, No. 1, January 2007 Digital Object Identifier 10.1109/TIP.2006.884949

[13] *Road Extraction From Aerial Images Using a Region Competing Algorithm* Miriam Amo, Fernando Martínez and Margarita Torre IEEE Transactions on Image Processing, Vol. 15, No. 5, May 2006 Digital Object Identifier 10.1109/TIP.2005.864232

[14] *Lidar detection of leads in Arctic sea ice* R. C. Schnell, R. G. Barry, M. W. Miles, E. L. Andreas, L. F. Radke, C. A. Brock, M.P. McCormick et J.L. Moore, Letters to Nature, Volume 339, 1989.

[15] *Analysis of Beaufort Sea Ice Leads Using Airborne Infrared Remote Sensing* Sam Dorsi, Bruno Tremblay, Chris Zappa et Andrew Jessup

[16] *Surface Temperature and Lead Heat Fluxes from Aircraft Surveys During LEADEX* R. W. Lindsay, A. T. Jessup et J. A. Francis, Polar Science Center, Applied Physics Laboratory University of Washington, Seattle, Washington

[17] *Determination of total strain from faulting using slip measurements* C.H. Scholz et Patience A. Cowie, Letters to Nature, Volume 346, 1990.

[18] *Least squares quantization in PCM* S. P. Lloyd IEEE Trans. Inf. Theory, vol. IT-28, no. 2, pp. 129-136, Mar. 1982.

[19] *Digital Image Processing, second edition* Rafael C. Gonzales; Richard E. Woods Prentice Hall, 2002

[20] *Sun-synchronous orbit*, `http://en.wikipedia.org/wiki/Sun-synchronous_orbit` le 6 décembre 2007

[21] *LIDAR*, `http://fr.wikipedia.org/wiki/LIDAR` le 25 février 2008

[22] *Information Retrieval*, `http://en.wikipedia.org/wiki/Information_Retrieval` le 18 décembre 2008

[23] *Precision and recall*, `http://en.wikipedia.org/wiki/Precision_and_recall` le 18 décembre 2008

[24] *Tutoriel : Notions fondamentales de télédétection Plates-formes et capteurs - Caractéristiques d'un satellite : l'orbite et sa fauchée* http://www.ccrs.nrcan.gc.ca/resource/tutor/fundam/chapter2/02_f.php le 6 février 2008

[25] *Fengyun*, http://en.wikipedia.org/wiki/Fengyun le 19 février 2008.

[26] *2007 Chinese anti-satellite missile test*, http://en.wikipedia.org/wiki/2007_Chinese_anti-satellite_missile_test le 19 février 2008.

[27] *Java Media APIs*, http://java.sun.com/javase/technologies/desktop/media/ le 31 janvier 2008

[28] *Image Analysis*, http://java.sun.com/products/java-media/jai/forDevelopèrs/jai1_0_1guide-unc/Analysis.doc.html le 14 décembre 2008

[29] *jai-imageio-core : jai-imageio-core Project Page*, https://jai-imageio-core.dev.java.net/ le 31 janvier 2008

[30] *JGoodies : : Java User Interface Design*, http://www.jgoodies.com/index.html le 30 janvier 2008

[31] *jgoodies : Home*, https://jgoodies.dev.java.net/ le 30 janvier 2008

[32] *binding : Home*, https://binding.dev.java.net/ le 30 janvier 2008

[33] *forms : Home*, https://forms.dev.java.net/ le 30 janvier 2008

[34] *looks : Home*, https://looks.dev.java.net/ le 30 janvier 2008

[35] *Welcome to The New JUnit.org ! — JUnit.org*, http://www.junit.org/ le 30 janvier 2008

[36] *Test unitaire*, http://fr.wikipedia.org/wiki/Test_unitaire le 30 janvier 2008

[37] *forklabs : Forklabs Project Home Page*, https://forklabs.dev.java.net/ le 31 janvier 2008

[38] *baselib : Forklabs Baselib Project Home Page*, https://baselib.dev.java.net/ le 31 janvier 2008

[39] *hdr-codec : Forklabs HDR Codec Project Home Page*, `https://hdr-codec.dev.java.net/` le 31 janvier 2008

[40] *High Dynamic Range Imaging*, `http://en.wikipedia.org/wiki/High_dynamic_range_imaging` le 31 janvier 2008

Annexe I

Le script d'extraction

```
#!/usr/bin/bash

#
# Go into the data directory, change if the directory is elsewhere
#
source_dir="../data/89Ghz"
cd $source_dir

sink_dir="./fullsize"
mkdir -p $sink_dir

#
# Extract the H and V image from each HDF
#
for hdf in `ls ./hdf/*.hdf`; do
    day=$hdf
    day=${day#*AMSR_E_L3_SeaIce6km_B06_}
    day=${day%.*}

    h=SI_06km_NH_89H_DAY
    v=SI_06km_NH_89V_DAY

    sink_h=$sink_dir/$day-$h.bin
    sink_v=$sink_dir/$day-$v.bin

    echo hdp dumpsds -n $h -o $sink_h -b $hdf
    hdp dumpsds -n $h -o $sink_h -b $hdf

    echo hdp dumpsds -n $v -o $sink_v -b $hdf
    hdp dumpsds -n $v -o $sink_v -b $hdf
done
```

Listing I.1 – Le script pour l'extraction des images.

Annexe II

Manuel du logiciel `GUIAnalyser`

Cette annexe décrit l'utilisation du logiciel `GUIAnalyser`, le logiciel utilisé pour nettoyer des images ainsi que pour classifier les images nettoyées. Pour la description de l'architecture, il faut lire l'annexe III.

II.1 Démarrage

Il est très facile de démarrer le logiciel `GUIAnalyser`. Quand une ligne de commande est disponible, il ne suffit de faire la commande illustrée par le listing II.1. Si un méchanisme de *double-click* existe, comme sur les diverses versions de Windows, il suffit alors de l'utiliser sur le fichier `diro-arctic-ice-gui.analyser-0.0-M0.jar`.

Il est à noter que le logiciel `GUIAnalyser` charge automatiquement les images du répertoire `data/89Ghz/crop2` (voir l'annexe IV pour le contenu de la distribution), la séquence d'images utilisées pour générer les résultats. Il est tout à fait possible de charger de nouvelles images en suivant les instructions de la section II.3.1.

II.2 Les sections de l'interface graphique

En plus du menus, l'interface graphique du logiciel `GUIAnalyser` comprend deux sections principales, la section affichant les diverses images, à gauche, et la section affichant les panneaux de contrôle, à droite. La figure II.1 illustre l'application `GUIAnalyser`.

La section affichant les images les présente sous différents onglets, chacun ayant un libellé décrivant l'image ainsi qu'une infobulle (*tooltip*) optionnelle offrant plus

```
$ java -jar diro-arctic-ice-gui.analyser-0.0-M0.jar
```

Listing II.1 – Commande pour lancer l'application `GUIAnalyser` par la ligne de commande.

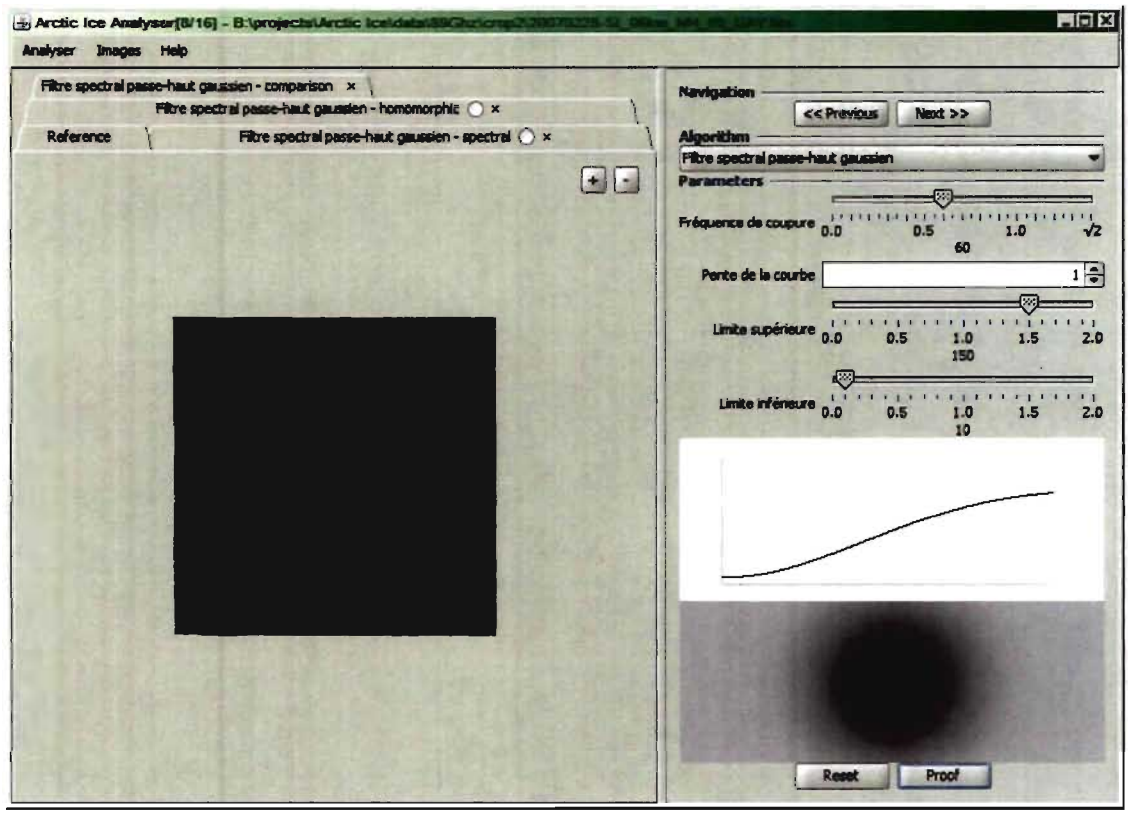


FIG. II.1 – L'application `GUIAnalyser`.

d'information, entre autres les paramètres ayant servi à la création d'une image.

Chacun des onglets, autre que le premier qui représente l'image de référence, peut être fermé en cliquant sur le x à la droite de l'onglet.

Certains onglets ont un bouton radio. Ces boutons ne servent à rien dans la version du logiciel accompagnant ce mémoire et peuvent être ignorés. Le but futur est de choisir une des images nettoyées afin de pouvoir l'associer à son image de référence quand il sera possible de sauvegarder une séquence.

La section affichant le panneau de contrôle est elle-même divisée en quatre parties : la navigation, les algorithmes, les paramètres et les actions.

La première partie est la partie de navigation et comprend les deux boutons *Previous* et *Next*. Ces deux boutons permettent de naviguer dans la séquence d'images,

allant respectivement à l'image précédente ou l'image suivante. Quand il n'est pas possible d'aller à une de ces images, le bouton est désactivé. La position de l'image dans la séquence ainsi que le nom de son fichier sont affichés dans le titre du cadre de l'application.

La seconde partie est la partie des algorithmes et consiste en la boîte combinée (*combo-box*) contenant la liste des algorithmes disponibles. Lorsqu'un algorithme est choisi, la section des paramètres affiche les paramètres propres à l'algorithme choisi.

La troisième partie est le panneau affichant les divers contrôles propres à l'algorithme choisi.

La quatrième partie comprend les deux boutons *Reset* et *Proof*. Le bouton *Reset* enlève tous les onglets, sauf l'image de référence, de la section affichant les images. Le bouton *Proof* quant à lui recueille les paramètres et l'algorithme et fait apparaître dans la section affichant les images le ou les onglets représentant les nouvelles images que l'algorithme aura créées.

II.3 Opérations importantes

Cette section montre comment performer les actions les plus importantes du logiciel `GUIAnalyser` : le chargement des images, la sauvegarde d'une image, le nettoyage, la classification et l'appel au logiciel `ClusterViewer`. Une dernière sous-section montre comment voir la transformée de Fourier d'une image.

II.3.1 Chargement d'images

Le menu *Analyser* contient un item *Load Images...* Cet item fait apparaître la fenêtre de dialogue pour le chargement des images, fenêtre illustrée à la figure II.2.

La fenêtre permet la sélection d'une ou plusieurs images dans sa partie gauche. Pour que les images sélectionnées soient prise en compte, il faut les transférer dans la partie en utilisant le bouton encerclé de bleu. Les images peuvent être transférées une par une ou par groupe, autant de fois que nécessaire et n'ont pas à être dans

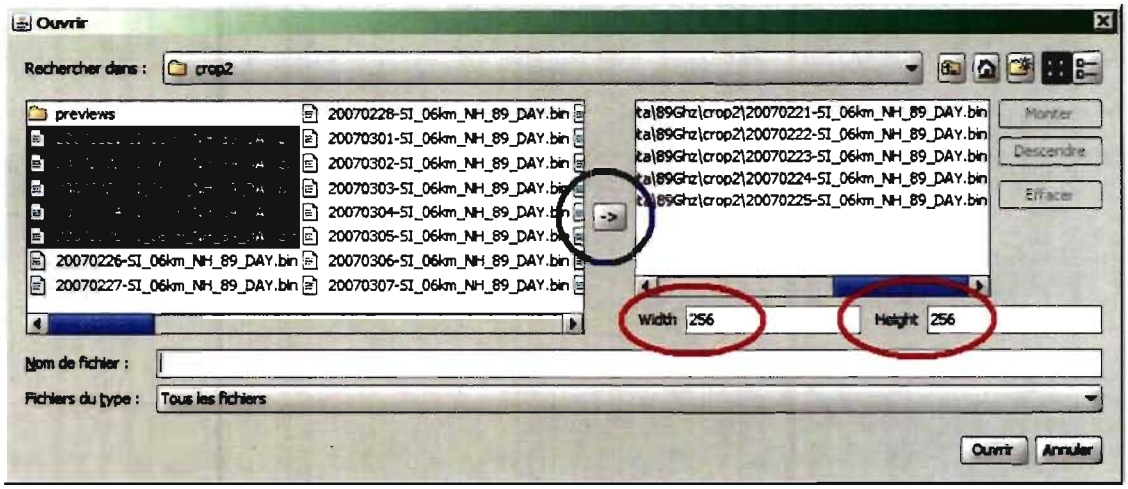


FIG. II.2 – La fenêtre de dialogue pour le chargement des image de l'application `GUIAnalyser`.

le même répertoire, elles peuvent être n'importe où.

L'ordre des images peut être ajusté en utilisant les boutons *Monter* et *Descendre*, une image à la fois. Les images indésirables peuvent être enlevées de la liste en les sélectionnant et en pressant le bouton *Effacer*.

Les images doivent toutes être de mêmes dimensions et celles-ci doivent être entrées dans les champs de texte encerclés de rouge.

Une fois les images choisies et les dimensions indiquées, il ne suffit que d'appuyer sur le bouton *Ouvrir*. Une note cependant, cette fenêtre étend la classe `JFileChooser`[1] et doit avoir au moins un fichier apparaissant dans la zone de texte *Nom de fichier* pour que le bouton *Ouvrir* fonctionne.

II.3.2 Sauvegarde d'une image

Toutes les images affichées dans la section affichant les images peuvent être sauvegardées. Pour sauvegarder ces images, il ne suffit que d'utiliser le menu contextuel et choisir l'option *Save Image...*. L'image peut être sauvegarder dans tous les for-

[1] `javax.swing.JFileChooser`

mats connus de la bibliothèque `JAI`, entre autre PNG, JPEG et TIFF en donnant au nom du fichier l'extension appropriée.

II.3.3 Nettoyage d'une image

Pour appliquer un filtre de nettoyage sur une image, il faut utiliser le panneau de contrôle. L'algorithme est choisi dans la boîte combinée et ses paramètres sont sélectionnés parmi les contrôles offerts.

Une fois les paramètres choisis, il suffit d'appuyer sur le bouton *Proof* pour afficher la ou les images nettoyées.

II.3.4 Classification d'une image nettoyée

Pour faire la classification des pixels d'une image nettoyée, il faut utiliser le menu contextuel. Les algorithmes de classification disponibles seront alors visibles avec leurs paramètres. Une fois un algorithme et ses paramètres choisis, un nouvel onglet pour cette image sera visible.

II.3.5 Appel au logiciel `ClusterViewer`

Toutes les images classifiées ont un item dans leur menu contextuel qui permet d'ouvrir cette image dans le logiciel `ClusterViewer`. L'interaction avec cette image se fait comme décrit à la section 6.5.

II.3.6 Transformée de Fourier

Dans le menu contextuel des images originales (les images dont l'onglet ne peut être fermé) et des images nettoyées il y a un item qui permet de voir le module centré de la transformée de Fourier de cette image.

II.4 Travail futur

Le logiciel `GUIAnalyser` est loin d'être terminé. Plusieurs améliorations le rendraient encore plus performant.

Les boutons radios des onglets ne servent présentement à rien. Ils devaient servir pour la réalisation de la commande *Save Sequence...*, une commande qui permet de sauvegarder le travail déjà fait, c'est-à-dire de sauvegarder l'algorithme de nettoyage (et ses paramètres) ainsi que l'algorithme de classification (et ses paramètres) afin de pouvoir reproduire les résultats sans intervention humaine.

Aussi, plusieurs composantes, mais pas toutes, ont une version française et une version anglaise de leurs libellés. Les composantes monolingues devraient être retouchées pour pouvoir être facilement traduites.

Une autre amélioration serait d'être capable de visualiser rapidement l'histogramme d'une image. Cet histogramme pourrait aider à déterminer le nombre de classes pour certains algorithmes de classification.

Une dernière autre retouche serait de rendre la sauvegarde d'une image plus robuste. Pour le moment, l'extension du nom de l'image détermine le format de celle-ci. Le format devrait être déterminé en utilisant la section des filtres de la fenêtre de sauvegarde.

Annexe III

Architecture du logiciel `GUIAnalyser`

Cette annexe décrit l'architecture interne de l'application `GUIAnalyser`, entre autres comment ajouter de nouveaux filtres de nettoyage et de nouveaux filtres de classification. La discussion suppose que le lecteur est familier avec l'annexe II qui explique l'utilisation du logiciel `GUIAnalyser`, qu'il possède un accès au code source et aussi qu'il ait une bonne connaissance en dévelopement de logiciels.

III.1 Architecture des filtres de nettoyage

L'architecture des filtres de nettoyage est assez simple, bien que beaucoup de travail soit nécessaire pour en ajouter un nouveau. L'ajout de nouveaux filtres commence avec la classe `Algorithm`[1] et celle-ci est supportée par de nombreuses autres classes et bibliothèques externes.

III.1.1 La classe `Algorithm`

Pour ajouter un nouveau filtre de nettoyage, il faut commencer par regarder la classe abstraite `Algorithm`. Cette classe contient cinq méthodes que la sous-classe représentant le nouveau filtre de nettoyage doit implémenter.

`Algorithm()` La classe `Algorithm` n'a qu'un seul constructeur prenant un nom en paramètre, le nom d'affichage de l'algorithme. Ce nom sera affiché dans la boîte de choix au-dessus de la section contenant l'interface responsable de la présentation des paramètres.

`buildEntryPanel()` Cette méthode est responsable de la construction de l'interface graphique qui sera présentée à l'utilisateur. Cette méthode est appelée une seule fois au début pour la construction et l'interface construite sera réutilisée, ajustée par des appels successifs à la méthode `adjustTo()`.

[1] `c.u.i.i.a.analyser.algorithm.Algorithm`

`getParameterBlock()` Les paramètres de chacune des images nettoyées sont retenus par l'application `GUIAnalyser`. La responsabilité de fournir ces paramètres revient à cette méthode. Lorsque vient le temps d'ajuster l'interface, le `ParameterBlock` retourné par cette méthode sera alors passé à la méthode `adjustTo()`. Il en va de même pour la méthode `invoke()` pour la génération de l'image nettoyée.

`adjustTo()` Cette méthode est responsable d'ajuster l'interface graphique avec les paramètres fournis par la méthode `getParameterBlock()`.

`getProofImages()` Les épreuves fournies lorsque le bouton *Épreuve* est pressé proviennent de cette méthode. Cette méthode retourne un tableau d'épreuves, ce qui permet à certains algorithmes de présenter diverses épreuves avec le même jeu de paramètres (i.e., les filtres de nettoyage spectraux présentent trois épreuves, une épreuve normale, une épreuve homomorphique et la différence).

`invoke()` Cette méthode est responsable de l'application de l'algorithme sur une ou des images. Les paramètres fournis sont ceux qui ont été retournés par la méthode `getParameterBlock()`. De plus, la ou les images à nettoyer sont incluses dans le bloc de paramètres.

De plus, la méthode `getSources()` est responsable de trouver les images sources. Par défaut, seulement l'image à l'index indiqué par le curseur est retournée, mais si l'algorithme a besoin de plusieurs images, c'est à cet endroit qu'elles sont choisies (voir la section 6.6.2).

Finalement, la nouvelle classe représentant cet algorithme doit être enregistrée. Cet enregistrement se fait dans la classe `Algorithm`, dans le bloc `<cinit>`, le code exécuté lors du chargement de la classe. Une meilleure technique d'enregistrement consisterait à lire un fichier externe contenant seulement le nom des classes représentant les divers algorithmes. Ceci permettrait de pouvoir changer la liste sans avoir à recompiler.

III.1.2 La classe `ProofImage` et la classe `ProofImageTabbedPaneModel`

Les images présentées par le logiciel `GUIAnalyser` sont présentées par la composante graphique `JTabbedPane`[2]. Contrairement à plusieurs composantes graphiques, la composante `JTabbedPane` ne contient pas de modèle explicite. Un modèle explicite a été créé, l'interface `TabbedPaneModel`[3] et une implémentation simple, la classe `AbstractTabbedPaneModel`[4] et sa méthode de classe `createTabbedPane()`.

Le modèle `ProofImageTabbedPaneModel`[5] est la spécialisation du modèle pour les besoins de l'application `GUIAnalyser`. Ce modèle spécialisé utilise des instances de la classe `ProofImage`[6] afin d'afficher les diverses images nettoyées et classifiées ainsi que le texte et la décoration de chacun des *tabs*.

III.1.3 La classe `DifferenceAlgorithm`

Les filtres spectraux déjà réalisés (idéal, gaussien et de Butterworth) génèrent deux épreuves, une épreuve pour la version normale du filtre et une épreuve pour la version homomorphique. Ce filtres fournissent aussi une image de la différence entre ces deux versions, c'est-à-dire une image de la distance L1 entre ces deux versions.

La création de cette image de différence est la responsabilité de la classe utilitaire `DifferenceAlgorithm`[7]. Cet algorithme est une implémentation vide de la classe abstraite `Algorithm`, sauf pour la méthode `invoke()`, qui prend un bloc de paramètre contenant les deux images à comparer.

---

[2] `javax.swing.JTabbedPane`
[3] `c.u.i.i.a.analyser.swing.TabbedPaneModel`
[4] `c.u.i.i.a.analyser.swing.AbstractTabbedPaneModel`
[5] `c.u.i.i.a.analyser.ProofImageTabbedPaneModel`
[6] `c.u.i.i.a.analyser.ProofImage`
[7] `c.u.i.i.a.analyser.algorithm.DifferenceAlgorithm`

III.1.4 La classe `FrequencyGraphCanvas`, la classe `FrequencyGraphImage` et la classe `BoundedImageFunctionModel`

Les coupes et les images des fonctions des filtres spectraux sont affichées à l'aide des composantes graphiques `FrequencyGraphCanvas`[8] pour la coupe de la fonction et `FrequencyGraphImage`[9] pour son image. Le modèle pour ces deux composantes graphiques est la classe `FilterImageFunction`[10].

Pour faciliter la synchronization entre les composantes graphiques, le modèle et la représentation fournis par la coupe et l'image de la fonction, il est recommandé d'utiliser la classe d'enrobage `BoundedImageFunctionModel`[11] et les concepts de la bibliothèque *JGoodies Binding*. Les classes `BoundedImageFunctionEntryPanel`[12] et `AbstractEntryPanel`[13] utilisent ces concepts.

III.1.5 Les bibliothèques *JGoodies Forms* et *JGoodies Binding*

Les bibliothèques *JGoodies Forms* et *JGoodies Binding* sont deux bibliothèques de code externes servant surtout aux applications avec interface graphique, tel qu'expliqué à la section 6.1.2. Elles sont d'autant plus utiles ici pour la création de l'interface (la méthode `buildEntryPanel()` et la bibliothèque *JGoodies Forms*) et sa synchronization avec le bloc de paramètres (les méthodes `getParameterBlock()` et `adjustTo()` et la bibliothèque *JGoodies Binding*).

III.2 Architecture des filtres de classification

L'architecture des filtres de classification est beaucoup plus simple que celle des filtres de nettoyage. Il y a dans la classe `GuiAnalyser`[14] une méthode nommée `getClassifyingMenuItems()`. Cette methode est responsable de la création des

[8] `c.u.i.i.a.analyser.swing.FrequencyGraphCanvas`
[9] `c.u.i.i.a.analyser.swing.FrequencyGraphImage`
[10] `c.u.i.i.a.media.jai.function.FilterImageFunction`
[11] `c.u.i.i.a.analyser.algorithm.BoundedImageFunctionModel`
[12] `c.u.i.i.a.analyser.algorithm.BoundedImageFunctionEntryPanel`
[13] `c.u.i.i.a.analyser.algorithm.AbstractEntryPanel`
[14] `c.u.i.i.a.laucher.GuiAnalyser`

items du menu contextuel des images nettoyées. Pour ajouter un nouveau filtre de classification, il suffit alors d'ajouter dans le tableau retourné l'item de menu qui activera le code. Le code des méthodes incluses dans la classe `GuiAnalyser`, `getNaiveKMeansMenu()` et `getVeryRobustKMeansMenu()`, peut servir de base.

En fait, cette architecture est très ad-hoc. Elle fonctionne bien puisque seulement deux filtres de nettoyage sont présents au moment de l'écriture de ce mémoire. Lors de l'ajout du prochain filtre de classification, une nouvelle architecture similaire à celle des filtres de nettoyage devrait être faite et appliquée.

Annexe IV

La distribution du logiciel `GUIAnalyser`

Cette annexe décrit la distribution contenant les logiciels. Cette distribution est une distribution binaire bien qu'elle comprenne le code source des programmes et de ce mémoire. Les instructions pour compiler les logiciels sont comprises dans les fichiers *Ant*[1] `build.xml` et `build.properties`. Les fichiers `.project` et `.classpath` sont les fichiers de configuration de l'environnement de développement intégré *Eclipse*[2].

La distribution comprend les répertoires suivants :

bin Ce répertoire comprend le script pour extraire les images des fichiers HDF, c'est le même script qu'illustré à l'annexe I.

data Ce répertoire comprend dans ses répertoires toutes les images utilisées. Les images originales sont dans le répertoire `hdf` tandis que les images extraites sont dans le répertoire `fullsize`. Le répertoire `256x256` contient les images de taille 256 pixels par 256 pixels et le répertoire `512x512` contient les images de taille 512 pixels par 512 pixels. Le répertoire `crop2` contient un sous-ensemble des images de taille 256 pixels par 256 pixels, celles utilisées pour produire les résultats du chapitre 5.

docs Ce répertoire comprend la génération de la documentation *JavaDoc* ainsi qu'une copie électronique des principaux articles cités dans ce mémoire.

lib Ce répertoire comprend tous les fichiers qui correspondent à toutes les bibliothèques externes utilisées dans la réalisation des logiciels.

reports Ce répertoire comprend le rapport de l'outil JUnit, le rapport des tests unitaires.

---

[1] `http://ant.apache.org/`
[2] `http://www.eclipse.org/`

```
$ java -jar <fichier JAR> [paramètres]
```

Listing IV.1 – Commande générale pour lancer une application par la ligne de commande.

save Ce répertoire comprend les images présentées dans le chapitre des résultats. Il comprend aussi l'image par défaut de l'application `ClusterViewer` et d'autres images de test.

src Ce répertoire comprend le code source de ce projet, le code source des filtres, le code des applications et le code de test et d'expérimentation. Le code du projet `jai-operators` est disponible sur le site web du projet lui-même[3]. Il comprend aussi les fichiers LaTeX originaux de ce mémoire.

La distribution comprend aussi trois fichiers pour le démarrage des diverses applications.

diro-arctic-ice-gui.analyser-0.0-M0.jar Cette archive contient les instructions pour le démarrage de l'application `GUIAnalyser`.

diro-arctic-ice-cluster.viewer-0.0-M0.jar Cette archive contient les instructions pour le démarrage l'application `ClusterViewer`.

diro-arctic-ice-quick.hdf.viewer-0.0-M0.jar Cette archive contient les instructions pour le démarrage l'application `QuickHDFViewer`.

Pour démarrer ces applications, il suffit de faire, sous les diverses versions de Windows, un *double-click* sur le fichier correspondant. Pour les autres systèmes d'exploitation, ou quand une ligne de commande est disponible, il suffit de faire la commande illustrée au listing IV.1.

Les autres fichiers sont les fichiers de license pour les bibliothèques externes et les fichiers nécessaires pour reconstruire les logiciels.

[3] `http:jaioperators.dev.java.net/`

Annexe V

La bibliothèque `jai-operators`

Cette annexe décrit chacun des treize opérateurs présents dans la bibliothèque `jai-operators` au moment de l'écriture de ce mémoire.

V.1 La classe `CollectionDescriptor`

La bibliothèque `JAI` contient une interface, `OperationDescriptor`[1], dont les responsabilités, entre autres, sont de décrire un opérateur ainsi que de valider les images sources et les arguments qui seront présentés à l'implémentation. Les quelques opérateurs originaux de la bibliothèque `JAI` qui acceptent plusieurs images sources n'acceptent qu'un seul item comme source, une collection d'images (des objets de type `Collection`[2]). En revanche, la classe des objets de paramètres, la classe `ParameterBlock`[3], permet d'entrer plusieurs items comme source.

La classe `CollectionDescriptor`[4] offre des facilités pour aider à valider ces items sources lorsque l'opérateur peut en accepter plusieurs. Les items sources peuvent alors être présentés comme un amalgame d'items singuliers, de tableaux ou de collections. Tous ces items seront alors mis, dans l'ordre, dans une seule et même collection qui sera présentée à l'opérateur. Les instructions aux listings V.1, V.2 et V.3 sont alors équivalentes lorsque présentées à un opérateur dont le descripteur est de type `CollectionDescriptor`.

V.2 Opérateur `applytocollection`

L'opérateur `applytocollection` est un opérateur qui applique à une suite d'images une même opération. Il permet de ne définir qu'un seul bloc de paramètres

---

[1] `javax.media.jai.OperationDescriptor`
[2] `java.util.Collection`
[3] `java.awt.image.renderable.ParameterBlock`
[4] `ca.forklabs.media.jai.CollectionDescriptor`

```
RenderedImage source1, source2, source3, source4, source5;

Collection sources<RenderedImage> = new LinkedList<RenderedImage>();
sources.add(source1);
sources.add(source2);
sources.add(source3);
sources.add(source4);
sources.add(source5);

ParameterBlock pb = new ParameterBlock().addSource(sources);
```

Listing V.1 – Présentation d'une collection d'images comme source à un opérateur dont le descripteur est de la classe `CollectionDescriptor`.

```
RenderedImage source1, source2, source3, source4, source5;

ParameterBlock pb = new ParameterBlock().addSource(source1)
                                        .addSource(source2)
                                        .addSource(source3)
                                        .addSource(source4)
                                        .addSource(source5);
```

Listing V.2 – Présentation d'images séparées comme source à un opérateur dont le descripteur est de la classe `CollectionDescriptor`.

```
RenderedImage source1, source2, source3, source4, source5;

Collection<RenderedImage> sources1 = new LinkedList<RenderedImage>();
sources.add(source1);
sources.add(source2);

RenderedImage[] sources2 = new RenderedImage[] {
   source4, source5,
   };

ParameterBlock pb = new ParameterBlock().addSource(sources1)
                                        .addSource(source3)
                                        .addSource(sources2);
```

Listing V.3 – Présentation des images sources à un opérateur dont le descripteur est de la classe `CollectionDescriptor` à l'aide d'un amalgame d'images séparées, de collection d'images et de tableaux.

```
String operator = ApplyToCollectionDescriptor.NAME;

Collection<RenderedImage> sources = Arrays.asList(image1, image2, ...);
String name = "addconst";
ParameterBlock add_const_pb = new ParameterBlock().add(new double[] { 1.0, });

ParameterBlock pb = new ParameterBlock().addSource(sources)
                                        .add(name)
                                        .add(add_const_pb);

Collection<RenderedImage> sinks = JAI.createCollection(operator, pb);
```

Listing V.4 – Utilisation de l'opérateur `applytocollection` pour ajouter une même constante à toute une suite d'images.

et d'abstraire le concept de boucle. Le listing V.4 illustre l'utilisation de l'opérateur `applytocollection`.

V.3 Opérateur `autorescale`

La bibliothèque `JAI` fournit un opérateur pour appliquer une transformation linéaire à la valeur de chacun des pixels d'une image, l'opérateur `rescale`. L'utilisation la plus fréquente de ce puissant opérateur est le recalage d'une image pour la présentation.

Pour recaler une image en utilisant l'opérateur `rescale`, il est nécessaire de connaître la valeur minimale et maximale des pixels afin de calculer le multiplicateur et la constante additive à appliquer. Par exemple, le code du listing V.5 montre comment recaler une image entre 0 et 255.

Ce calcul répétitif et sujet à l'erreur a donné naissance à une façade, l'opérateur `autorescale` dont l'utilisation est illustrée par le listing V.6. À partir de valeurs minimales et maximales, cet opérateur applique automatiquement la bonne transformation linéaire pour y arriver. De plus, si une bande de l'image est constante, c'est-à-dire que le minimum et le maximum sont les mêmes, l'opérateur assigne la valeur milieu de l'intervalle désiré.

```
ParameterBlock pb = new ParameterBlock()
                        .addSource(source);
RenderedImage stats = JAI.create("extrema", pb);
double[] mins = (double[]) stats.getProperty("minimum");
double[] maxs = (double[]) stats.getProperty("maximum");

double[] constants = new double[bands];
double[] offsets = new double[bands];
for (int i = 0; i < bands; i++) {
   constants[i] = 255.0 / (maxs[i] - mins[i]);
   offsets[i] = (255.0 * mins[i]) / (mins[i] - maxs[i]);
   }
pb.add(constants)
  .add(offsets);

RenderedImage sink = JAI.create("rescale", pb);
```

Listing V.5 – Recalage d'une image entre 0 et 255.

```
double[] lower_bounds = new double[] { 0.0, };
double[] upper_bounds = new double[] { 255.0, };
ParameterBlock pb = new ParameterBlock()
                        .addSource(source)
                        .add(lower_bounds)
                        .add(upper_bounds);
RenderedImage sink = JAI.create("autorescale", pb);
```

Listing V.6 – Recalage automatique d'une image entre 0 et 255.

V.4 Opérateur `dft`

La librairie `JAI` possède déjà un opérateur pour la transformée de Fourier, l'opérateur `dft`. Cet opérateur produit la transformée de Fourier sur une seule image, comme illustré par le listing V.7.

Plusieurs filtres de nettoyage expliqués au chapitre 4 requièrent une transformée de Fourier sur une suite d'images, une transformée de Fourier 3D. Un nouvel opérateur a été créé pour produire la transformée de Fourier sur une suite d'images tout en gardant le même nom. Cet opérateur s'utilise de façon identique à l'opérateur original, si ce n'est qu'il prend plusieurs images en entrée au lieu d'une seule, comme illustré par le listing V.8.

Comme l'opérateur de base, le nouvel opérateur ajoute des rangées, des colonnes et des images noires si les dimensions ne sont pas des puissances de 2.

```
ParameterBlock pb = new ParameterBlock()
                        .addSource(source)
RenderedImage sink = JAI.create("dft", pb);
```

Listing V.7 – Transformée de Fourier d'une seule image.

```
ParameterBlock pb = new ParameterBlock()
                        .addSource(source1)
                        .addSource(source2)
// ...
                        .addSource(sourcen);
Collection<RenderedImage> sinks = JAI.createCollection("dft", pb);
```

Listing V.8 – Transformée de Fourier de plusieurs image.

V.5 Opérateur `idft`

La librairie `JAI` possède déjà un opérateur pour la transformée de Fourier inverse, l'opérateur `idft`. Cet opérateur produit la transformée de Fourier inverse sur une seule image, comme illustré par le listing V.9.

L'extension de l'opérateur `dft` pour transformer une suite d'images a entraîné l'extension en 3D de l'opérateur `JAI idft`. Cette extension produit la transformée de Fourier inverse sur une suite d'images tout en gardant le même nom. Cet opérateur s'utilise de façon identique à l'opérateur original, si ce n'est qu'il prend plusieurs images en entrée au lieu d'une seule, comme illustré par le listing V.10.

Comme l'opérateur de base, le nouvel opérateur ajoute des rangées, des colonnes et des images noires si les dimensions ne sont pas des puissances de 2.

V.6 Opérateur `imagefunction`

La bibliothèque `JAI` possède un opérateur qui permet de définir une image selon une fonction, l'opérateur `imagefunction`. Cet opérateur est très pratique pour créer une image filtre à multiplier à une transformée de Fourier.

```
ParameterBlock pb = new ParameterBlock()
                        .addSource(source)
RenderedImage sink = JAI.create("idft", pb);
```

Listing V.9 – Transformée de Fourier inverse d'une seule image.

```
ParameterBlock pb = new ParameterBlock()
                        .addSource(source1)
                        .addSource(source2)
// ...
                        .addSource(sourcen);
Collection<RenderedImage> sinks = JAI.createCollection("idft", pb);
```

Listing V.10 – Transformée de Fourier inverse de plusieurs images.

Les techniques de filtrage en utilisant la transformée de Fourier en 3D ont entraîné l'extension de l'opérateur `imagefunction` pour créer des images 3D à partir d'une fonction. Cette extension nécessite l'utilisation d'une nouvelle interface, l'interface `ImageFunction3D`[5].

Le listing V.11 illustre l'utilisation de l'opérateur pour la création d'une image en trois dimensions à deux bandes dont la valeur de chacun des pixels est la position du plan dans l'image cubique.

V.7 Opérateur `kmeans`

L'un des algorithmes de classification de la section 4.2 est l'algorithme des k-moyennes. Un opérateur `JAI`, l'opérateur `kmeans`, a été créé pour réaliser ce filtre et une utilisation simple est illustrée par le listing V.12.

Trois des quatre paramètres sont évidents, le premier indique le nombre d'amas (*cluster*) désirés, le troisième le nombre maximal d'itérations et le quatrième, la palette de couleurs (*color_map*) indique la couleur désirée pour chacun des amas.

Le paramètre le plus important est la fonction d'évaluation en deuxième position. Ce paramètre de type `EvaluationFunction`[6] prend en entrée la position du pixel à évaluer ainsi qu'un accès total à l'image sous forme de `RandomIter`[7], une classe offrant un accès direct en lecture à tous les pixels de son image. Une fois son évaluation terminée, la fonction d'évaluation retourne un vecteur qui sera ensuite comparé avec tous les autres vecteurs pour la classification.

[5] `ca.forklabs.media.jai.ImageFunction3D`
[6] `ca.forklabs.media.jai.operator.KMeansDescriptor$EvaluationFunction`
[7] `javax.media.jai.iterator.RandomIter`

```
ImageFunction3D function = new ImageFunction3D() {
   @Override public int getNumBands() { return 2; }
   @Override public boolean isComplex() { return false; }

   @Override
   public void getBand(int band, int slice, int dimX, int dimY, int dimZ,
                       float[] real, float[] imag) {
      for (int i = 0; i < dimX * dimY; i++) {
         real[i] = slice;
         }
      }

   @Override
   public void getBand(int band, int slice, int dimX, int dimY, int dimZ,
                       double[] real, double[] imag) {
      for (int i = 0; i < dimX * dimY; i++) {
         real[i] = slice;
         }
      }
   };

int width = 3;
int height = 3;
int depth = 3;

ParameterBlock pb = new ParameterBlock().add(function)
                                        .add(width)
                                        .add(height)
                                        .add(depth);

Collection<RenderedImage> sinks = JAI.createCollection("imagefunction", pb);
```

Listing V.11 – Exemple d'utilisation de l'opérateur `imagefunction` pour créer une image en trois dimensions.

```
int[][] pixels = new int[][] {
   { 1, 2, 3, 4, 5, 6, 7, 8, 9, },
   };
int cols = 3;
int rows = 3;
RenderedImage source = RasterAdapter.buildIntImage(pixels, cols, rows);

int clusters = 3;
KMeansDescriptor.EvaluationFunction function = null;
int iterations = 100;
int[][] color_map = new int[][] { { 2, }, { 4, }, { 6, }, };

ParameterBlock pb = new ParameterBlock().addSource(clusters)
                                        .add(function)
                                        .add(iterations)
                                        .add(color_map);

RenderedOp sink = JAI.create("kmeans", pb);
```

Listing V.12 – Exemple d'utilisation de l'opérateur `kmeans` pour la séparation en trois amas d'une image 3x3.

```
ParameterBlock pb = new ParameterBlock()
                        .addSource(source1)
                        .addSource(source2)
// ...
                        .addSource(sourcen);
RenderedImage sink = JAI.create("mediancollection", pb);
```

Listing V.13 – Utilisation de l'opérateur `mediancollection`.

La fonction d'évaluation par défaut retourne comme vecteur la couleur du pixel, mais n'importe quelle type d'évaluation est possible. D'ailleurs, la classe `RobustKMeansEvaluationFunction`[8] réalise la fonction d'évaluation contextuelle décrite à la section 4.2.1.2.

V.8 Opérateur `mediancollection`

Le filtre de nettoyage décrit à la section 4.1.6 est le filtre de la médiane temporelle. Une nouvelle image mediane est créée à partir d'une suite d'images, chacun des pixels de cette nouvelle image ayant la valeur médiane des pixels correspondants dans la suite d'images. Le listing V.13 illustre l'utilisation de l'opérateur `mediancollection`.

V.9 Opérateur `periodicshift`

La bibliothèque `JAI` possède un opérateur, qui permet de faire une translation périodique, l'opérateur `periodicshift`. Cet opérateur est très pratique pour centrer une transformée de Fourier, que ce soit pour sa représentation habituelle ou encore pour appliquer un filtre spectral centré. Le listing V.14 illustre l'utilisation de l'opérateur `periodicshift`.

Les techniques de filtrage en utilisant la transformée de Fourier en 3D ont entraîné l'extension de l'opérateur `periodicshift` pour appliquer une translation périodique à une suite d'images, comme illustré au listing V.15.

[8] ca.umontreal.iro.image.arcticice.analyser.RobustKMeansEvaluationFunction

```
ParameterBlock pb = new ParameterBlock()
                        .addSource(source)
                        .add(dx)
                        .add(dy);
RenderedImage shifted = JAI.create("periodicshift", pb);
```

Listing V.14 – Utilisation de l'opérateur `periodicshift` avec une seule image.

```
ParameterBlock pb = new ParameterBlock()
                        .addSource(source1)
                        .addSource(source2)
// ...
                        .addSource(sourcen)
                        .add(dx)
                        .add(dy)
                        .add(dz);
Collection<RenderedImage> sinks = JAI.createCollection("periodicshift", pb);
```

Listing V.15 – Utilisation de l'opérateur `periodicshift` avec plusieurs images.

V.10 Opérateur `pipeline`

Lors du traitement d'une image, il n'est pas rare que plusieurs opérateurs soient appliqués à une image, l'un à la suite de l'autre ; l'image de sortie d'un opérateur devenant l'image d'entrée d'un autre, comme illustré au listing V.16.

Pour faciliter ce pipeline de traitement, l'opérateur `pipeline` a été créé et son utilisation est illustrée par le listing V.17. Il est à noter qu'aucun des blocs de paramètres dans le tableau `parameters` n'a d'image source. Celle-ci sera ajoutée à ces blocs de paramètres durant l'opération.

V.11 Opérateur `precisionandrecall`

L'opérateur `precisionandrecall` permet de calculer la précision et le rappel (cf. section 5.1.1) entre une image source et une image vérité, comme illustré au listing V.18

```
RenderedImage image1 = JAI.create("operation1", pb1.addSource(source));
RenderedImage image2 = JAI.create("operation2", pb2.addSource(image1));
RenderedImage image3 = JAI.create("operation3", pb3.addSource(image2));
// ...
```

Listing V.16 – Une séquence d'opérations.

```
String[] operations = new String[] {
   "operation1",
   "operation2",
   "operation3",
// ...
   };
ParameterBlock[] parameters = new ParameterBlock[] {
   new ParameterBlock(),
   new ParameterBlock(),
   new ParameterBlock(),
// ...
   };
ParameterBlock pb = new ParameterBlock()
                        .addSource(source)
                        .add(operations)
                        .add(parameters);
RenderedImage sink = JAI.create("pipeline", pb);
```

Listing V.17 – Utilisation de l'operateur `pipeline`.

```
RenderedImage truth = ...
int[] t_color = ...

RenderedImage candidate = ...
int[] c_color = ...

ParameterBlock pb = new ParameterBlock()
                        .addSource(candidate)
                        .add(c_color)
                        .add(truth)
                        .add(t_color);

RenderedImage sink = JAI.create("precisionandrecall", pb);

double precision = ((double[]) sink.getProperty("precision"))[0];
double recall = ((double[]) sink.getProperty("recall"))[0];
```

Listing V.18 – Utilisation de l'operateur `precisionandrecall`.

L'image utilisée comme image source est l'image candidate, c'est-à-dire l'image pour laquelle la précision et le rappel sont calculés. Les paramètres principaux sont, dans l'ordre : - la couleur de la classe dans l'image candidate, - l'image vérité et - la couleur de la classe vérité.

Les couleurs sont représentées sous forme de tableau d'entiers, tableau de même longueur que le nombre de bande dans l'image associée.

L'opérateur `precisionandrecall` accepte aussi trois paramètres secondaires comme les autres opérateurs statistiques de JAI [28] : - une région d'intérêt sous la forme d'un objet de la classe `ROI`[9], - une fréquence horizontale d'échantillonage et - une fréquence verticale d'échantillonage. Quand ces paramètres ne sont pas présents, le calcul se fait sur l'image en entier.

V.12 Opérateur `spectralfilter`

L'opérateur `spectralfilter` permet de filtrer une image dans le domaine spectral en une seule opération.

Pour filtrer une image dans le domaine spectral, il faut d'abord faire la transformée de Fourier de celle-ci. Une fois la transformée effectuée, elle est centrée (le pixel représentant la moyenne de l'image est mis au centre de l'image). Cette image centrée est filtrée par une multiplication complexe avec l'image du filtre, elle aussi centrée. Une fois cette multiplication faite, le résultat est décentré et finalement la transformée inverse est effectuée pour donner le résultat.

Le pseudo-code de cette opération est illustré au listing V.19. Le nom des fonctions correspond à des opérateurs `JAI`, de la bibliothèque de base ou de la bibliothèque `jai-operators`.

Le listing V.20 illustre l'utilisation de l'opérateur `spectralfilter`. L'opérateur `spectralfilter` peut être utilisé pour les images en deux dimensions, le paramètre de l'image filtre étant de la classe `SpectralFilter2D`[10]. Pour les images en trois

[9] `javax.media.jai.ROI`
[10] `ca.forklabs.media.jai.SpectralFilter2D`

```
Image d'origine O.
Image de la transformée de Fouriere TF = dft(O)
Transformée centrée STF = periodicshift(TF)

Image filtre centrée FC.
Image filtré centrée SF = multiplycomplex(STF, FC)

Image filtré IF = periodicshift(SF)
Image filtré F = idft(IF)
```

Listing V.19 – Pseudo-code d'un filtrage dans le domaine spectral.

```
SpectralFilter2D filter = new SpectralFilter2D() {
   private int width = 0;
   private int height = 0;
   public void setWidth(int width) { this.width = width; }
   public void setHeight(int height) { this.height = height; }
   @SuppressWarnings("hiding")
   public RenderedImage getFilterImage(int elements, RenderingHints hints) {
      float width = this.width;
      float height = this.height;
      Number[] values = new Double[] { 1.0, 0.0, };
      ParameterBlock pb = new ParameterBlock().add(width)
                                              .add(height)
                                              .add(values);
      RenderedImage sink = JAI.create("constant", pb, hints);
      return sink;
      }
   };

ParameterBlock pb = new ParameterBlock().addSource(source)
                                        .add(filter);

RenderedImage sink = JAI.create("spectralfilter", pb);
```

Listing V.20 – Utilisation de l'opérateur `spectralfilter` pour une image en deux dimensions, l'image source est filtrée par un filtre identité.

dimensions, le type du paramètre est alors `SpectralFilter3D`[11].

V.13 Opérateur `spectralhomomorphic`

Un filtre spectral homomorphique est un filtre spectral dont l'image d'entrée a été mise dans le domaine logarithmique. Après le filtre, l'image est retournée dans son domaine par une exponentielle.

L'opérateur `spectralhomomorphic` adapte l'opérateur `spectralfilter` pour le transformer en filtre spectral homomorphique, tel qu'illustré par le listing V.21.

[11] `ca.forklabs.media.jai.SpectralFilter3D`

```
SpectralFilter3D filter = new SpectralFilter3D() {
   private int width = 0;
   private int height = 0;
   private int depth = 0;
   public void setWidth(int width) { this.width = width; }
   public void setHeight(int height) { this.height = height; }
   public void setDepth(int depth) { this.depth = depth; }
   @SuppressWarnings("hiding")
   public CollectionImage getFilterImage(int elements, RenderingHints hints) {
      float width = this.width;
      float height = this.height;
      float depth = this.depth;
      Number[] values = new Double[] { 1.0, 0.0, };
      ParameterBlock pb = new ParameterBlock().add(width)
                                              .add(height)
                                              .add(values);
      RenderedImage constant = JAI.create("constant", pb, hints);

      CollectionImage sink = new SimpleCollectionImage();
      for (int i = 0; i < depth; i++) {
         sink.add(constant);
         }
      return sink;
      }
   };

ParameterBlock pb = new ParameterBlock().addSource(source1)
                                        .addSource(source2)
// ...
                                        .addSource(sourcen)
                                        .add(filter);

Collection<RenderedImage> sinks = JAI.createCollection("spectralhomomorphic", pb);
```

Listing V.21 – Utilisation de l'opérateur `spectralhomomorphic` pour une image en trois dimensions, l'image source est filtrée par un filtre identité.

V.14 Opérateur `unaryfunction`

Plusieurs opérateurs existent déjà pour appliquer une même fonction sur chacun des pixels d'une image, les opérateurs `log` et `exp` en sont des exemples[12]. Par contre, il se peut qu'un utilisateur veuille appliquer une fonction arbitraire, comme la racine carrée[13].

L'opérateur `unaryfunction` répond à ce besoin. Avec cet opérateur, il est possible d'appliquer une fonction arbitraire sur chacun des pixels au lieu de créer un nouvel opérateur pour cette fonction arbitraire.

[12]Ils appliquent respectivement la fonction log et la fonction exp à la valeur de chacun des pixels d'une image

[13]`https://jai.dev.java.net/servlets/ReadMsg?listName=interest&msgNo=3971`

```
UnaryFunction<Double, Double> function = new UnaryFunction<Double, Double>() {
   @Override
   public Double invoke(Double value) {
      double sqrt = Math.sqrt(value);
      retur sqrt;
      }
   };
ParameterBlock pb = new ParameterBlock()
                        .addSource(source)
                        .add(function);
RenderedImage sqrt = JAI.create("unaryfunction", pb));
```

Listing V.22 – Utilisation de l'operateur `unaryfunction`.

L'exemple illustré par le listing V.22 crée une nouvelle image dont la valeur de chacun des pixels est la racine carrée de la valeur du pixel original. La classe `UnaryFunction` provient de la bibliothèque `baselib` [38].