MASTER EN SCIENCES MATHÉMATIQUES

Impact de la précision du calcul du gradient dans le recalage d'images médicales avec interpolation par B-splines

Cohilis, Marie

Award date:
2017

Awarding institution:
Universite de Namur

UNIVERSITE DE NAMUR

Faculté des Sciences

IMPACT DE LA PRECISION DU CALCUL DU GRADIENT DANS LE RECALAGE D'IMAGES MEDICALES AVEC INTERPOLATION PAR B-SPLINES

Promoteur : Annick Sartenaer

Mémoire présenté pour l'obtention

du grade académique de master en Sciences mathématiques à finalité spécialisée

Marie COHILIS

Août 2017

Remerciements

J'aimerais remercier toutes les personnes ayant joué un quelconque rôle dans la concrétisation de mon mémoire.

Je remercie tout d'abord particulièrement ma promotrice Annick Sartenaer pour sa relecture, ses bons conseils et, de manière générale, le temps qu'elle m'a consacré.

Je voudrais également remercier le CHU de Dinant Godinne, et plus particulièrement Hubert Meurisse et Damien De Nizza, pour leur collaboration avec l'Université de Namur sans laquelle je n'aurais pas eu l'occasion de travailler sur un sujet aussi intéressant.

J'ai également une pensée pour mes professeurs de traitement des images médicales de l'Université d'Aveiro, Augusto Silvo et Silvia Francesco. Merci pour les indispensables bases apprises à votre cours, et merci d'avoir participé à développer mon intérêt pour l'imagerie médicale.

Enfin, merci à ma famille et à mes amis pour les encouragements durant toute la durée de ce travail. Je pense notamment aux étudiants de master 2, pour tous les bons moments passés et les encouragements mutuels.

Résumé

Dans le cadre du recalage d'images médicales, il est souvent question d'un compromis entre précision et temps d'exécution. Ce mémoire se penche sur le problème du point de vue du calcul du gradient de l'image, cette étape étant l'une des parties d'un algorithme de recalage d'images contribuant souvent à la perte de précision. Le gradient est en effet généralement approximé par des différences finies. Nous étudions ici le calcul du gradient exact, rendu possible par une interpolation de l'image utilisant B-splines cubiques, fonctions deux fois continûment différentiables. Cette alternative est ici couplée à un algorithme de recalage faisant évoluer les ensembles de niveau de l'image mouvante vers ceux de l'image fixe. Dans cette optique, ce mémoire expose d'abord un état de l'art de l'imagerie médicale et du recalage d'images avant de développer les méthodes utilisées. Une partie plus pratique présente ensuite l'implémentation du code à l'aide d'une librairie de traitement d'images du C++, ITK, ainsi que les différentes comparaisons réalisées entre les deux calculs de gradient possibles, menant à une conclusion mitigée.

Mots-clés : imagerie médicale, recalage d'images, interpolation, B-splines, gradient de l'image, librairie ITK.

Abstract

In the context of medical image registration, one of the main issues is to find a good compromise between accuracy and duration. This master thesis addresses the problem in terms of the computation of image gradient since this step often leads to a loss of accuracy. The gradient is indeed generally approximated by finite differences. We study here the computation of the exact gradient, made possible by an image interpolation using cubic B-splines functions, which are twice continuously differentiable. This alternative is here combined with a registration algorithm evolving the level-sets of the moving image into the ones of the reference image. To this end, the present work starts by establishing a state of the art of medical imaging along with image registration and then develops the used methods. A more practical part presents afterwards the algorithm implementation within ITK, a C++ library for image processing. It also shows the different tests conducted in order to compare both gradient computations, which lead to a mixed conclusion.

Keywords : medical imaging, image registration, interpolation, B-splines, image gradient, ITK library.

Table des matières

Introduction

Dans le milieu hospitalier, l'imagerie tient une place extrêmement importante. Différentes techniques permettent d'obtenir des images médicales montrant l'intérieur du corps du patient. Leur but est de fournir des informations sur l'anatomie et le fonctionnement des organes, participant ainsi au diagnostic, au suivi de l'évolution du patient ou encore à la planification de traitement. Les images obtenues ne sont cependant pas toujours utilisables directement ; un traitement préalable est alors nécessaire. C'est dans ce cadre que s'inscrit ce mémoire, réalisé en collaboration avec le Centre Hospitalier Universitaire de Dinant Godinne. Dans ce travail, nous nous intéressons à une discipline particulière du traitement d'images qu'est le recalage d'images. Cette technique permet de mettre en correspondance deux images similaires contenant des informations différentes. Son but est ainsi d'aider le médecin dans ses décisions par l'apport d'informations complémentaires.

La problématique du recalage d'images consiste souvent en un compromis entre temps et précision. En effet, le recalage doit être effectué de manière rapide tout en gardant un maximum de précision dans la mise en correspondance des deux images. Dans ce mémoire, cette problématique est abordée sous l'angle du calcul du gradient de l'image, nécessaire à la plupart des méthodes de recalage d'images. Nous évoquons deux possiblités pour ce calcul. L'une, plus fréquemment utilisée de par la rapidité du processus de recalage qu'elle engendre, propose d'approximer le gradient à l'aide de différences finies. L'autre, alternative que nous étudions ici pour son avantage de précision, calcule le gradient à l'aide de dérivées exactes. Ceci est rendu possible par l'introduction de fonctions continues et différentiables appelées B-splines dans la phase d'interpolation du recalage d'images qui permet de rendre continue une image discrète.

Ce travail est divisé en six chapitres. Le premier introduit formellement tout ce qui a trait à l'imagerie médicale. Nous abordons différentes caractéristiques de l'image ainsi que les moyens d'acquisition des images médicales. Le deuxième chapitre se concentre sur le traitement d'images, et plus particulièrement sur le recalage. Le Chapitre 3 introduit les fonctions B-splines ainsi que leur utilisation dans le cadre de l'interpolation pour les images via la théorie du signal. Le quatrième chapitre, lui, a pour but d'expliquer les notions théoriques liées à l'algorithme de recalage utilisé dans ce mémoire, à savoir la méthode des ensembles de niveau. Le Chapitre 5 expose ensuite la mise en pratique des notions vues précédemment via ITK, une librairie du C++ dédiée au traitement d'images médicales. Enfin, le dernier chapitre présente les différents tests effectués et les résultats obtenus.

Chapitre 1

Notions de base de l'imagerie médicale

Dans ce chapitre, nous introduisons progressivement le concept d'imagerie médicale et de tout ce qui y a trait.

1.1 Le concept d'image

A l'heure actuelle, l'être humain se retrouve quotidiennement face à des quantités d'images de tous genres. Le concept d'image est visuel ; nous savons tous ce que c'est, sans pour autant savoir ce qu'il en est de sa construction informatique, mathématique. C'est donc un prérequis de base qu'il nous faut établir pour commencer. La question est donc : qu'est-ce qu'une image ?

Nous nous intéressons bien sûr ici aux images dites digitales ou numériques. Celles-ci peuvent être définies comme des images pouvant être stockées, transmises et manipulées à l'aide d'un support informatique [14]. Si nous considérons les images de notre point de vue de mathématicien, nous pouvons en donner une définition plus formelle [16].

Définition 1.1 (Image)**.** *Une image est une fonction d-dimensionnelle qui représente une mesure de certaines caractéristiques telles que la couleur, la brillance d'une scène. Une image peut donc être définie par*

$$f : \mathbb{R}^d \longrightarrow \mathbb{R} : (x_1, \ldots, x_d) \longmapsto f(x_1, \ldots, x_d),$$

où $(x_1, \ldots, x_d)$ représente un point de l'image et $f(x_1, \ldots, x_d)$ l'intensité de l'image en ce point.

Notons que les points auxquels s'applique une fonction image forment une *grille d*-dimensionnelle. A partir de la définition 1.1, nous pouvons distinguer deux types d'images numériques :

- Les images *matricielles* :
  Ces images sont constituées de pixels (*picture elements*). Les pixels sont des petits carrés ayant chacun une position et des caractéristiques bien définies au sein de l'image. L'image peut donc être considérée comme une grille de points. En général, on ne distingue pas ces pixels à l'oeil nu, ce n'est que lorsque l'on zoome sur l'image que l'on s'aperçoit de cette construction, et c'est d'ailleurs le principal défaut de ce type d'images : on ne peut pas à la fois les agrandir et garder leur netteté. Cette situation est illustrée à la figure 1.1, où l'on voit qu'en zoomant sur une certaine zone de l'image, on peut distinguer ses pixels.

- Les images *vectorielles* :
  Une image vectorielle est composée non pas de pixels mais de formes géométriques élémentaires telles que des segments de droites, des ellipses, des arcs de cercle, etc. Chacune de ces formes possède un ensemble de caractéristiques telles qu'une couleur et une position. De ce fait, l'image est beaucoup plus "mathématique". En pratique, lorsque l'on souhaite agrandir

une image vectorielle, l'ordinateur recalcule toutes les formes à l'échelle souhaitée, ce qui permet d'éviter une image floue, comme le montre la figure 1.2.

La grande différence entre les deux types d'images se situe donc au niveau du redimensionnement.

Lisant la description ci-dessus, on pourrait penser que les images matricielles n'ont pas de réel intérêt par rapport aux images vectorielles. Ce n'est bien sûr pas le cas : les images vectorielles ont en fait l'inconvénient d'être difficilement manipulables, mais surtout difficiles à obtenir. En effet, il n'est pas toujours évident de garder le réalisme, la précision ou encore les couleurs de certaines images (notamment les images médicales) en les construisant uniquement avec des formes géométriques. En pratique, donc, les images vectorielles seront essentiellement des logos, des images créées artificiellement et non pas des photographies. Les images médicales que nous manipulons ici sont donc de type matriciel. Tous les concepts que nous abordons par la suite concernent ainsi les images matricielles.

1.2 Caractéristiques propres à une image

Les images peuvent être classées selon plusieurs critères qui leur sont propres. Nous allons ici en passer certains en revue, à savoir ceux qui nous intéressent dans le contexte de ce mémoire. Six caractéristiques des images nous semblent ainsi essentielles. Certaines sont propres aux images médicales, d'autres non.

1.2.1 La dimension

La définition 1.1 donnée précédemment mentionne les images comme étant des fonctions multi-dimensionnelles. Il existe en effet des images de *dimension* supérieure à 2, particulièrement dans le monde de l'imagerie médicale où l'on peut obtenir des images 2D, 3D ou même 4D. Dans ce mémoire, nous nous concentrerons uniquement sur les images en deux ou trois dimensions.

FIGURE 1.1 – Illustration de la nature d'une image matricielle [51]

FIGURE 1.2 – Agrandissement d'images au format vectoriel (à gauche) et matriciel (à droite) [14]

Une image 2D est simplement une image telle que décrite jusqu'ici : c'est une grille 2D de pixels qui, mathématiquement, peut être représentée par une certaine fonction de deux variables $f(x_1, x_2)$ où (x_1, x_2) correspond à un point de la grille.

Une image 3D, au premier abord, semble plus difficile à visualiser. En pratique, en imagerie médicale, une image 3D se concrétise par une pile d'images 2D. Pour que cela ait du sens, il faut bien sûr que les images représentent des scènes proches, à savoir par exemple plusieurs coupes d'un poumon. Ainsi, une fois superposées, on peut obtenir une visualisation 3D du poumon, grâce par exemple à des logiciels de traitement d'images comme *MevisLab* [17]. L'image ne nous parvient donc pas directement comme une image 3D mais bien comme une série d'images que l'on retravaille ensuite pour obtenir une visualisation 3D. La figure 1.3 représente un échantillon d'une telle pile d'images 2D (l'original en compte 436), tandis que la figure 1.4 nous montre leur représentation 3D, basée sur la pile entière. La différence en termes de grille entre une image 2D et une image 3D est illustrée à la figure 1.5.

Remarquons que dans les images 3D, un élément (x_1, x_2, x_3) est appelé voxel (*volume element*) et non pixel. Nous prendrons cependant ici la convention d'utiliser le mot pixel pour désigner un pixel ou voxel d'une image de dimension quelconque.

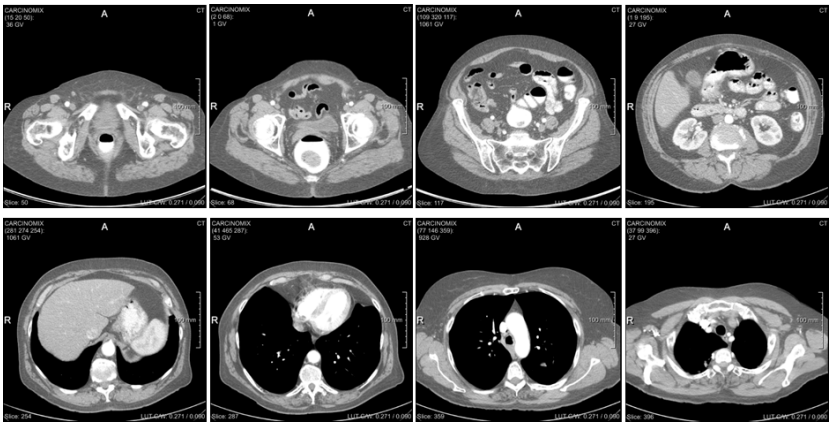


FIGURE 1.3 – Echantillon d'une pile d'images 2D illustrant un tronc humain [31]

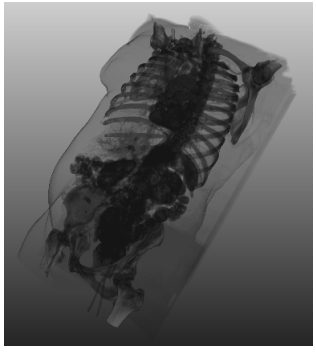


FIGURE 1.4 – Visualisation 3D d'une pile d'images illustrant un tronc humain [31]

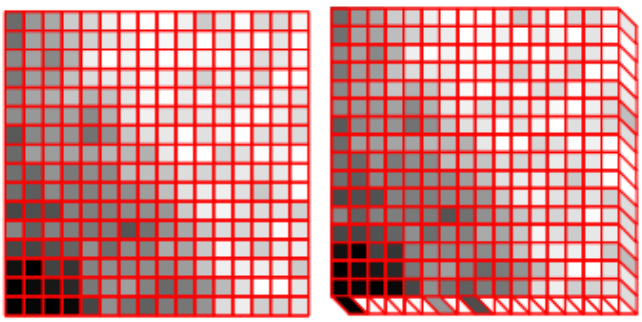


FIGURE 1.5 – Une grille de pixels (à gauche) et une grille de voxels (à droite) [1]

1.2.2 La définition

La *définition* d'une image est le nombre total de pixels dont elle est composée. Elle est souvent exprimée sous la forme $m_1 \times \ldots \times m_d$, où $m_1, \ldots, m_d$ sont des entiers représentant respectivement le nombre de pixels selon chaque dimension de l'image. La définition correspond donc en quelque sorte au poids de l'image. Son unité est le pixel (p) [4].

1.2.3 La résolution

La *résolution* d'une image est le nombre de pixels d'une image par unité de surface ou de volume. Elle correspond donc à la finesse de l'image. Son unité est le pixel par pouce (*ppp*) [4].

1.2.4 L'espace physique

Les images sont également définies via leur *espace physique*. Comme nous l'avons déjà mentionné, une image est une fonction donnant l'intensité d'une grille de points. Cette grille de points peut être accédée de manière absolue à l'aide d'index, les pixels étant alors simplement numérotés de manière entière. Une autre approche est d'utiliser son espace physique, c'est-à-dire de considérer la grille dans un système de coordonnées. Une image possède en effet intrinsèquement une origine ainsi que ce qu'on appelle un spacing, donnant l'écart entre les points de la grille, i.e., entre les pixels. Le spacing est unique pour une même direction ; une image d-dimensionnelle a donc un spacing sous forme de vecteur de taille d. Ces caractéristiques de l'image sont essentielles lorsqu'on s'intéresse au traitement d'images ; nous y reviendrons.

1.2.5 Le format

Dans cette partie, nous présenterons quelques *formats* d'image pour les images de type matriciel, dans le cadre de l'imagerie médicale mais aussi en dehors. Cependant, avant de les passer en revue, il faut savoir que toute image, quel que soit son format, possède un agencement typique. En effet, les images sont composées de deux parties :

– *Le header :*
Toute image est entre autres formée d'un header, aussi dit *Metadata*, qui contient des informations techniques sur le format, la couleur, la définition, la compression, ... En bref, il doit contenir tout ce qu'il faut pour pouvoir reconstruire l'image à partir de la seconde partie de l'image décrite ci-dessous ; c'est grâce à lui qu'un logiciel est capable d'ouvrir correctement une quelconque image [16]. Dans le cas des images médicales, le header contient également des informations sur le mode de production de l'image. Celles-ci dépendent de la modalité

d'acquisition de l'image, mais nous reviendrons là-dessus plus loin, au paragraphe 1.3 [25].

– *L'information sur les pixels :*
La deuxième partie d'une image contient simplement les valeurs numériques des pixels, les valeurs permettant d'exprimer les différentes intensités de l'image. Il existe plusieurs manières de stocker ces nombres ; c'est fonction du format de l'image [25].

Avant de décrire les différents formats, notons que la problématique du format d'images repose sur plusieurs exigences. Citons entre autres le poids de l'image, le type de header, la facilité de transmission ou encore la compatibilité des logiciels [7]. Le format préférable dépend donc du contexte dans lequel on utilise l'image, et nous aborderons ici le problème sous l'angle de l'imagerie médicale. Citons donc rapidement quelques formats d'images couramment utilisés dans tous types de domaines, avant de nous étendre plus longuement sur le format préféré dans le monde de l'imagerie médicale : le format DICOM.

- *Le format JPEG :*

  Ce qu'on appelle communément le format JPEG (*Joint Photographic Experts Group*) est en fait une méthode de compression et non un format. En effet, les fichiers *.jpg* sont des images déjà compressées. Ils sont donc moins lourds que d'autres types de fichiers que nous verrons par la suite. Cependant, la compression JPEG entraine des pertes d'informations sur l'intensité, la couleur de l'image. Dans le cadre de l'imagerie médicale, ce n'est donc pas un type de format adapté puisque trop imprécis.

- *Le format GIF :*

  Le format GIF (*Graphic Interchange Format*), qui est également un format compressé, cible un type d'images particulier. En effet, il ne permet que très peu de précision et peu de couleurs. Cependant, contrairement au format JPEG, un seul fichier GIF peut contenir une pile d'images 2D, c'est-à-dire une image 3D. Dans le domaine de l'imagerie médicale, les fichiers GIF ne sont bien sûr pas du tout utilisés puisque beaucoup trop imprécis, malgré le fait qu'ils puissent être à trois dimensions.

- *Le format PNG :*

  A nouveau, le format PNG (*Portable Network Graphics*) est un format d'images compressées. Cependant, contrairement au JPEG, il ne provoque pas de perte d'information sur l'intensité et la couleur de l'image. Il prend donc plus de place en mémoire mais reste en général acceptable pour les transmissions. Dans le cadre de l'imagerie médicale, le format PNG pourrait convenir grâce à sa bonne précision et sa facilité de transmission. Cependant, son header lui fait défaut, comparé au standard DICOM que nous verrons par la suite.

- *Le format TIFF :*

  Le format TIFF (*Tagged Image File Format*) mise tout sur la qualité de l'image. De plus, il permet, tout comme le format GIF, de stocker une pile d'images 2D dans un seul fichier. Ce n'est pas un format compressé de base, mais il existe des manières de le compresser afin de pouvoir le transmettre plus facilement. En effet, à cause de sa bonne qualité et de sa possibilité 3D, un fichier TIFF prend énormément de place en mémoire. Dans le cadre de l'imagerie médicale, un fichier TIFF pourrait donc être tout à fait envisageable. Cependant, en pratique, il est peu utilisé car il n'est compatible qu'avec peu de logiciels.

- *Le format DICOM :*

  Comme dit précédemment, le format DICOM (*Digital Imaging and Communications in Medicine*) est le format le plus utilisé en imagerie médicale. Créé en 1993 par l'ACR (*American College of Radiology*) et la NEMA (*National Electrical Manufacturers Association*), ce format fait son apparition suite à plusieurs problèmes rencontrés par les médecins.

  Un premier problème est le transfert d'images. Grâce à la norme DICOM, les transferts d'images entre les différents constructeurs sont devenus beaucoup plus aisés. De même, on a pu standardiser les formats d'images d'un type d'appareil à l'autre, c'est-à-dire que quel que soit le moyen d'acquisition de l'image, l'image produite a toujours le même format. La création de la norme DICOM a donc permis d'éviter de nombreux problèmes de maintenance et gestion dans les centres médicaux, notamment des problèmes d'incompatibilité, de coût, ou encore de perte d'information [41].

  Deuxièmement, la norme DICOM a permis d'introduire dans l'image des informations concernant le patient, le moyen d'acquisition de l'image, etc. Le header d'une image DICOM contient donc des champs spécialement dédiés à des données médicales. Cela permet d'éviter les erreurs humaines, ou encore de faciliter les échanges entre hôpitaux.

  Plus concrètement, de quoi est composée une image DICOM ? Comme on le sait déjà, il y a d'une part les pixels de l'image (leur valeur), et d'autre part le header. Celui-ci est composé par les quatre éléments suivants :

  – Tout d'abord, une image DICOM comprend toujours une *étiquette* (ou *tag*), qui est composée d'un numéro de groupe encodé par deux octets ainsi que d'un numéro d'élément encodé par deux octets également. Par exemple, une étiquette peut être $(0010, 0030)$. Le premier élément, c'est-à-dire le numéro de groupe, correspond au type d'information général. C'est en fait une catégorie de données. Le second élément d'une étiquette, c'est-à-dire le numéro d'élément, précise l'information qui sera donnée. Dans notre exemple, 0010 représente les données concernant le patient, tandis que 0030 indique plus particulièrement que l'information fournie sera la date de naissance du patient [50]. Les principaux numéros de groupes indiquent notamment les données sur le patient (0010), les données sur le centre médical (0008), les informations sur l'acquisition de l'information (0018), les données sur le positionnement du patient (0020), les données sur la présentation de l'image (0028) ou encore simplement les commentaires (4000) [11].

  – Ensuite, le header d'une image DICOM comprend ce qu'on appelle une *représentation de valeur*, encodée grâce à deux caractères. Il fournit le type de l'information donnée. Par exemple, pour notre exemple d'étiquette $(0010, 0030)$, la représentation de valeur sera DA, qui est le sigle représentant une date. Parmi les nombreuses représentations de valeur possibles, citons notamment AS (Age String), LT (Long Text), PN (Person Name) et UL (Unsigned Long) [50].

  – Le header contient également une information sur la longueur de l'information donnée, appelée *longueur de la valeur* [11]. C'est un entier non signé.

  – Pour finir, le header DICOM comprend l'information en elle-même, dite *valeur* [11]. Dans notre exemple d'étiquette $(0010, 0030)$, il s'agit donc de la date de naissance du patient.

La figure 1.6, obtenue à l'aide du logiciel de traitement d'images *ImageJ* [38], nous montre une partie des informations disponibles concernant une image DICOM.

Il faut donc retenir de tout ceci que, aujourd'hui, le format DICOM domine le monde de l'imagerie médicale de par son adéquation aux problèmes rencontrés dans ce milieu.

1.2.6 L'angle de vue

Jusqu'ici, nous n'avons pas encore vu de concepts réellement propres au domaine de l'imagerie médicale. Cette section s'inscrit dans ce cadre uniquement. Comme nous l'avons vu dans la section précédente, les images 3D de l'imagerie médicale sont formées de plusieurs images 2D représentant des coupes d'une partie du corps. Mais ces coupes peuvent être réalisées selon différents plans [11] :

- *Le plan frontal :*
  Aussi appelé *plan coronal*, ce plan sépare le corps en une partie avant (ventrale) et une partie arrière (dorsale).
- *Le plan sagittal :*
  Il s'agit du plan qui sépare la moitié gauche et la moitié droite du corps.
- *Le plan transversal :*
  C'est le plan qui sépare le corps en une partie supérieure et une partie inférieure.

0002,0002 Media Storage SOP Class UID: 1.2.840.10008.5.1.4.1.1.2
0002,0003 Media Storage SOP Inst UID: 1.2.392.200036.9116.2.2.2.1762660276.1130217711.527565
0002,0010 Transfer Syntax UID: 1.2.840.10008.1.2.1
0002,0012 Implementation Class UID: 2.16.840.1
0002,0013 Implementation Version Name: MergeCOM3_330
0008,0008 Image Type: ORIGINAL\PRIMARY\AXIAL
0008,0016 SOP Class UID: 1.2.840.10008.5.1.4.1.1.2
0008,0018 SOP Instance UID: 1.2.392.200036.9116.2.2.2.1762660276.1130217711.527565
0008,0020 Study Date: 20051025
0008,0022 Acquisition Date: 20051025
0008,0023 Image Date: 20051025
0008,0030 Study Time: 141144.000
0008,0032 Acquisition Time: 141852.650
0008,0033 Image Time: 141857.275
0008,0050 Accession Number: 481497
0008,0060 Modality: CT
0008,0070 Manufacturer: TOSHIBA
0008,0080 Institution Name: ???
0008,0090 Referring Physician's Name: ???
0008,1010 Station Name: ASTEION FEIRA
0008,1040 Institutional Department Name: ID_DEPARTMENT
0008,1090 Manufacturer's Model Name: Asteion
0010,0010 Patient's Name: A.M.R.
0010,0020 Patient ID: 20060119172140
0010,0030 Patient's Birth Date: 19290519
0010,0040 Patient's Sex: O
0010,1010 Patient's Age: 076Y
0010,4000 Patient Comments: CONTRASTE\ENDOVENOSO\\MR SC
0018,0022 Scan Options: HELICAL_CT
0018,0050 Slice Thickness: 5.0

FIGURE 1.6 – Echantillon d'informations associé à une image au format DICOM [38]

Ces trois plans forment donc un système d'axes en trois dimensions ; chacun d'eux peut offrir une prise de vue différente d'un même organe, permettant aux médecins d'observer différentes parties de cet organe, ou une même partie sous différents angles.

La figure 1.7 schématise ces trois plans, tandis que la figure 1.8 nous montre des images réelles d'un cerveau réalisées selon chacun des trois plans.

1.2.7 La couleur

Ce que nous appelons ici la couleur correspond en fait au type d'images en fonction de leurs couleurs. Plus précisément, cette section concerne l'intensité des pixels, c'est-à-dire leur valeur. De ce point de vue, il existe trois types d'images :

– *L'image binaire :*
Il s'agit du type d'images le plus basique. Les pixels valent soit 1, soit 0. Concrètement, cela signifie que l'image comprend uniquement du noir et du blanc (0 pour noir, 1 pour blanc). Il n'y a donc pas réellement d'échelle d'intensité. La figure 1.9 montre un exemple d'image binaire.

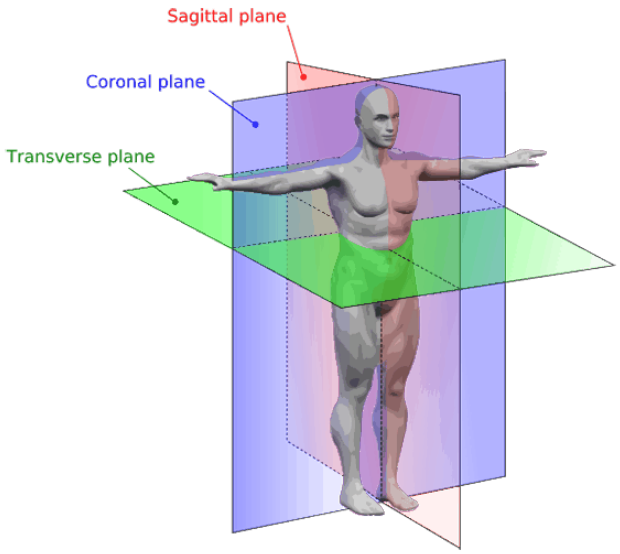


FIGURE 1.7 – Les différents plans du corps humain [40]

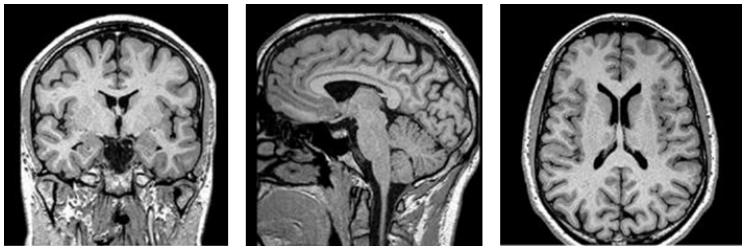


FIGURE 1.8 – Coupes frontale (à gauche), sagittale (au milieu) et transversale (à droite) d'un cerveau [12]

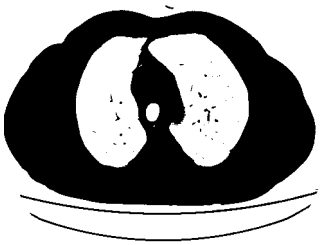


FIGURE 1.9 – Image binaire (coupe transversale d'un tronc humain) [31]

– *L'image en niveaux de gris :*
Dans le cas de l'image en niveaux de gris, les pixels peuvent prendre un nombre de valeurs quelconque. Cependant, plus ils peuvent prendre de valeurs différentes, plus l'image prend de la place en mémoire. Les niveaux de gris correspondent à une échelle d'intensité qui, la plupart du temps, est positive : le 0 représente le noir tandis que le plus haut niveau représente le blanc, et tous les nombres entre 0 et ce plus haut niveau représente différentes nuances de gris (du plus foncé au plus clair). Malgré qu'un nombre quelconque de niveaux de gris soient possibles, il existe des tailles types d'images. Une image de taille j-bit offrira 2^j différentes possibilités de niveaux de gris. Une image 8-bit, par exemple, donnera une variation d'intensité des pixels entre 0 et 255, c'est-à-dire 256 niveaux de gris possibles. Cependant, il n'est pas rare qu'une image 8-bit ne contienne pas réellement 256 valeurs de pixels différentes. Considérons par exemple une image 8-bit. Soit a_{min} la plus petite valeur prise par un de ses pixels et a_{max} la plus grande. Le nombre

$$d = a_{max} - a_{min}$$

est alors appelé *gamme dynamique* de l'image [13]. A partir des valeurs de chaque pixel, on peut ainsi créer un histogramme dépendant de la gamme dynamique ou de l'intervalle [0,255], nous donnant les proportions de pixels pour chacune de leurs valeurs possibles. Cet histogramme est très utilisé dans le domaine du traitement d'images. La figure 1.10 montre une image en niveaux de gris (8-bit) et son histogramme associé, obtenu par le logiciel *MevisLab* [17].

Notons qu'une image binaire peut facilement être obtenue à partir d'une image en niveaux de gris. Par exemple, pour une image 8-bit, il suffit pour cela de déterminer un nombre seuil T

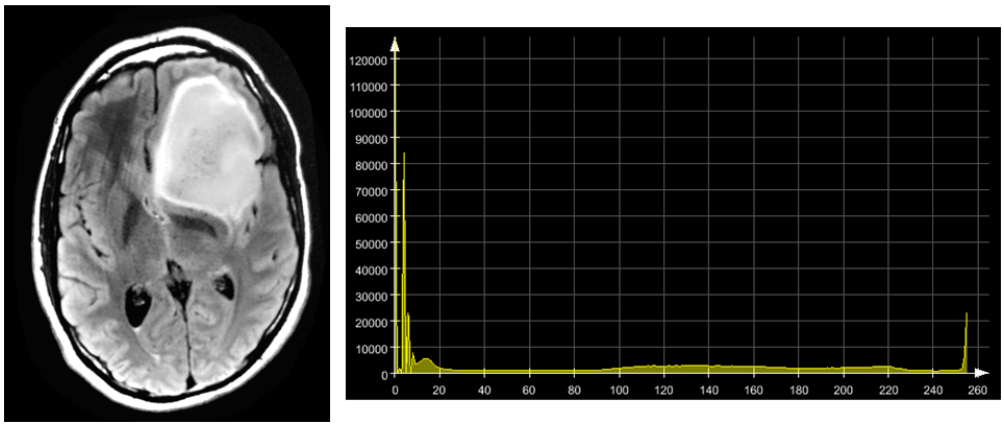


FIGURE 1.10 – Image 8-bit et son histogramme [31], [17]

(*treshold*) entre 0 et 255 comme étant la limite entre noir et blanc : tous les pixels de l'image en niveaux de gris ayant une valeur inférieure à T vaudront 0 sur l'image binaire, tandis que tous les nombres supérieurs à T vaudront 1.

Précisons aussi que toutes les images en niveaux de gris ne possèdent pas une échelle comme celle décrite. C'est notamment le cas en imagerie médicale, où il n'est pas rare de rencontrer des pixels à intensité négative.

– *L'image en couleur :*
Avant de parler des images couleurs, il faut connaitre le principe de *trichromie*. Celui-ci nous apprend que, sur base d'au moins trois couleurs (pas n'importe lesquelles), il est possible, en les mélangeant, de recréer toutes les autres couleurs. Plusieurs combinaisons de couleurs sont possibles pour cela, mais la plus classique est l'espace couleur RGB (*red - green - blue*). La plupart des images couleur utilisent ce système [30].

Revenons à nos images. Les pixels d'une image couleur possèdent la particularité d'avoir trois valeurs. Prenons un exemple 2D. Au lieu d'avoir une image définie par une matrice $m_1 \times m_2$, nous avons une image définie par trois matrices $m_1 \times m_2$. Chacune de ces matrices représente la quantité (l'intensité) de rouge, de vert ou de bleu qui définit la couleur de chaque pixel. Donc, chaque pixel est un vecteur de trois composantes : une pour le rouge, une pour le vert et une pour le bleu. Similairement aux images en niveau de gris, ces valeurs sont comprises entre 0 et 255 (pour les images 8-bit). Le système des images couleurs est illustré à la figure 1.11, tandis que la figure 1.12 illustre la décomposition d'une image couleur en trois sous-images, une pour les tons rouges, une pour les tons verts et une pour les tons bleus. Chacune de ces trois images peut donc être comparable à une image en niveaux de gris pour laquelle on utilise une couleur au lieu du blanc.

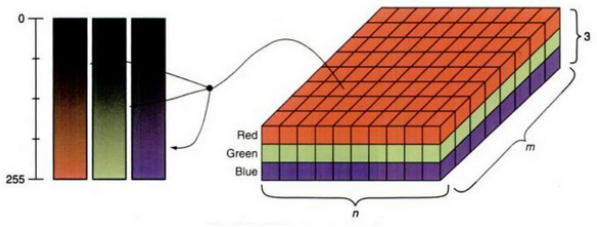


FIGURE 1.11 – Représentation mathématique d'une image couleur via l'espace RGB [16]

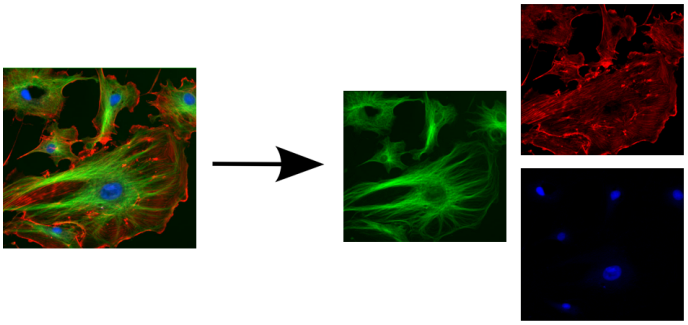


FIGURE 1.12 – Séparation d'une image RGB en trois images basiques (rouge, vert et bleu) [38]

1.3 Acquisition des images médicales

Maintenant que nous avons vu différentes caractéristiques d'une image, il est temps de s'intéresser à ce qu'on appelle la *modalité* d'une image : comment l'image a-t-elle été obtenue ? Nous faisons bien sûr ici référence aux images médicales uniquement.

Pour commencer, ils nous faut nous intéresser au but de l'examen médical que l'on souhaite réaliser. En effet, on peut vouloir étudier une partie du corps selon deux états différents : soit l'état *structurel*, soit l'état *fonctionnel*. Certaines modalités permettront d'obtenir des images *structurelles*, tandis que d'autres donneront des images *fonctionnelles* [52]. Une image *structurelle* ou *anatomique* fournit des informations sur la structure des organes : forme, limites, volume, etc. Une image *fonctionnelle* renseigne sur la fonction des organes : métabolisme, physiologie, etc.

Nous allons donc regrouper les différentes modalités de l'imagerie médicale selon qu'elles fournissent des images structurelles ou fonctionnelles. Cependant, au vu du grand nombre de technologies différentes existant sur le marché, nous nous contentons ici de décrire les grandes familles de modalités.

1.3.1 Techniques d'imagerie structurelle

Nous retrouvons principalement, dans cette catégorie, les quatre types de modalités suivants [7] :

- *L'imagerie par résonance magnétique (IRM) :*
  Le principe est simple : les molécules d'eau (plus particulièrement leurs atomes d'hydrogène) possédant un moment magnétique, l'appareil crée un champ magnétique puissant dans un tunnel où est placé le patient. Sous l'effet de ce champ, les molécules d'eau vont tendre à s'orienter toutes dans le même sens. Ensuite, une antenne émettra des brefs signaux (fréquences) qui perturberont la trajectoire des molécules d'eau. Une fois le signal fini, les molécules reprendront leur place initiale due au champ magnétique, émettant au passage des signaux électroniques, d'autres fréquences, qui seront captées par l'antenne. Celle-ci contiendra alors un signal électrique qui sera analysé et permettra la création de l'image, grâce au fait que le signal renvoyé diffère en fonction de la quantité d'eau contenue par les tissus [27].

- *L'imagerie par rayons X :*
  C'est ce qu'on appelle couramment la radiographie. La radiographie utilise le fait que les rayons X traversent le corps de manière plus ou moins importante en fonction de la densité des tissus. Des photons émis par une source placée devant le corps sont en partie absorbés par les tissus humains, et ceux qui traversent sont réceptionnés par un détecteur placé à l'arrière du corps. Le domaine de l'imagerie par rayons X regroupe notamment la radiologie standard (technique 2D) telle que décrite ci-dessus, mais également des dérivés tels que le CT-scan (Computed Tomography) qui correspond à de la radiographie 3D.

- *L'imagerie optique :*
  Contrairement aux techonologies utilisant des rayons X, l'imagerie optique utilise la lumière visible. Les propriétés des photons sont exploitées pour obtenir des images précises d'organes, tissus ou même de très petits éléments tels que des cellules. L'avantage est qu'on obtient des images en couleurs, et que le patient n'est que très peu exposé à la radiation [37].

- *L'imagerie par ultrasons :*
  C'est ce qu'on appelle couramment les échographies. Similairement aux technologies par rayons

X, cette technologie se base sur l'émission et la réception d'ondes ultrasonores. Les ondes émises traversent les tissus et, en fonction de la nature des tissus, y font écho différement (plus un tissu est dense, plus l'écho est important) [27].

La figure 1.13 illustre chacune de ces quatre grandes familles de modalités.

1.3.2 Techniques d'imagerie fonctionnelle

Passons à présent en revue les deux principales familles de l'imagerie fonctionnelle.

- *L'imagerie par émission (ou scintigraphie) :*
  Dans le cas de l'imagerie par émission, des radioéléments bien choisis sont injectés dans le corps grâce à un traceur (en petite quantité bien sûr) et vont se fixer, grâce à leurs propriétés, sur l'organe ou le tissu que l'on souhaite examiner. Ensuite, en se désintégrant, ces radioéléments émettent des rayons gamma et un détecteur de rayons gamma relié à un ordinateur les détecte et les analyse. Cet appareillage de détection est appelé gamma-caméra, il permet de créer l'image [27]. Parmi les technologies fonctionnant par émission, on retrouve la scintigraphie planaire, mais aussi le SPECT-scan (Single Photon Emission Computed Tomography) et le PET-scan (Positron Emission Tomography) ; ces deux dernières technologies fonctionnent par *tomographie*, c'est-à-dire qu'elles utilisent une gamma-caméra tournante ou une couronne de détecteurs fixes dans le but d'obtenir plusieurs images et donc par la suite une reconstruction en 3D.

  Notons que cette famille de techniques d'imagerie fonctionnelle, basées sur l'administration de substances radioactives au patient dans le but d'établir un diagnostic, constitue l'activité principale du domaine spécialisé de la médecine appelé médecine nucléaire.

  La figure 1.14 illustre une scintigraphie planaire du corps entier (d'abord de face puis de dos). Elle indique au niveau du genou droit une probable tumeur.

- *L'imagerie par résonance magnétique fonctionnelle (IRMf) :*
  Le principe de base de l'IRMf est le même que celui de l'IRM classique. Cependant, l'IRMf est principalement destinée à la neurologie. En soumettant le patient à un stimulus (par exemple le faire lire, écouter, calculer, etc), les zones du cerveau activées par cette tâche reçoivent plus d'oxygène que lorsque le patient ne fait rien. Ces échanges d'oxygène entre le sang et les neurones modifient ainsi le signal émis par les molécules d'eau. La différence entre les deux états (inactif et actif) est alors analysée par ordinateur [27].

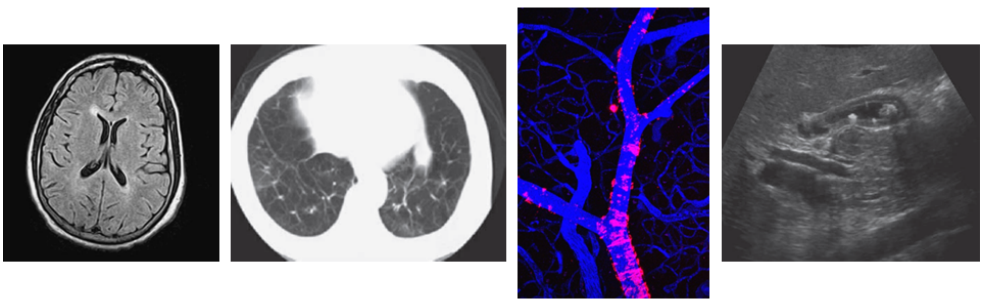


FIGURE 1.13 – De gauche à droite : IRM, CT-scan, imagerie optique, échographie [27], [13]

FIGURE 1.14 – Scintigraphie planaire du corps humain [13]

1.4 Conclusion

Nous avons brièvement introduit, dans ce chapitre, le domaine de l'imagerie médicale. Nous avons vu ce qu'était une image, quelles étaient les différentes possibilités pour la caractériser, mais également comment obtenir une image médicale.

Dans le prochain chapitre, nous nous penchons plutôt sur ce que l'on peut faire avant d'analyser des images médicales pour en améliorer l'interprétation : il s'agit du traitement d'images, une discipline tirée des domaines de l'informatique et des mathématiques. Nous nous concentrons particulièrement sur le recalage d'images puisque ce mémoire traite majoritairement de ce domaine.

Chapitre 2

Recalage d'images

Dans ce chapitre, nous abordons le *traitement d'images*, et principalement le *recalage d'images*. Comme dit précédemment, le traitement d'images est une discipline mathématique et informatique. Elle a été créée dans le but d'aider les médecins dans leur tâche d'analyse des images médicales. En effet, il n'est pas rare que les images produites par les technologies d'imagerie médicale ne soient pas faciles à interpréter : peu de contraste, beaucoup de bruit (c'est-à-dire des éléments qui ne devraient pas se trouver sur l'image), de l'information incomplète, etc. Certains domaines du traitement d'images permettent également simplement d'apporter des informations complémentaires.

C'est donc dans ce cadre que les médecins font appel aux informaticiens, mathématiciens ou encore ingénieurs pour faciliter et assurer leurs analyses d'images. Grâce à leurs compétences particulières, ceux-ci ont pu mettre en place des techniques diverses permettant une meilleure visualisation des éléments des images, de manière à clarifier les informations données par celles-ci. Avant de rentrer dans le détail de l'explication du recalage d'images, nous abordons rapidement quelques grands domaines du traitement d'images.

Typiquement, la première étape du traitement d'image est l'*amélioration de l'image*. Celle-ci a un but de clarification de l'image ; on souhaite améliorer la perception des informations données par l'image. La plupart du temps, les médecins désirent obtenir des images médicales dans le but d'observer des éléments bien précis (organes, tissus, os, etc). Le but de cette première famille du traitement d'images consiste à mettre ces éléments en valeur, à les faire ressortir afin de faciliter la tâche d'analyse du médecin.

Un exemple classique est de modifier le contraste pour augmenter la visibilité de certaines parties de l'image et de réduire le bruit. Une fois cela fait, la plupart du temps, on obtient une image pouvant faciliter le diagnostic [2]. Un bon contraste est défini via l'histogramme de l'image. En effet, il n'est pas bon que le contraste soit trop faible, mais il ne doit pas être trop fort non plus ! L'idéal est donc que la gamme dynamique de l'image soit grande et que l'histogramme de l'image soit assez bien réparti sur cette gamme dynamique.

Un autre domaine essentiel du traitement d'images est la *segmentation*. Il se situe dans la continuation de l'amélioration d'images puisque son but est l'isolation de certaines régions de l'image. Cela peut être fait via un coloriage ou un contour ; on crée ainsi ce qu'on appelle des *régions d'intérêt* au sein de l'image. Une fois cela réalisé, il est possible d'obtenir mathématiquement des informations bien plus précises sur ces régions que celles que fournirait l'oeil.

La figure 2.1 illustre, à gauche, une image 8-bit à bon contraste et la même image à mauvais contraste, ainsi que leurs histogrammes respectifs. A droite, nous observons une IRM du cerveau

où une tumeur a été segmentée par un contour vert via le logiciel *ImageJ* [38].

Avant de passer au recalage d'images, mentionnons un dernier domaine du traitement d'images qu'est la *quantification*. Celui-ci intervient après la segmentation puisque son but est de quantifier de différentes manières les régions segmentées. Citons notamment le calcul du volume ; il peut par exemple être très intéressant de surveiller l'évolution d'une tumeur en connaissant son volume exact.

Passons enfin au dernier domaine du traitement d'images qui nous intéresse, à savoir le recalage d'images. Le recalage consiste en la superposition de deux images représentant des objets similaires. Son but est de pouvoir ainsi comparer ou combiner leurs informations respectives [33]. Toute la problématique de ce domaine repose sur la mise en correspondance des images, i.e., la recherche de la meilleure façon de les superposer.

Le rôle de l'optimisation dans le recalage apparaît donc ici clairement : on recherche la transformation spatiale optimale qui permettra d'envoyer une première image J, que l'on appellera ici image *mouvante*, sur une deuxième image I dite *de référence*. Mathématiquement parlant, il s'agira donc de trouver une transformation $\hat{T}$ telle que

$$\hat{T} = \arg \max_{T \in \mathcal{T}} S(I, T(J)), \tag{2.1}$$

où $\mathcal{T}$ est l'ensemble des transformations spatiales et S est une fonction décrivant la *similarité* entre deux images, ici I et $T(J)$, l'image J après transformation.

Comme l'indique l'équation (2.1), l'objectif mathématique du recalage est donc de trouver la fonction $\hat{T}$ qui maximisera la similarité entre une image I et une image transformée $\hat{T}(J)$. Notons que nous utilisons ici un maximum pour symboliser le fait que l'on souhaite maximiser la similarité entre images, mais en pratique, selon la fonction de similarité, il se peut que nous devions plutôt utiliser un minimum [6]. Le but est alors de minimiser la dissimilarité entre images.

Une autre remarque importante à faire concerne la notation de l'image mouvante transformée, $T(J)$. En effet, il est important de se souvenir, cf. Chapitre 1, que l'image J est une fonction attri-

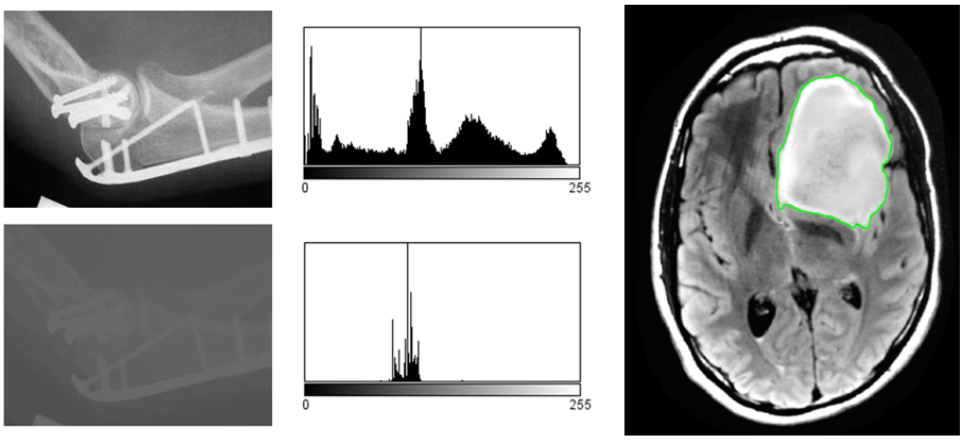


FIGURE 2.1 – A gauche : images bien contrastée (haut) et mal contrastée (bas) et leurs histogrammes ; à droite : IRM du cerveau avec segmentation d'une tumeur en vert [7], [38], [31]

buant à un point de coordonnées spatiales une intensité. La notation $T(J)$ indique donc simplement la transformée de l'image de manière intuitive, mais cependant il serait plus correct, mathématiquement parlant, de noter $J(T)$, puisque la transformation T s'applique bien aux coordonnées des points de l'image et non à leur intensité. On applique donc en réalité la fonction d'intensité aux nouvelles coordonnées des points de l'image mouvante.

Avant de poursuivre plus en détails, il nous faut remarquer que le recalage d'images peut s'effectuer sur tout type d'images, quelles que soient la dimension, la modalité d'acquisition, la couleur, etc. Il peut notamment s'effectuer entre des images possédant des caractéristiques différentes : citons entre autre une image 2D et une image 3D, une image obtenue par scintigraphie et une image obtenue par CT-scan ou encore des images provenant de différents patients. Chaque type de recalage peut avoir son utilité. On distingue d'ailleurs souvent les recalages effectués avec des images provenant d'une même modalité ou non (mono-modal ou multi-modal), ainsi que les recalages effectués avec des images d'un même patient ou non (intra-sujet ou inter-sujets). En combinant ces possibilités, on obtient ainsi quatre types de recalages ; chacun d'eux a son utilité et mène à des techniques de recalage particulières.

Il existe donc toute une pléthore d'algorithmes de fusion d'images. Les différents types de recalages peuvent ainsi être classifiés selon différentes possibilités ; pour en avoir une vue globale, le lecteur intéressé peut se référer à [29]. Nous analyserons principalement ici cinq grands critères de classification des méthodes de recalage, à savoir ceux qui nous permettront de décrire au mieux un processus général de recalage. Il s'agit des *attributs de l'image*, de la *fonction de similarité*, des *modèles de transformation*, des *techniques de rééchantillonnage* et des *méthodes d'optimisation*. Ces critères sont décrits dans les sections qui suivent.

Avant de les étudier, observons un premier exemple de recalage à la figure 2.2, obtenu par le logiciel *Mevislab* [17]. Il s'agit du recalage de deux images 3D d'un même patient, l'une obtenue par rayons X (CT-scan) et l'autre par PET-scan. Le but ici est donc de combiner informations *structurelles* et *fonctionnelles* du cerveau. Notons que la couleur de l'image obtenue par PET-scan n'est pas naturelle ; elle a été rajoutée afin de mieux percevoir les informations données par le PET-scan. Cela nous permet ainsi de faire ressortir certaines zones telles que celles en orange, indiquant notamment la présence d'une tumeur aux poumons, pointée par le curseur sur les images recalées. On perçoit donc bien ici l'importance de l'étape d'amélioration de l'image. Grâce au recalage, il est également possible d'obtenir des informations structurelles sur cette tumeur, telles que son emplacement exact ou son volume.

2.1 Attributs de l'image

Les *attributs* d'une image se définissent comme les caractéristiques extraites de l'image de référence que l'on utilise pour guider le recalage. Ce sont les éléments que l'on voudra, en priorité, faire correspondre avec leurs homologues sur l'image mouvante. Il existe plusieurs types d'attributs, et par conséquent plusieurs méthodes pour les choisir.

Les méthodes géométriques

Ces méthodes se basent sur la recherche des caractéristiques (aussi dites *primitives*) géométriques des images. Il peut s'agir de points, de courbes ou encore de surfaces ; ces dernières sont probablement les primitives les plus couramment utilisées. Le choix des primitives peut être réalisé soit manuellement par un expert, soit par l'algorithme de recalage.

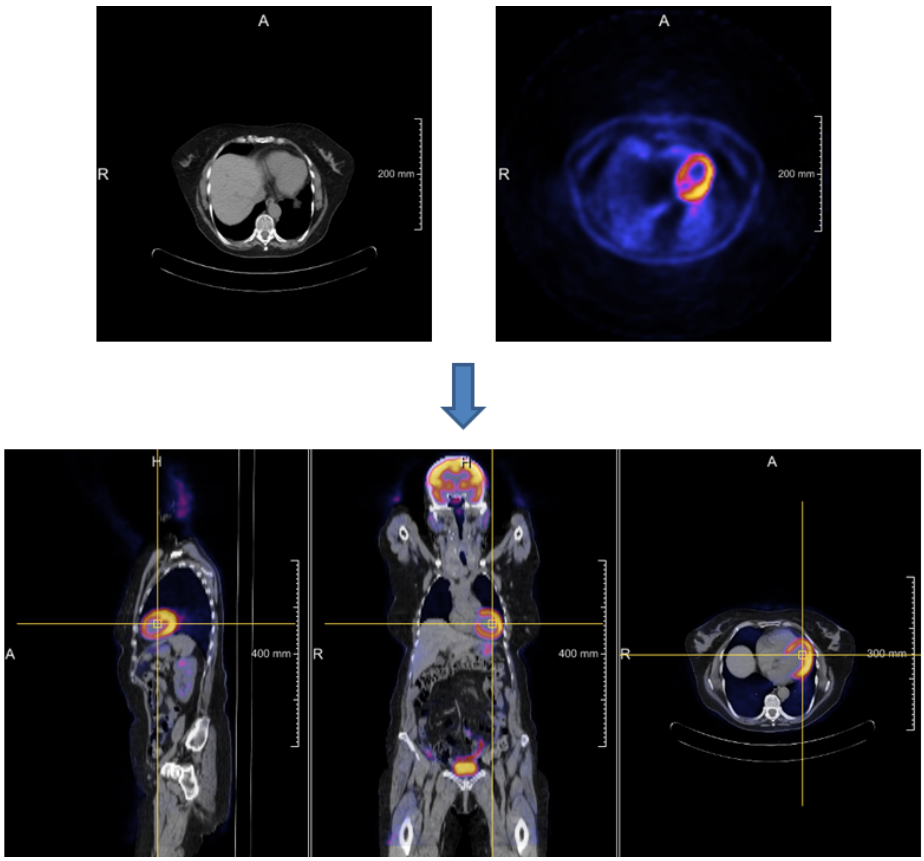


FIGURE 2.2 – Image CT, image PET et leur recalage sous les angles sagittal, frontal et transversal [31]

Pour les points, on peut les choisir de manière manuelle ou laisser l'algorithme les choisir. Il suffit ensuite de trouver la transformation $\hat{T}$ qui appariera le mieux possible les points homologues entre image de référence et image mouvante. Pour les courbes, on peut également les choisir par l'algorithme, mais cela ne donne en général pas de très bons recalages. Pour les surfaces, la meilleure solution est généralement d'appliquer une segmentation à une partie de l'image et d'ensuite déterminer la transformation $\hat{T}$ qui appliquera le mieux possible cette partie de l'image mouvante sur la surface correspondante de l'image de référence [18].

Notons que les primitives choisies auront un impact sur le choix de la fonction de similarité S vue précédemment. Par exemple, pour des primitives points, la fonction de similarité sera clairement une fonction de distance. Il faudra donc la minimiser.

Les méthodes iconiques

Les méthodes iconiques, elles, décomposent l'image en pixels et utilisent leurs niveaux de gris respectifs pour effectuer la mise en correspondance des deux images. On peut donc parler ici de primitives d'intensités, prenant la forme d'un vecteur à trois ou quatre composantes : la position du point (2D ou 3D) et l'intensité lui correspondant [18]. Comme les méthodes iconiques ne nécessitent aucune extraction de caractéristiques géométriques, elles peuvent être appliquées de manière totalement automatique, par algorithme.

Les méthodes hybrides

Les méthodes géométriques et iconiques possèdent chacune leurs avantages et inconvénients. Ceux-ci sont repris dans la table 2.1. Certains chercheurs en sont ainsi venus, pour pallier aux désavantages de l'une comme de l'autre, à imaginer des méthodes utilisant différents attributs dans le but d'obtenir des algorithmes plus robustes. Nous pouvons classer ces techniques en trois grande famille, que nous décrivons brièvement ici.

	Méthodes géométriques	Méthodes iconiques
Avantages	– Charge calculatoire réduite car on ne s'intéresse qu'à un sous-ensemble de l'image – Caractéristiques géométriques plus discriminantes que les caractéristiques d'intensité	– Utilisation de toute l'information donnée par l'image et donc pas de pré-traitement nécessaire – Complètement automatique
Inconvénients	– Imprésicion des primitives extraites – Souvent manuel – Précision du recalage garantie uniquement dans le voisinage des primitives – Impossibilité de définir ce que sont des bonnes ou mauvaises primitives – Points des courbes/surfaces indiscernables	– Coût calculatoire car on considère chaque pixel – Caractéristiques d'intensité peu informatives (la mise en correspondance des images via l'intensité n'est pas si évidente) – Optimisation plus difficile (présence de minima locaux)

TABLE 2.1 – Avantages et inconvénients des méthodes géométriques et iconiques [18]

1. Combiner différentes primitives géométriques :
   Un avantage ici est qu'on peut augmenter le nombre de primitives géométriques et pallier ainsi au désavantage des méthodes géométriques concernant la précision du recalage, qui n'est souvent présente que dans le voisinage des primitives.

2. Utiliser l'intensité mais aussi d'autres informations qui en découlent :
   Le but de cette technique est de tirer un maximum d'informations des caractéristiques des images afin de faciliter leur mise en correspondance. En effet, nous avons vu que les caractéristiques d'intensité ne sont pas toujours suffisantes pour obtenir un recalage évident. Un exemple classique est d'utiliser l'information obtenue en dérivant l'image.

3. Combiner l'approche géométrique et l'approche iconique :
   Par exemple en sélectionnant des primitives géométriques et en calculant l'intensité des points qui les composent, on récupère plus d'information et donc de précision lors du recalage, et ce tout en gardant le faible coût calculatoire de l'approche géométrique. De plus, on peut ainsi distinguer les points des primitives géométriques puisqu'on calcule également leur intensité.

2.2 Fonctions de similarité

Comme déjà précisé, la fonction (ou critère) de similarité S dépend des attributs de l'image. Nous allons donc tout d'abord explorer les différentes fonctions de similarité possibles pour l'approche géométrique, puis ensuite pour l'approche iconique. Les listes de critères cités ne sont pas exhaustives, elles reprennent les plus couramment utilisés.

Critères de similarité géométriques

Pour mesurer la similarité entre des points homologues, la fonction distance euclidienne est en général utilisée. En effet, elle permet souvent d'obtenir une solution analytique pour les paramètres de la transformation $\hat{T}$ recherchée dans la phase d'optimisation. Une telle fonction de similarité est alors définie par

$$S : \Omega_R \times \Omega_M \longrightarrow \mathbb{R}^+ : (x_R, x_M) \longmapsto \sqrt{\sum_{i=1}^{d} (x_{R_i} - x_{M_i})^2},$$

où Ω_R est le domaine de l'image de référence, Ω_M le domaine de l'image mouvante, d est la dimension des images à recaler, $x_R = (x_{R_1}, ..., x_{R_d})$ et $x_M = (x_{M_1}, ..., x_{M_d})$.

Pour mesurer la similarité entre des courbes ou des surfaces, c'est-à-dire des ensembles de points, il nous faut toujours calculer une distance, mais ici quatre possibilités sont couramment utilisées [35] :

- Il est possible de se ramener au problème de distance entre deux points ; cette technique se retrouve en fait dans un algorithme de recalage appelé ICP (*Iterative Closest Point*).
- On peut considérer le carré de la distance entre un point de la primitive à recaler et le point le plus proche de la primitive de l'image de référence dans la direction du centroïde de celle-ci (le centroïde peut être considéré comme le centre de masse d'un ensemble de points au niveau de leur intensité).
- Une autre approche est le calcul de ce qu'on appelle des cartes de distance. Pour construire une telle carte, on commence par mettre l'intensité de tous les pixels correspondant au contour d'une primitive de l'image de référence à zéro. On calcule ensuite la distance entre chaque point et le point de contour le plus proche ; on peut envisager différentes distances pour cela. Ensuite, pour utiliser la carte, il suffit de calculer la moyenne des valeurs de la carte de distance qui sont superposées avec les contours de l'image mouvante. On obtient ainsi une mesure de distance entre deux ensembles de points.
- Finalement, il est possible d'utiliser la distance de Hausdorff. La fonction de similarité S devient alors

  $$S(\Gamma_R, \Gamma_M) = \max \left(\sup_{x_R \in \Gamma_R} \inf_{x_M \in \Gamma_M} dist(x_R, x_M), \sup_{x_M \in \Gamma_M} \inf_{x_R \in \Gamma_R} dist(x_R, x_M) \right),$$

  où Γ_R est un ensemble de points de l'image de référence, Γ_M un ensemble de points de l'image mouvante et *dist* est une distance quelconque, souvent la distance euclidienne.

Critères de similarité iconiques

Dans le cas de l'approche iconique, on cherche à mesurer une similarité entre des intensités. Cependant, on ne cherche pas à obtenir une forte similarité entre des points séparément mais bien entre deux ensembles de points. On va donc s'intéresser ici au type de relation qu'il peut y avoir entre les intensités de l'image de référence et les intensité de l'image mouvante de manière groupée :

il nous faut trouver une relation valable pour toute l'image et non plus pour une partie seulement comme dans le cas géométrique. Cette relation peut être considérée comme *hypothèse* du critère de similarité : en fonction des images à recaler, on fait une supposition sur le type de relation qui lie leurs intensité, ce qui nous fournit un critère de similarité. Nous allons donc ici développer quelques manières de mesurer la similarité entre deux images mises en correspondance grâce aux intensités respectives de leurs points [35].

– *La relation identité :*
La relation identité est d'application pour le recalage monomodal. En effet, on peut dès lors supposer que les intensités des deux images (représenant un même objet) seront approximativement liées par la relation identité. Ainsi, les systèmes reposant sur cette hypothèse tâchent de minimiser des fonctions de similarité basées sur la différence entre les intensités des images. On utilise notamment souvent le critère des moindres carrés, c'est-à-dire la norme $\mathcal{L}_2$. En gardant les notations utilisées précédemment, la fonction de similarité S est alors de la forme

$$S(I, T(J)) = \sum_{s \in \Omega} \big(I(s) - T(J(s))\big)^2,$$

où Ω est l'intersection des domaines de I et de $T(J)$, c'est-à-dire que

$$\Omega = \{s | s \in \Omega_R \text{ et } s \in T(\Omega_M)\}.$$

Ce critère doit bien sûr être minimisé et non maximisé. D'autres critères peuvent également être utilisés, notamment la différence absolue des intensités, à savoir la norme $\mathcal{L}_1$.

– *La relation affine :*
En réalité, l'hypothèse d'une relation identité entre intensités n'est pas toujours vérifiée, car les intensités dépendent fortement des appareils de mesure. On peut donc faire l'hypothèse d'une relation affine entre les intensités de nos deux images, qui correspondrait à une remise à l'échelle des intensités. Dès lors, une bonne mesure de similarité est le coefficient de corrélation linéaire. En effet, il étudie par définition la qualité d'un ajustement affine [10]. La fonction de similarité S est dans ce cas définie par

$$S(I, T(J)) = \frac{1}{\sigma_I \sigma_{T(J)}} \sum_{s \in \Omega} \big(I(s) - \mu_I\big)\big(T(J(s)) - \mu_{T(J)}\big),$$

où μ_I et $\mu_{T(J)}$ représentent respectivement les intensités moyennes des images I et $T(J)$ et σ_I et $\sigma_{T(J)}$ les écarts-types des intensités. Ce coefficient est compris entre -1 et 1. Plus il est proche de -1 ou 1, plus la corrélation est forte. Lorsqu'on optimisera cette fonction de similarité, on voudra donc qu'elle soit proche de ± 1.

– *La relation fonctionnelle :*
La relation affine que nous venons de voir est souvent très bien adaptée pour le recalage monomodal. Cependant, cela ne marche plus du tout pour le recalage multimodal car les images à recaler sont trop différentes. L'hypothèse à formuler quant à la relation existant entre les deux images est alors simplement une hypothèse de relation fonctionnelle quelconque, c'est-à-dire qu'à chaque intensité d'une image peut être associée seulement une intensité d'une seconde image. Plusieurs critères de similarité peuvent convenir pour cette hypothèse. Ils dépendent généralement des modalités des images à recaler puisque chaque type de recalage multimodal sera caractérisé par une relation fonctionnelle particulière. Par exemple, pour le recalage IRM/PET-scan, le critère de Woods est souvent utilisé. Il se base sur la notion de *dispersion* : il mesure, pour une valeur d'intensité donnée dans une image donnée, la

dispersion des pixels possédant cette intensité dans une seconde image. Cette dispersion est supposée faible pour l'ensemble des intensités lorsque les images sont recalées. Il faudra donc ici minimiser la fonction de similarité S définie par

$$S(I, T(J)) = \sum_j p_j \frac{\sigma_{I|j}}{\mu_{I|j}},$$

où p_j est la probabilité qu'un pixel de J ait l'intensité j, $\mu_{I|j}$ et $\sigma_{I|j}$ sont respectivement la moyenne et l'écart-type de l'intensité des pixels de I correspondant aux pixels de J ayant pour intensité j.

– *La relation de dépendance :*
Une dernière classe de relations possible, encore plus générale que celle des fonctions, est celle des dépendances. On considère alors les images comme des variables aléatoires et, dans le cas du recalage, on cherche à calculer la dépendance entre deux de ces variables. Pour ce faire, deux approches sont envisageables. Premièrement, on peut construire l'histogramme conjoint, c'est à dire l'histogramme des intensités des deux images en même temps. Cela revient en fait à sommer les histogrammes de chaque image. Les critères de similarité utilisés sont alors basés sur la dispersion de l'histogramme conjoint : plus elle est faible, meilleur est le recalage puisque la dépendance entre les deux images est grande. Deuxièmement, on peut considérer la distance entre la distribution conjointe des images et la distribution conjointe qu'elles auraient sous l'hypothèse d'indépendance. Les critères de similarité sont donc ici des mesures de distance entre distributions.

2.3 Modèles de transformation

Il nous faut à présent évoquer un point essentiel du recalage d'images, à savoir les différentes formes que peuvent prendre la transformation de l'image mouvante. Ces transformations peuvent être regroupées en différentes classes, selon qu'elles soient linéaires ou non, globales ou locales. Les modèles non linéaires donneront lieu à des déformations de l'image, contrairement aux modèles linéaires. Les transformation globales s'appliqueront à l'image tout entière tandis que les transformations locales ne feront effet que sur une partie de l'image. La figure 2.3 compare les différentes transformations que nous évoquerons par la suite, pour le cas global et le cas local.

Transformations rigides

Les transformations rigides sont les plus simples. Elles se contentent de transformer l'image mouvante par des opérations de *translation* et de *rotation*. Elles sont donc souvent globales. Mathématiquement, une transformation rigide pour un recalage d-dimensionnel est définie par

$$T : p \longmapsto Rp + t,$$

où $p = (x_1, ..., x_d)$ est un point de l'image mouvante, $R \in \mathbb{R}^{d \times d}$ est la matrice de rotation et $t \in \mathbb{R}^{d \times 1}$ est le vecteur de translation [19].

Par exemple, pour $d = 3$ et en notant respectivement R_x, R_y et R_z les matrices de rotation

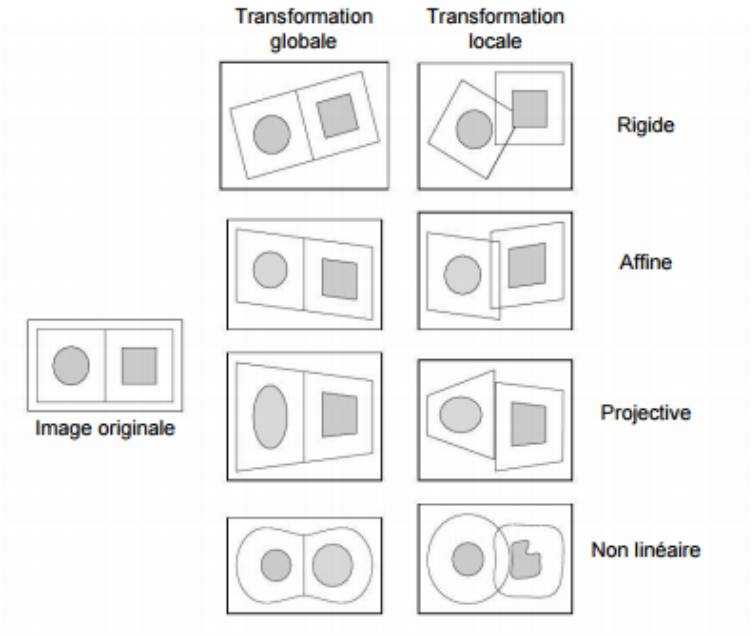


FIGURE 2.3 – Différentes transformations pour le cas global et le cas local (2D) [35]

autour des axes x, y et z, et α, β et γ les angles de rotation correspondants, nous aurons

$$\begin{aligned} R &= R_x R_y R_z \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} \cos\beta\cos\gamma & \cos\alpha\sin\gamma + \sin\alpha\sin\beta\sin\gamma & \sin\alpha\sin\gamma - cos\alpha\sin\beta\cos\gamma \\ -\cos\beta\sin\gamma & \cos\alpha\cos\gamma - \sin\alpha\sin\beta\sin\gamma & \sin\alpha\cos\gamma + \cos\alpha\sin\beta\sin\gamma \\ \sin\beta & -\sin\alpha\cos\beta & \cos\alpha\cos\beta \end{pmatrix} \end{aligned}$$

et

$$t = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix},$$

où t_x, t_y et t_z représentent respectivement les translations selon les axes x, y et z de l'image 3D.

Transformations affines

Bien souvent, de par leur simplicité, les transformations rigides ne suffisent pas pour obtenir un bon recalage. Les transformations affines viennent apporter une amélioration aux transformations rigides, à savoir une composante de cisaillement, c'est-à-dire d'étirement de l'image, mais également une composante de mise à l'échelle. Elles conservent cependant toujours le parallélisme, comme illustré par la figure 2.3 [53]. De même que les transformations rigides, les transformations affines sont souvent de type global. Mathématiquement, en gardant les notations précédemment établies

pour les transformations rigides, une transformation affine en dD est donc définie par

$$T : p \longmapsto ECRp + t,$$

où $E \in \mathbb{R}^{d\times d}$ est la matrice de mise à l'échelle et $C \in \mathbb{R}^{d\times d}$ est la matrice de cisaillement. Cependant, par simplicité, on utilise souvent les coordonnées homogènes pour décrire la transformation, qui permettent ainsi d'obtenir une transformation de la forme

$$T : \dot{p} \longmapsto A\dot{p},$$

où $\dot{p} = (x_1, ..., x_d, 1)$ et $A \in \mathbb{R}^{(d+1)\times(d+1)}$ correspond à l'ensemble des transformations à effectuer.

En reprenant notre exemple d'une simple transformation rigide avec $d = 3$, nous créerons donc une matrice de translation M_t ainsi qu'une nouvelle matrice de rotation M_r (cette fois-ci de dimension 4), données par

$$M_t = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ et } M_r = \begin{pmatrix} & & & 0 \\ & R & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

et nous obtiendrons ainsi

$$A = M_t M_r.$$

Pour le cas d'une transformation affine, nous aurons donc

$$A = M_e M_c M_t M_r,$$

avec

$$M_e = \begin{pmatrix} e_x & 0 & 0 & 0 \\ 0 & e_y & 0 & 0 \\ 0 & 0 & e_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ et } M_c = \begin{pmatrix} 1 & c_{yx} & c_{zx} & 0 \\ -c_{yx} & 1 & c_{zy} & 0 \\ -c_{zx} & -c_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

où e_x, e_y et e_z sont les paramètres de la mise à l'échelle tandis que c_{yx}, c_{zx} et c_{zy} sont les paramètres du cisaillement [11].

Transformations projectives

Contrairement aux transformations affines, les transformations projectives ne conservent plus le parallélisme. Cependant, elles exigent toujours que l'image d'une droite soit une droite. Leur but est de prendre en compte l'effet de perspective dans l'image. Ce sont donc des transformations qui permettent de transformer les images mouvantes en images de dimension inférieure. L'exemple typique est le recalage 3D/2D, lorsque l'on souhaite recaler une image 3D sur une image 2D. C'est cependant assez peu courant [3].

Transformations non linéaires

Les transformations dites linéaires, c'est-à-dire celles décrites précédemment, produisent des changements peu conséquents, assez superficiels, sans déformations. Elles ne sont donc en général adaptées qu'à des recalages d'images où les images à recaler sont déjà assez semblables : typiquement, des images d'un même patient. Cela ne suffit généralement pas lorsque l'on souhaite

superposer des images de différents patients. On a dès lors besoin de transformations non linéaires. Les transformations non linéaires sont des transformations qui provoquent une réelle déformation, c'est-à-dire qu'elles transforment les lignes droites en courbes [5]. Elles sont appliquées localement ; la transformation appliquée en un point de l'image peut donc être différente de celle appliquée aux points voisins.

Un exemple typique de transformation non linéaire est la transformation polynomiale [35], où la transformation T est simplement un polynôme. Pour un polynôme d'ordre 2 et une image 3D par exemple, elle est donc décrite par

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,10} \\ a_{2,1} & a_{2,2} & \dots & a_{2,10} \\ a_{3,1} & a_{3,2} & \dots & a_{3,10} \\ 0 & 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} x^2 & y^2 & z^2 & xy & xz & yz & x & y & z & 1 \end{pmatrix}^t,$$

où (x', y', z') sont les nouvelles coordonnées (après transformation) du point (x, y, z) de l'image mouvante et les $a_{i,j}$ $(1 \leq i \leq 3,\ 1 \leq j \leq 10)$ sont les coefficients du polynôme. Les polynômes les plus utilisés sont ceux d'ordre 2, 3 et 4. Au-delà de 4, ils ne sont en général plus adaptés au recalage d'images.

Un autre exemple courant est celui des bases de fonctions : on utilise comme transformation T une combinaison linéaire de d'un nombre l de fonctions $\theta_i(x_1, ..., x_d)$ $(1 \leq i \leq l)$, où le choix de l est libre. Ces fonctions peuvent notamment être des polynômes, des fonctions splines, des fonctions trigonométriques, etc.

Il existe en réalité un grand nombre de transformations non linéaires applicables au recalage d'images. Nous citerons entre autres les modèles de fluides, les modèles de flux optique et les modèles élastiques [42].

2.4 Interpolation et rééchantillonnage

Un problème que nous n'avons pas encore considéré jusqu'ici est celui de l'échantillonnage des images que l'on souhaite recaler [2]. En effet, avant d'appliquer une transformation T à un point de coordonnées $(x_1, ..., x_d)$ de l'image mouvante, où d est la dimension de l'image, ces coordonnées sont entières, ou du moins uniformément espacées selon chaque dimension i $(1 \leq i \leq d)$ [35]. Cependant, après transformation, il est rare qu'elles le soient toujours. En fait, il n'y a que pour le cas de certaines translations ou d'une rotation d'un angle multiple de 90° que les nouvelles coordonnées resteront sur la grille de l'image. Si T est plus complexe que ça, le point $(x_1, ..., x_d)$ sera envoyé à une position réelle $(x'_1, ..., x'_d)$ hors de la grille de l'image. Or ce qui nous intéresse est de connaitre les intensités aux noeuds de la grille de l'image transformée. La solution à ce problème est l'interpolation. Son but est de déterminer l'intensité d'un point sur base de points voisins. L'interpolation calcule une fonction continue passant par un échantillon de points qui sont en fait les intensités connues aux points à coordonnées réelles obtenues par T. L'étape suivante est ensuite le rééchantillonnage, qui discrétise à nouveau la fonction obtenue, cette fois aux neouds de la grille dont on cherche l'intensité. Cette façon de faire est connue sous le nom d'*approche directe*.

Cependant, en pratique, nous n'utiliserons pas la transformation T de manière directe. Nous passerons plutôt par la transformation inverse T^{-1}, afin d'interpoler à partir de points ayant une répartition régulière. Au lieu de rééchantillonner des points à coordonnées entières, nous ré-échantillonnons ainsi des points à coordonnées réelles ; c'est l'*approche indirecte*. Notons comme

précédemment notre image mouvante J, et plus particulièrement notre image mouvante avant transformation par J_{avant} et notre image mouvante après transformation par J_{apres}. La figure 2.4 illustre alors cette façon de procéder. La partie supérieure illustre l'interpolation par approche directe : les points de l'image mouvante sont transformés par T deviennent ainsi des points qui ne sont pas des noeuds de la grille image. L'interpolation en un noeud est alors réalisée sur base de points non uniformément espacés. La partie inférieure de l'image, elle, illustre l'approche indirecte : les noeuds de la grille sont transformés par T^{-1} et l'intensité de ces points est interpolée sur base de points uniformément espacés de l'image avant transformation.

Une fois en possession d'une transformation T à une certaine itération de l'algorithme de recalage, le processus d'iterpolation peut donc être établi par les étapes suivantes :

1. Application de T^{-1} aux points uniformément espacés de l'image mouvante (i.e., les noeuds de la grille), on obtient ainsi des points non nécessairement uniformément espacés ;
2. Recherhe d'une fonction continue passant par les noeuds de la grille de l'image avant transformation (passage du discret au continu) ;
3. Rééchantillonnage de cette fonction aux points trouvés à l'étape 1, on obtient ainsi l'intensité des neouds de la grille de l'image transformée (passage du continu au discret).

Plusieurs méthodes sont couramment utilisées dans le cadre du traitement d'images. La plupart d'entre elles sont des méthodes linéaires, c'est-à-dire telles que la somme de deux fonctions interpolées est égale à l'interpolation de la somme de deux fonctions. Ici, nous nous concentrerons sur les modèles de la forme

$$f(x) = \sum_{k \in \mathbb{Z}^d} f_k \phi(x - k) \quad \forall\, x = (x_1, ..., x_d) \in \mathbb{R}^d, \tag{2.2}$$

où une valeur interpolée $f(x)$ en une coordonnée x réelle est donnée comme une combinaison linéaire des échantillons f_k évalués aux coordonnées entières $k = (k_1, ..., k_d) \in \mathbb{Z}^d$, les poids étant donnés par les valeurs de la fonction $\phi(x - k)$. Nous avons donc ici fait l'hypothèse simplificatrice que les noeuds de la grille possèdent des coordonnées entières ; si en réalité ce n'est pas vérifié, il est

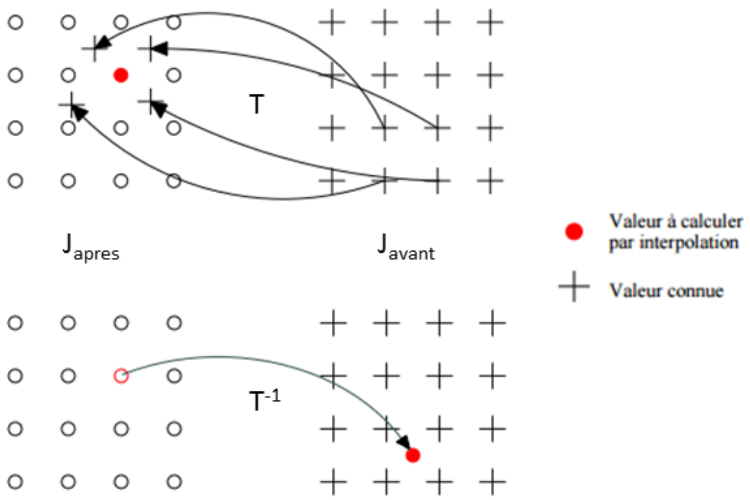


FIGURE 2.4 – Différence entre les approches par transformations directe et indirecte [35]

facile de se ramener à ce cas grâce au spacing uniforme de l'image (cf. Chapitre 1, Section 1.2.4). Dans les modèles de ce genre, il est évident que la fonction ϕ doit respecter certaines conditions : premièrement, elle doit s'annuler lorsqu'elle est évaluée en un nombre entier, sauf si ce nombre entier est k ; dans ce cas elle doit valoir 1 pour vérifier $f(k) = f(k)$.

Cette condition est très contraignante, et nous préférons dès lors les modèles décrits par

$$f(x) = \sum_{k \in \mathbb{Z}^d} c_k \phi(x - k) \quad \forall\, x = (x_1, ..., x_d) \in \mathbb{R}^d, \tag{2.3}$$

où les coefficients c_k ne sont plus forcément les échantillons f_k. Cette formulation permet un plus large choix de fonctions ϕ. Une fois ces fonctions choisies, le problème d'interpolation se résume à la recherche des coefficients c_k. Pour déterminer ces coefficients, considérons l'équation (2.3) pour des entiers uniquement :

$$f(k_0) = \sum_{k \in \mathbb{Z}^d} c_k p_{k_0 - k} \quad \forall\, k_0 \in \mathbb{Z}^d, \tag{2.4}$$

où $p_k = \phi(k)$. Nous obtenons donc un système linéaire

$$f = Pc,$$

où f est le vecteur des échantillons de données, c le vecteur de coefficients et P la matrice d'évaluations des fonctions ϕ. Ce système est de dimension infinie, mais en pratique nous ne possédons pas un échantillon de taille infinie. De plus, nous pouvons contraindre les fonctions ϕ à avoir un support fini afin d'obtenir une matrice P bande. Le système peut alors être résolu via, par exemple, une factorisation LU.

Il est cependant possible d'obtenir les coefficients c_k différemment, en remarquant que (2.4) décrit en fait une convolution discrète

$$f_{k_0} = (c * p)_{k_0}.$$

En convoluant des deux côtés par p^{-1}, nous obtenons

$$c_{k_0} = ((p)^{-1} * f)_{k_0}. \tag{2.5}$$

C'est cette méthode que nous utiliserons dans ce travail. Nous y reviendrons plus en détails dans le Chapitre 3 pour le cas des B-splines.

Il nous faut à présent nous intéresser au choix des fonctions ϕ. Enormément de possibilités sont proposées dans la littérature. Nous décrirons ici les plus populaires. Nous allons néanmoins, avant cela, étudier les propriétés que devraient posséder les fonctions ϕ pour que l'on obtienne une interpolation efficace [2] :

- **Support fini :**
  Il est essentiel d'utiliser des fonctions ϕ à support fini, c'est-à-dire des fonctions non nulles uniquement sur un hypercube H^n. En effet, si ϕ était à support infini, cela demanderait beaucoup trop d'évaluation de fonctions et l'interpolation ne serait donc pas du tout optimale d'un point de vue temps de calcul.

- **Séparabilité :**
  Supposons que les fonctions ϕ soient bien à support fini. Imaginons une interpolation en une dimension requiérant un nombre fini p d'évaluations de fonctions. Dès lors, en deux dimensions,

l'interpolation demanderait p^2 évaluations de fonctions et en 3D, p^3. Ce nombre peut donc vite devenir énorme. Pour pallier à ce problème, on peut utiliser des fonctions ϕ respectant la propriété de séparabilité, i.e.,

$$\phi(x) = \prod_{i=1}^{d} \phi(x_i) \quad \forall\, x = (x_1, ..., x_d) \in \mathbb{R}^d.$$

L'avantage est que l'interpolation peut alors être réalisée dimension par dimension. Pour le cas 2D, par exemple, on interpole alors en 1D ligne par ligne, obtenant ainsi un point interpolé par ligne. La même procédure est ensuite utilisée par colonne sur les points résultants, comme illustré à la figure 2.5. Le nombre d'évaluations est donc réduit à dp, au lieu de p^d.

- **Symétrie :**
  La symétrie est une propriété très désirée également pour les fonctions ϕ. C'est lié à la qualité d'interpolation lorsque l'on utilise l'équation (2.5) pour résoudre le problème, mais nous n'entrons pas ici dans les détails. Le lecteur intéressé peut se référer à [2].

- **Partition de l'unité :**
  Imaginons que tous les points à interpoler aient exactement la même intensité, i.e., $f(k) = f_0$ $\forall\, k \in \mathbb{Z}^d$. Intuitivement, il semble alors logique que la fonction interpolante continue soit constante. Pour l'équation (2.2), nous obtenons alors

$$1 = \sum_{k \in \mathbb{Z}^d} \phi(x - k) \quad \forall\, x \in \mathbb{R}^d.$$

  Similairement, il est possible de déduire la partition de l'unité pour les fonctions ϕ dans l'équation (2.3) [2].

Recensons à présent quelques-unes des principales fonctions ϕ utilisées pour l'interpolation en traitement d'images.

Plus proche voisin

La méthode du plus proche voisin utilise comme fonction ϕ une simple fonction de type "palier" :

$$\phi(x) = \begin{cases} 1 & \text{si } \frac{-1}{2} \leq x < \frac{1}{2} \\ 0 & \text{sinon.} \end{cases}$$

Cette fonction est symétrique (sauf en un point), elle a un support local, elle respecte la partition de l'unité et elle est séparable. Elle produit par contre une fonction interpolante discontinue, ce

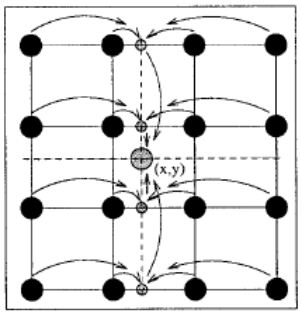


FIGURE 2.5 – Interpolation effectuée dimension par dimension [26]

qui peut poser problème dans les algorithmes de recalage. Elle est extrêmement simple, ce qui constitue à la fois un avantage et un inconvénient. En effet, cela permet une implémentation facile et efficace, mais on perd énormément en qualité d'interpolation. Concrètement, cela signifie que le point dont on cherche l'intensité prend simplement l'intensité du point connu le plus proche. Elle n'utilise donc qu'une seule information de l'échantillon [2].

Interpolation linéaire

La fonction ϕ utilisée pour l'interpolation linéaire est une fonction de type "tente" donnée par

$$\phi(x) = \begin{cases} 1 - |x| & \text{si } |x| < 1 \\ 0 & \text{sinon.} \end{cases}$$

C'est une fonction symétrique, séparable, à support local et qui respecte la partition de l'unité. A nouveau, elle est très simple et la méthode est donc très rapide. C'est l'implémentation la plus basique qui fournit un interpolant continu. L'interpolant n'est cependant pas différentiable. Elle utilise deux échantillons pour produire une valeur interpolée. En deux dimensions, cette méthode réalise donc quatre évaluations et en trois dimensions, six. Elle est fréquemment utilisée de par sa simplicité et sa continuité [2].

Interpolation quadratique

Les polynômes sont très utilisés en interpolation. Les deux cas précédemment vus sont d'ailleurs respectivement des polynômes constants et linéaires. Les polynômes quadratiques peuvent également être utilisés sous certaines conditions pour qu'ils vérifient les quatre propriétés qu'une fonction ϕ devrait respecter. Similairement au plus proche voisin, ils ne peuvent cependant être totalement symétriques. Ces polynômes ne sont pas les plus utilisés ; malgré leur continuité et leur différentiabilité, l'ordre supérieur (polynômes cubiques) offre souvent un meilleur compromis entre qualité d'interpolation et coût de calcul.

B-splines

Une des classes de fonctions les plus utilisées pour ϕ est la classe des B-splines. Ces fonctions forment une base pour les polynômes splines. Les B-splines constituant l'objet de ce travail, nous y reviendrons plus longuement dans le chapitre suivant, consacré à l'interpolation par B-splines.

2.5 Méthodes d'optimisation

Rappelons que le recalage d'images consiste avant toute chose en une *optimisation*. Une dernière étape essentielle est donc le choix de la méthode d'optimisation. A nouveau, en fonction du type d'images que l'on souhaitera recaler, certaines méthodes seront plus adaptées que d'autres.

Mais pour commencer, formulons de manière mathématique un problème classique d'optimisation. Par définition, l'optimisation consiste à trouver le minimum ou le maximum d'une fonction possiblement soumise à des contraintes sur ses variables [11]. En réalité, il n'y a en général pas de contraintes intervenant dans les problèmes de recalage d'images. Notons $f : \mathbb{R}^d \longrightarrow \mathbb{R}$ la fonction objectif, c'est-à-dire la fonction à optimiser, et x le vecteur des variables. On peut dès lors définir les problèmes d'optimisation sans contraintes par

$$\min_{x \in \mathbb{R}^d} f(x) \tag{2.6}$$

pour le cas des minimisations [36].

Dans les sous-sections qui suivent, nous décrivons brièvement les différentes familles de méthodes d'optimisation utilisées dans le cadre du recalage d'images [35]. Le choix de la méthode d'optimisation dépend de la forme de la fonction objectif, c'est-à-dire du critère de similarité choisi.

Méthodes directes

Bien que rares, il existe des cas où le problème d'optimisation du recalage admet une solution exacte. Cela peut notamment arriver pour des recalages de type géométrique. Par exemple, la recherche d'une transformation rigide ou affine à partir de primitives géométriques points et utilisant comme fonction de similarité la somme des carrés des erreurs sur l'ensemble des points admet une solution exacte.

Méthodes exhaustives

Le principe des méthodes exhaustives est simple : il s'agit d'échantillonner à intervalles réguliers l'ensemble des variables et de retenir la meilleure solution, la solution optimale. Elles ne se basent donc que sur l'évaluation de la fonction objectif en un certain nombre de points. Ces méthodes sont cependant peu utilisées car elles sont coûteuses en temps de calcul.

Méthodes itératives

Contrairement aux méthodes exhaustives, les méthodes itératives ont généralement également besoin de calculer le gradient et/ou le hessien de la fonction objectif, c'est-à-dire du critère de similarité. Leur particularité est bien sûr leur caractère itératif : elles cherchent à s'approcher le plus possible de la solution au fur et à mesure des itérations. Elles s'arrêtent une fois le critère d'arrêt défini atteint. Ces méthodes sont les plus utilisées dans le cadre du recalage. Parmi les nombreuses méthodes itératives, on distingue deux grandes familles : les méthodes de recherche linéaire et les méthodes de région de confiance. Dans le cadre du recalage, ce sont principalement les méthodes de recherche linéaire qui sont utilisées.

Méthodes stochastiques

Nous avons vu, dans la Section 2.2, que les critères de similarités pouvaient être basés sur les statistiques, considérant dès lors les images comme des variables aléatoires. Rappelons par exemple le cas du coefficient de corrélation linéaire, qui mesure la qualité d'un ajustement affine. C'est dans ce cadre que les méthodes stochastiques sont d'application ; la fonction objectif est alors une fonction aléatoire.

Autres méthodes

Si le recalage d'images consiste toujours en une optimisation, celle-ci n'est pas toujours explicite. Nous entendons par là que certains algorithmes de recalage n'utilisent aucune méthode d'optimisation car le processus de recalage en lui-même est déjà défini pour itérer dans la bonne direction : l'optimisation d'un certain critère de similarité est alors intrinsèque au processus. Un exemple classique est celui où le recalage d'images est défini par une équation au dérivées partielles décrivant l'évolution de l'image mouvante. Comme nous le verrons plus loin, c'est d'ailleurs ainsi que fonctionnera l'algorithme étudié ici.

2.6 Conclusion

Dans ce chapitre, nous avons abordé les principaux domaines du traitement d'images, et plus particulièrement le recalage d'images. Nous tentons ici de résumer en quelques phrases les éléments importants d'un processus général de recalage.

Le problème de recalage est un problème d'*optimisation* dont la fonction objectif est appelée *critère de similarité* et mesure donc la similarité entre les images à recaler. D'une itération à l'autre, l'image mouvante tente d'approcher l'image fixe via l'application d'une *transformation* qui peut prendre diverses formes. Le recalage peut être basé soit sur des caractéristiques géométriques extraites des images qu'il faut alors mettre en correspondance entre image fixe et image mouvante, soit sur des caractéristiques liées à l'intensité des pixels. Ce choix influence d'ailleurs le choix de la fonction de similarité. Finalement, il est important de remarquer que nous ne pouvons pas appliquer la transformation telle quelle à l'image mouvante ; une interpolation préalable est indispensable.

La première étape d'un recalage est donc le choix de la méthode. Cela comprend le type d'attributs, le critère de similarité, la méthode d'interpolation, la méthode d'optimisation et le type de transformation. Ensuite, on peut lancer le processus itératif, dont une itération quelconque est donnée par les étapes suivantes :

1. Calcul du critère de similarité quantifiant la ressemblance entre les deux images ;
2. Calcul d'une nouvelle transformée T ;
3. Interpolation de l'image mouvante pour obtenir une image continue ;
4. Rééchantillonnage de l'image mouvante pour obtenir les intensités de l'image mouvante transformée par T.

Chapitre 3

B-splines et interpolation pour le recalage d'images

Nous nous intéressons dans ce chapitre de plus près à la théorie de ce que nous mettrons en application plus loin dans ce mémoire, à savoir tout ce qui a trait à l'interpolation dans le cadre du recalage d'images. Plus précisément, la technique qui retient ici notre attention est l'interpolation par bases splines, aussi appelées B-splines.

Commençons par rappeler le problème d'interpolation que nous avions défini à la Section 2.4. Soit d la dimension de l'image à interpoler. Nous cherchons alors à interpoler l'image mouvante par une fonction

$$f(x) = \sum_{k \in \mathbb{Z}^d} c_k \phi(x - k) \quad \forall\, x \in \mathbb{R}^d, \tag{3.1}$$

où la fonction ϕ doit dans l'idéal posséder les propriétés de support local, de symétrie, de séparabilité et de partition de l'unité.

La première partie de ce chapitre introduit tout d'abord les B-splines en une dimension, en gardant toujours à l'esprit que nous cherchons des fonctions qui soient de bonnes candidates pour jouer le rôle de ϕ. Nous montrons ensuite comment adapter cette théorie au cas multidimensionnel, avant d'aborder quelques résultats importants. Finalement, un cas particulier des B-splines est envisagé : celui des B-splines cubiques à noeuds uniformément espacés.

Une deuxième partie de ce chapitre est consacrée à la construction concrète et implémentable d'une méthode d'interpolation par B-splines. Nous introduisons d'abord pour cela les bases de la théorie du signal, avant d'adapter ces notions à l'interpolation par B-splines.

3.1 Les B-splines

Nous abordons dans cette section les splines et B-splines, fonctions d'une importance capitale dans ce mémoire. Nous voyons tout d'abord les définitions basiques menant aux B-splines unidimensionnelles puis multidimensionnelles avant d'aborder leurs propriétés les plus intéressantes ainsi qu'un cas particulier de B-splines. Les résultats présentés dans cette section sont inspirés de [43], [45] et [47].

3.1.1 Premières définitions

Pour introduire les B-splines, il nous faut d'abord définir formellement les polynômes et les splines [43].

Définition 3.1 (Espace des polynômes)**.** *L'espace des polynômes d'ordre n est défini par*

$$\mathcal{P}_n = \{p(x) : p(x) = \sum_{i=0}^{n} c_i x^i, \quad c_1, \ldots, c_n, \quad x \in \mathbb{R}\}$$

Théorème 3.1. *Pour un réel a donné, les fonctions*

$$1, (x-a), \ldots, (x-a)^n$$

forment une base de $\mathcal{P}_n$.

Définition 3.2 (Espace des splines)**.** *Soit $[a, b]$ un intervalle fini fermé et soit*

$$\Delta = \{x_i\}_{i=1}^{k} \quad \text{avec } a = x_0 < x_1 < \ldots < x_k < x_{k+1} = b$$

une partition de $[a, b]$ formée de $k+1$ sous-intervalles

$$I_i = [x_i, x_{i+1}[, \quad i = 0, 1, \ldots, k-1 \quad \text{et} \quad I_k = [x_k, x_k + 1].$$

Soit n un entier positif et soit $\mathcal{N} = (n_1, \ldots, n_k)$ un vecteur d'entiers tels que $1 \leq n_i \leq n+1$, $i = 1, 2, \ldots, k$.
Dès lors, on appelle l'espace

$$\begin{aligned}\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta) \ = \{s : \exists\ & s_0, \ldots, s_k \in \mathcal{P}_n \text{ tels que } s(x) = s_i(x)\ \forall\, x \in I_i,\ i = 0, 1, \ldots, k, \\ & \text{et } s_{i-1}^{(j)}(x_i) = s_i^{(j)}(x_i) \text{ pour } j = 0, \ldots, n - n_i,\ i = 1, \ldots, k\}\end{aligned}$$

l'espace des splines d'ordre n définies sur les noeuds $x_1, \ldots, x_k$ de multiplicités respectives $n_1, \ldots, n_k$.

Une fonction spline est donc une fonction polynômiale par morceaux $n - n_i$ fois continûment dérivable en chaque noeud de raccord x_i, $i = 1, \ldots, k$. Le vecteur $\mathcal{N}$ permet ainsi une définition précise de la nature d'une spline via le contrôle du caractère lisse de la spline aux noeuds de raccord. Dès lors, si par exemple $n_1 = n + 1$, il n'y aura aucune contrainte sur le caractère continûment dérivable et la spline pourra très bien être discontinue, et donc non dérivable, au noeud x_1. A l'autre extrême, si $n_1 = 1$, la fonction spline sera aussi lisse qu'elle peut l'être au noeud x_1. Pour l'ordre 3 par exemple, la spline sera deux fois continûment dérivable.

Théorème 3.2. *Soit $K = \sum_{i=1}^{k} n_i$. Alors $\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta)$ est un espace vectoriel linéaire de dimension $n + 1 + K$.*

Maintenant que nous connaissons la dimension de l'espace des splines, il est possible de lui associer une base. Le but ici est de rechercher des fonctions adaptées pour jouer le rôle des fonctions ϕ dans l'équation (3.1) qui, pour rappel, était donnée par

$$f(x) = \sum_{k \in \mathbb{Z}} c_k \phi(x - k) \ \ \forall\, x \in \mathbb{R},$$

pour le cas unidimensionnel que nous étudions ici. Notons que si nous choisissons une base de l'espace des splines comme fonctions ϕ, l'interpolant f sera alors une spline.

De manière logique, nous avons

$$\mathcal{P}_n \subseteq \mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta),$$

c'est-à-dire que tout polynôme d'ordre n est une spline d'ordre n. De plus, par le théorème 3.1, les fonctions

$$1, (x-a), \ldots, (x-a)^n$$

forment une base de l'espace des polynômes d'ordre n, $\mathcal{P}_n$, et peuvent donc également être inclues dans une base de $\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta)$. Par le théorème 3.2, il nous reste donc K éléments à trouver pour compléter la base.

Il est en fait possible de montrer que

$$\left\{\rho_{i,j}(x) = (x - x_i)_+^{n-j}\right\}_{j=0,\, i=0}^{n_i - 1,\, k},$$

où $x_0 = a$, $n_0 = n$ et

$$(x-y)_+^j = (x-y)^j \mu(x-y), \quad j > 0, \quad \mu(x-y) = \begin{cases} 1 & \text{si } x - y \geq 0 \\ 0 & \text{si } x - y < 0 \end{cases},$$

forme une base pour $\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta)$ [43].

Dès lors, une spline s peut s'écrire de manière unique sous la forme

$$\begin{aligned} s(x) &= \sum_{i=0}^{k} \sum_{j=0}^{n_i - 1} c_{ij} \rho_{i,j}(x) \\ &= \sum_{i=0}^{k} \sum_{j=0}^{n_i - 1} c_{ij} (x - x_i)_+^{n-j}. \end{aligned}$$

Cependant, pour beaucoup d'applications numériques, dont le recalage d'images, cette base n'est pas idéale. Rappelons en effet que les propriétés que devraient posséder la fonction ϕ pour un coût de calcul moindre incluent le support local et la symétrie. Celles-ci ne sont pas respectées par les fonctions de base définies ci-dessus, qui ne sont dès lors pas de bonnes candidates. Or il serait possible, à la place, d'utiliser une autre base de $\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta)$, définie localement et permettant ainsi, pour chaque évaluation de l'interpolant, de n'utiliser que quelques éléments de la base. C'est ici qu'interviennent les fonctions B-splines.

Définition 3.3 (Différence divisée)**.** *Soient des points $t_1 \leq \ldots \leq t_{r+1}$ et soit f une fonction. La différence divisée d'ordre r de f sur les points $t_1, \ldots, t_{r+1}$ est alors définie récursivement par*

$$[t_1, \ldots, t_{r+1}]f = \frac{[t_2, \ldots, t_{r+1}]f - [t_1, \ldots, t_r]f}{t_{r+1} - t_1}$$

si $t_1 \neq t_{r+1}$. Si $t_1 = t_{r+1}$, alors

$$[t_1, \ldots, t_{r+1}]f = \frac{f^{(r)}(t_1)}{r!}.$$

Définition 3.4 (B-spline)**.** *Soit*

$$\ldots \leq y_{-1} \leq y_0 \leq y_1 \leq y_2 \leq \ldots$$

une suite de nombres réels. Soient i et $n > 0$ des entiers. La fonction

$$B_i^n(x) = \begin{cases} (-1)^{n+1}[y_i, \ldots, y_{i+n+1}](x-y)_+^n & si\ y_i < y_{i+n+1} \\ 0 & sinon \end{cases},$$

$\forall x \in \mathbb{R}$, est dite B-spline d'ordre n associée aux noeuds $y_i, \ldots, y_{i+n+1}$.

Nous voici donc avec une première définition des B-splines, assez générale. En réalité, nous ne travaillerons pas avec ces fonctions mais bien avec les B-splines *normalisées*, données par

$$N_i^n(x) = (y_{i+n+1} - y_i)B_i^n(x).$$

Les fonctions N_i^n sont donc dites B-splines normalisées d'ordre n associées aux noeuds $y_i, \ldots, y_{i+n+1}$. L'avantage est qu'elles possèdent certaines propriétés intéressantes qui nous seront utiles, telles que la partition de l'unité.

A partir de maintenant, c'est à ces fonctions normalisées que nous ferons référence lorsque nous évoquerons les B-splines. Nous avons à présent tous les outils en main pour définir une nouvelle base de l'espace $\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta)$.

Définition 3.5 (Partition étendue)**.** *Soient $a < x_1 < x_2 < \ldots < x_k < b$ et $1 \leq n_i \leq n+1$, $i = 1, 2, \ldots, k$. Soient*

$$y_1 \leq y_2 \leq \ldots \leq y_{2n+K+2}$$

tels que

$$y_1 \leq \ldots \leq y_{n+1} \leq a, \qquad b \leq y_{n+K+2} \leq \ldots y_{2n+K+2}$$

et

$$y_{n+2} \leq \ldots \leq y_{n+K+1} = \underbrace{x_1, \ldots, x_1}_{n_1}, \ldots, \underbrace{x_k, \ldots, x_k}_{n_k}.$$

Dès lors, $\tilde{\Delta} = \{y_i\}_1^{2n+K+2}$ est une partition étendue associée à $\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta)$, où $\mathcal{N} = (n_1, \ldots, n_k)$ et $\Delta = \{x_i\}_{i=1}^k$.

Théorème 3.3. *Soit $\tilde{\Delta} = \{y_i\}_1^{2n+K+2}$ une partition étendue associée à $\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta)$. Supposons que $b < y_{2n+K+2}$. Pour $i = 1, 2, \ldots, n+K+1$, soient*

$$N_i^n(x) = (-1)^{n+1}(y_{i+n+1} - y_i)[y_i, \ldots, y_{i+n+1}](x-y)_+^n, \qquad a \leq x \leq b,$$

les fonctions B-splines définies sur l'intervalle $[a, b]$. Les fonctions $\{N_i^n\}_{i=1}^{n+K+1}$ forment alors une base pour l'espace $\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta)$.

Remarquons qu'une B-spline N_i^n est définie sur $n+2$ noeuds. La base $\{N_i^n\}_{i=1}^{n+K+1}$ est donc bien *locale*. Notons également que la présence de la condition

$$b < y_{2n+K+2}$$

est dûe au fait que l'intervalle $I_k = [x_k, x_{k+1}]$, i.e. le dernier sous-intervalle de $[a, b]$, est fermé. En effet, pour que la B-spline N_{n+K+1}^n appartienne bien à l'ensemble $\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta)$, il faut, par la définition 3.2, qu'elle soit un polynôme d'ordre n sur tout intervalle I_i, $i = 0, \ldots, k$. Or ce n'est pas le cas au point b de l'intervalle I_k. En effet, la différence divisée définissant la B-spline s'annule

en b, provoquant une discontinuité sur l'intervalle $[x_k, x_{k+1}]$. Il est cependant possible de définir la B-spline N^n_{n+K+1} au point b, si $b = y_{n+K+2} = \ldots = y_{2n+K+2}$, par

$$N^n_{n+K+1}(b) = \lim_{x \to b^-} N^m_{n+K+1}(x).$$

De cette façon, le théorème 3.3 reste valable pour $b = y_{n+K+2} = \ldots = y_{2n+K+2}$.

Nous avons donc construit ici une base locale pour l'espace des splines $\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta)$ et défini par là-même les fonctions B-splines. Pour l'instant, ils semblerait que ces fonctions puissent être de bonnes candidates pour ϕ puisqu'elles sont à support fini ; les autres propriétés désirables restent néanmoins encore à vérifier.

Dans la section suivante, nous nous intéressons à la généralisation des résultats obtenus ici au cas multivarié.

3.1.2 Le cas des B-splines multivariées

Les sections précédentes ont présenté différentes définitions et propriétés des B-splines unidimensionnelles. Nous allons voir à présent comment étendre ces notions au cas multidimensionnel, puisque nous n'utiliserons dans ce travail que de telles fonctions. En effet, une image est une fonction au minimum bidimensionnelle.

Supposons que nous souhaitions travailler dans un espace de dimension d. Pour obtenir un espace des splines multidimensionnel, nous allons utiliser le produit tensoriel. Donnons pour commencer quelques définitions et résultats généraux à ce sujet [43].

Définition 3.6 (Produit tensoriel d'espaces linéaires de fonctions)**.** *Soient U_i, $i = 1, \ldots, d$, des espaces linéaires de fonctions $X_i \to \mathbb{R}$ où X_i, $i = 1, \ldots, d$, sont des ensembles quelconques. Pour tout $u_i \in U_i$, la règle*

$$w(x_1, x_2, \ldots, x_d) := u_1(x_1)u_2(x_2)\ldots u_d(x_d), \quad (x_1, x_2, \ldots, x_d) \in X_1 \times X_2 \times \ldots \times X_d,$$

définit une fonction sur $X_1 \times X_2 \times \ldots \times X_d$ que l'on appelle le produit tensoriel de $u_1, u_2, \ldots, u_d$, dénoté par

$$u_1 \otimes u_2 \otimes \ldots \otimes u_d \quad ou \quad \bigotimes_{i=1}^{d} u_i.$$

De plus, l'ensemble de toutes les combinaisons linéaires finies des fonctions définies sur $X_1 \times X_2 \times \ldots \times X_d$ de la forme $u_1 \otimes u_2 \otimes \ldots \otimes u_d$, pour $u_i \in U_i$ $(i = 1, \ldots, d)$, est un espace linéaire appelé produit tensoriel des espaces $U_1, \ldots, U_d$ et est donné par

$$\begin{aligned}
\bigotimes_{i=1}^{d} U_i \;=\; & \Big\{ \sum_{i_1=1}^{p_1} \ldots \sum_{i_d=1}^{p_d} \alpha_{i_1,\ldots,i_d} u_{1,j_1}(x_1)\ldots u_{d,j_d}(x_d) \;:\; \alpha_{i_1,\ldots,i_d} \in \mathbb{R}, \quad u_{1,i_1} \in U_1, \ldots, u_{d,i_d} \in U_d, \\
& \qquad i_1 \in \{1, \ldots, p_1\}, \ldots, i_d \in \{1, \ldots, p_d\}, \quad p_1, \ldots, p_d \in \mathbb{N} \;\; et \;\; x_1 \in X_1, \ldots, x_d \in X_d \Big\} \\
= \; & span\Big\{ u_{1,j_1} u_{2,j_2} \ldots u_{d,j_d} \Big\}_{j_1,j_2,\ldots,j_d=1}^{p_1,p_2,\ldots,p_d}.
\end{aligned}$$

Théorème 3.4 (Dimension de l'espace produit tensoriel)**.** *Soient $d \in \mathbb{N}$ et $U_1, U_2, \ldots, U_d$ des espaces linéaires de fonctions à valeurs dans $\mathbb{R}$. Alors*

$$dim\left(\bigotimes_{i=1}^{d} U_i\right) = dim(U_1)\, dim(U_2)\, \ldots\, dim(U_d).$$

Adaptons à présent ces résultats aux fonctions splines. Soient i, un entier tel que $1 \leq i \leq d$, $[a_i, b_i]$ un intervalle fini fermé et n_i, un entier positif. Supposons alors que

$$\Delta_i = \{a_i = x_{i,0} < x_{i,1} < \ldots < x_{i,k_i+1} = b_i\}$$

est une partition de $[a_i, b_i]$ et que

$$\mathcal{N}_i = (n_{i,1}, \ldots, n_{i,k_i}), \quad 1 \leq n_{i,j} \leq n_i + 1, \quad j = 1, 2, \ldots, k_i.$$

Par le théorème 3.2 nous savons alors que les espaces $\mathcal{S}(\mathcal{P}_{n_i}; \mathcal{N}_i; \Delta_i)$ sont de dimension $n_i + K_i + 1$, où $K_i = \sum_{j=1}^{k_i} n_{i,j}$. De plus, si $b_i < y_{2n_i+K_i+2}$, alors par le théorème 3.3,

$$\mathcal{S}(\mathcal{P}_{n_i}; \mathcal{N}_i; \Delta_i) = \text{span}\big\{N_{i,j}^{n_i}(x_i)\big\}_{j=1}^{n_i+K_i+1}.$$

Nous sommes donc ici en possession de d espaces linéaires de splines, sur base desquels nous allons pouvoir construire un espace multidimensionnel de dimension d via la définition 3.6. Considérons à cette fin, pour $i = 1, \ldots, d$, les partitions étendues respectivement associées aux partitions Δ_i :

$$\tilde{\Delta}_i = \{y_{i,j}\}_{j=1}^{2n_i+K_i+2}$$

avec

$$y_{i,1} \leq \ldots \leq y_{i,n_i+1} \leq a_i, \quad b_i \leq y_{i,n_i+K_i+2} \leq \ldots \leq y_{i,2n_i+K_i+2}$$

et

$$y_{i,n_i+2} \leq \ldots \leq y_{i,n_i+K_i+1} = \underbrace{x_{i,1}, \ldots, x_{i,1}}_{n_{i,1}}, \ldots, \underbrace{x_{i,k_i}, \ldots, x_{i,k_i}}_{n_{i,k_i}}.$$

Soient les ensembles de B-splines normalisées $\{N_{i,j}^{n_i}\}_{j=1}^{n_i+K_i+1}$ associés aux partitions étendues $\tilde{\Delta}_i$, $i = 1, \ldots, d$.

Définition 3.7 (B-spline produit tensoriel)**.** *Pour tout $i_1, \ldots, i_d$ tel que $1 \leq i_j \leq n_j + K_j + 1$, $j = 1, \ldots, d$,*

$$N_{i_1,\ldots,i_d}(x_1, \ldots, x_d) = N_{1,i_1}^{n_1}(x_1) \cdots N_{d,i_d}^{n_d}(x_d)$$

est appelé B-spline produit tensoriel.

Nous pouvons à présent utiliser la définition 3.6 pour obtenir un produit tensoriel d'espaces de splines.

Définition 3.8 (Espace des splines produit tensoriel)**.** *L'espace des splines produit tensoriel est défini par*

$$\mathcal{S} = \bigotimes_{i=1}^{d} \mathcal{S}(\mathcal{P}_{n_i}; \mathcal{N}_i; \Delta_i) = span\big\{N_{1,i_1}^{n_1}(x_1) \cdots N_{d,i_d}^{n_d}(x_d)\big\}_{i_1=1,\ldots,i_d=1}^{n_1+K_1+1,\ldots,n_d+K_d+1}.$$

Par les théorèmes 3.2 et 3.4, il apparaît que l'espace $\mathcal{S}$ est de dimension

$$\prod_{i=1}^{k}(n_i + K_i + 1).$$

Notons également qu'une spline s de l'espace $\mathcal{S}$ est définie sur l'ensemble

$$H = \bigotimes_{i=1}^{d}[a_i, b_i] = \{(x_1, \ldots, x_d) \ : \ a_i \leq x_i \leq b_i, \ i = 1, \ldots, d\}.$$

Nous avons donc finalement vu, dans cette section, qu'il est très simple de passer de B-splines unidimensionnelles à des B-splines multidimensionnelles grâce à l'usage du produit tensoriel. Notons que les B-splines respectent ainsi la propriété de *séparabilité*, propriété voulue pour la fonction ϕ que nous avions introduite dans la Section 2.4.

Les résultats présentés dans la suite de ce chapitre sont donnés pour le cas d'une dimension unique puisqu'il est facile de ramener les B-splines multidimensionnelles à des B-splines unidimensionnelles.

3.1.3 B-splines à noeuds uniformément espacés

Nous avons vu, au Chapitre 2, Section 2.4, que l'interpolation dans le cadre du recalage d'images s'effectuait via la transformation inverse afin de pouvoir interpoler à patir de points régulièrement espacés, comme l'indiquait la figure 2.4. Si l'on interpole une image à l'aide de B-splines, cela se traduit par l'usage de B-splines ayant des noeuds uniformément espacés, c'est-à-dire, en reprenant les notations de la Section 3.1.1 :

$$y_1 < y_2 < \ldots < y_{2m+K+2}$$

et

$$\exists \ h \in \mathbb{R} \text{ tel que } \forall \ i \in \{1, \ldots, 2n + K + 1\} : y_{i+1} - y_i = h.$$

Les B-splines normalisées à noeuds uniformément espacés sont dites B-splines uniformes. Nous les noterons β.

Dès lors que nous considérons des noeuds uniformément espacés, il est possible de généraliser la définition de B-spline donnée précédemment. En effet, les B-splines uniformes ne sont que des translations mises à l'échelle l'une de l'autre, comme illustré par la figure 3.1 pour l'ordre 3, créée à l'aide de Matlab [32], comme tous les graphes de ce travail. Mathématiquement parlant, cela signifie que l'égalité

$$\beta_i^n(x) = \beta^n\left(\frac{x - y_i}{h}\right)$$

est vérifiée. Cette propriété nous permet ainsi de définir une B-spline dite *de référence*, indépendante de l'indice i mais aussi de h en considérant sans perdre de généralité que h vaut 1.

Nous pouvons ainsi donner une formulation des B-splines uniformes très générale. Celle-ci, exprimée à la définition 3.9, peut être construite grâce à une expression particulière des différences divisées lorsque tous les noeuds sont distincts [43], donnée par le théorème suivant.

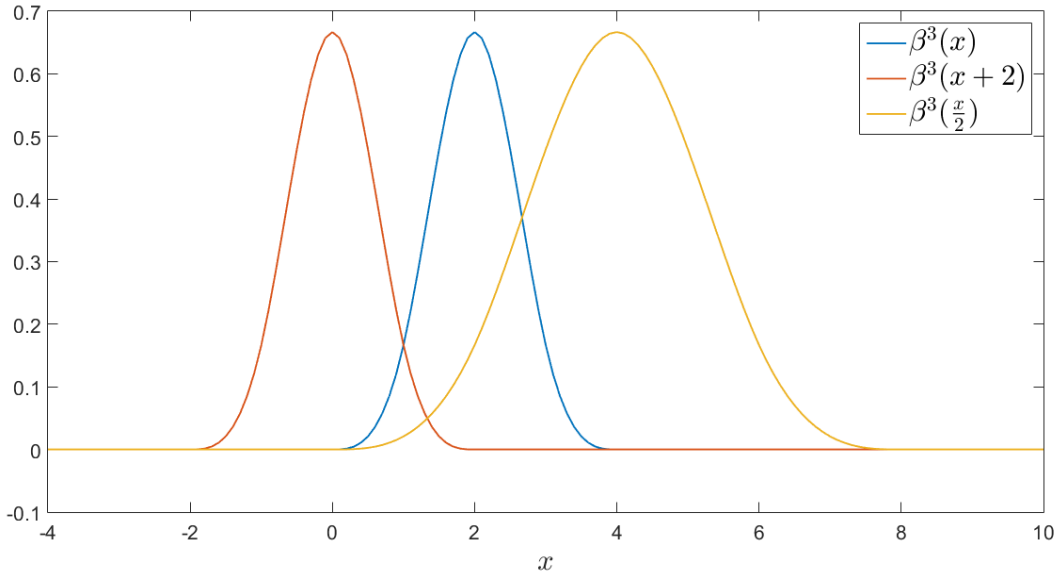


FIGURE 3.1 – Différentes B-splines cubiques à noeuds uniformément espacés

Théorème 3.5. *Si les noeuds $t_1, \ldots, t_{r+1}$ sont distincts, alors la différence divisée d'une fonction f en ces points peut être exprimée par*

$$[t_1, \ldots, t_{r+1}]f = \sum_{i=1}^{r+1} \frac{f(t_i)}{\prod\limits_{\substack{j=1 \\ j \neq i}}^{r+1} (t_i - t_j)}$$

Définition 3.9 (B-spline uniforme)**.** *Soit $n \in \mathbb{N}$. Une B-spline uniforme de degré n est le polynôme par morceaux*

$$\beta^n(x) = \sum_{i=0}^{n+1} \frac{(-1)^i}{n!} \binom{n+1}{i} (x-i)_+^n,$$

$\forall x \in \mathbb{R}$, *où*

$$\binom{n+1}{i} = \frac{(n+1)!}{(n+1-i)!\ i!}.$$

Nous remarquons cependant, via la courbe bleue (au centre) de la figure 3.1, que la B-spline de référence que nous venons de définir n'est pas symétrique par rapport à l'axe $x = 0$. Comme déjà vu, c'est une propriété souhaitable ; nous pouvons définir la B-spline de référence de manière symétrique grâce à l'introduction d'un simple décalage. Les B-splines seront ainsi symétriques partout sauf aux noeuds de raccord. Pour le cas cubique, on obtient la courbe rouge (à gauche) de la figure 3.1. C'est avec ces fonctions que nous allons travailler.

Définition 3.10 (B-spline uniforme symétrique)**.** *Soit $n \in \mathbb{N}$. Une B-spline uniforme symétrique de degré n est le polynôme par morceaux*

$$\beta^n(x) = \sum_{i=0}^{n+1} \frac{(-1)^i}{n!} \binom{n+1}{i} (x + \frac{n+1}{2} - i)_+^n,$$

$\forall x \in \mathbb{R}$, *où*

$$\binom{n+1}{i} = \frac{(n+1)!}{(n+1-i)!\ i!}.$$

Pour former une base de $\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta)$ à l'aide des B-splines ainsi définies, il suffit donc de prendre l'ensemble des B-splines shiftées $\beta^n(. - k)$, $k \in \mathbb{Z}$, non nulles sur une partie au moins de l'intervalle $[a, b]$. De plus, nous pouvons considérer que les coefficients c_k d'interpolation sont donnés par une fonction discrète c. Nous aurons donc par extension qu'une spline s d'ordre n peut s'exprimer de manière unique par

$$s(x) = \sum_{k \in \mathbb{Z}} c(k)\beta^n(x - k). \tag{3.2}$$

Maintenant que nous avons défini de manière définitive les fonctions B-splines que nous utiliserons dans par la suite, intéressons-nous à leurs propriétés. Un résultat très important, en particulier, est que les B-splines peuvent être obtenues récursivement par convolution.

Théorème 3.6. *Les B-splines d'ordre n peuvent être obtenues via la récursion*

$$\beta^n(x) = \beta^{n-1} * \beta^0(x) = \underbrace{\beta^0 * \beta^0 * \ldots * \beta^0(x)}_{(n+1)\ fois},$$

où $*$ désigne le produit de convolution.

Notons que, via la définition 3.9, il est facile de voir que la B-spline d'ordre 0 est définie par

$$\beta^0(x) = \begin{cases} 1 & \text{si } \frac{-1}{2} \leq x < \frac{1}{2} \\ 0 & \text{sinon.} \end{cases}$$

Nous obtenons donc bien une fonction d'ordre 0, de type palier et discontinue. De manière générale, une B-spline d'ordre n est une fonction polynomiale par morceaux, composée de $n + 1$ parties et définie sur $n + 2$ noeuds.

Les B-splines sont des fonctions très utilisées en analyse numérique, notamment de par certaines de leurs propriétés. Nous reprenons ici les plus importantes, dont certaines ont déjà été évoquées précédemment [43].

Propriété 3.1. *Soit β^n la B-spline de référence d'ordre n, définie sur les noeuds uniformément espacés $y_1, \ldots, y_{n+2}$. Cette fonction respecte les propriétés suivantes :*

1. *Support local : $\beta^n(x) = 0\ \forall x \notin [y_1,\ y_{n+2}]$;*
2. *Positivité : $\forall x \in \mathbb{R},\ \beta^n(x) \geq 0$;*
3. *Continuité et différentiabililité : $\beta^n \in \mathcal{C}^{n-1}$;*
4. *Partition de l'unité : $\sum_{i \in \mathbb{Z}} \beta^n(x - i) = 1$.*

Les points 1 et 4 de la propriété 3.1 confirment que les B-splines pourraient jouer le rôle des fonctions ϕ dans l'équation (3.1). De plus, nous avons déjà vu que les B-splines peuvent être symétriques et qu'elles sont séparables. Elles sont donc finalement de bonnes candidates pour ϕ. Les points 2 et 3 de la propriété 3.1 sont eux essentiels parce qu'ils montrent l'intérêt des B-splines

dans l'interpolation des *images*. En effet, il est pratique d'interpoler une image à l'aide de fonctions positives, car on cherche à interpoler des intensités de pixels, qui sont souvent des valeurs positives. De plus, le recalage d'images consiste en une optimisation, requérant ainsi, pour la plupart des algorithmes, des dérivations de l'image, c'est-à-dire des dérivations de son interpolant. En choisissant un ordre de spline suffisamment élevé, nous obtiendrons donc un interpolant continûment dérivable une ou plusieurs fois. Remarquons que cette propriété est due au fait que les noeuds $y_1, \ldots, y_{2n+K+2}$ sont distincts. En effet, par définition de partition étendue, cela implique que la multiplicité de chaque noeud x_i $(i = 1, \ldots, k)$ vaut 1. Les splines s de l'espace $\mathcal{S}(\mathcal{P}_n; \mathcal{N}; \Delta)$ doivent donc être $n-1$ fois continûment dérivables.

Un dernier résultat qui nous sera très utile dans le cadre de ce travail est l'expression de la dérivée d'une B-spline [47]. Ce théorème étant un des résultats phares de ce mémoire, nous avons tenu à tenter de le prouver malgré que nous n'ayons trouvé aucune source reprenant la démonstration.

Théorème 3.7 (Dérivée d'une B-spline)**.** *La dérivée d'une B-spline d'ordre n par rapport à x est donnée par*

$$\frac{\partial \beta^n(x)}{\partial x} = \beta^{n-1}\left(x + \frac{1}{2}\right) - \beta^{n-1}\left(x - \frac{1}{2}\right),$$

$\forall x \in \mathbb{R}$.

Démonstration. Démontrons la formule par récurrence sur n.

- **Pas initial** :
  Via la définition 3.9, nous trouvons qu'une B-spline de degré 1 est donnée par

$$\begin{aligned}\beta^1(x) &= (x+1)\ \mu(x+1) - 2x\ \mu(x) + (x-1)\ \mu(x-1) \\ &= \begin{cases} x+1 & \text{si } -1 \leq x < 0 \\ 1-x & \text{si } 0 \leq x < 1 \\ 0 & \text{sinon.} \end{cases}\end{aligned}$$

  Or nous avons

$$\begin{aligned}\beta^0(x+\frac{1}{2}) - \beta^0(x-\frac{1}{2}) &= \mu(x+1) + \mu(x) - \mu(x) - \mu(x-1) \\ &= \begin{cases} x+1 & \text{si } -1 \leq x < 0 \\ 1-x & \text{si } 0 \leq x < 1 \\ 0 & \text{sinon,} \end{cases}\end{aligned}$$

  et l'égalité est donc bien vérifiée pour $n = 1$.

- **Pas de récurrence** :
  Supposons maintenant que l'égalité soit vérifiée pour $m = 1, \ldots, n-1$ et démontrons-la pour $m = n$. Utilisons pour cela le théorème 3.6 ainsi que la règle de dérivation d'un produit de convolution.

$$\begin{aligned}\frac{\partial \beta^n(x)}{\partial x} &= \frac{\partial}{\partial x}(\beta^{n-1} * \beta^0)(x) \\ &= \Big(\frac{\partial \beta^{n-1}}{\partial x} * \beta^0\Big)(x),\end{aligned}$$

par règle de dérivation d'une convolution. Il en découle que

$$\begin{aligned}\frac{\partial \beta^n(x)}{\partial x} &= \int_{-\infty}^{+\infty} \frac{\partial \beta^{n-1}}{\partial x}(x-t)\,\beta^0(t)\,dt \\ &= \int_{-\infty}^{+\infty} \left(\beta^{n-2}\Big(x-t+\frac{1}{2}\Big) - \beta^{n-2}\Big(x-t-\frac{1}{2}\Big)\right)\beta^0(t)\,dt,\end{aligned}$$

par hypothèse de récurrence, et donc

$$\begin{aligned}\frac{\partial \beta^n(x)}{\partial x} &= \int_{-\infty}^{+\infty} \beta^{n-2}\Big(x-t+\frac{1}{2}\Big)\beta^0(t)\,dt - \int_{-\infty}^{+\infty} \beta^{n-2}\Big(x-t-\frac{1}{2}\Big)\beta^0(t)\,dt \\ &= (\beta^{n-2} * \beta^0)\Big(x+\frac{1}{2}\Big) - (\beta^{n-2} * \beta^0)\Big(x-\frac{1}{2}\Big) \\ &= \beta^{n-1}\Big(x+\frac{1}{2}\Big) - \beta^{n-1}\Big(x-\frac{1}{2}\Big),\end{aligned}$$

par le théorème 3.6.

□

Nous pouvons dès lors déduire du théorème 3.7 et de (3.2) que la dérivée d'une spline d'ordre n est donnée par

$$\begin{aligned}\frac{\partial s}{\partial x}(x) &= \sum_{k\in\mathbb{Z}} c_k \frac{\partial \beta^n}{\partial x}(x-k) \\ &= \sum_{k\in\mathbb{Z}} c_k \left(\beta^{n-1}\Big(x-k+\frac{1}{2}\Big) - \beta^{n-1}\Big(x-k-\frac{1}{2}\Big)\right) \\ &= \sum_{k\in\mathbb{Z}} c_k \beta^{n-1}\Big(x-k+\frac{1}{2}\Big) - \sum_{k\in\mathbb{Z}} c_k \beta^{n-1}\Big(x-k-\frac{1}{2}\Big) \\ &= \sum_{k\in\mathbb{Z}} c_k \beta^{n-1}\Big(x-k+\frac{1}{2}\Big) - \sum_{k\in\mathbb{Z}} c_{k-1} \beta^{n-1}\Big(x-k+\frac{1}{2}\Big) \\ &= \sum_{k\in\mathbb{Z}} (c_k - c_{k-1}) \beta^{n-1}\Big(x-k+\frac{1}{2}\Big). \qquad (3.3)\end{aligned}$$

Il nous faut cependant garder en tête que le cas multidimensionnel est un peu différent ; par la propriété de séparabilité des B-splines, nous pouvons écrire une spline d'ordre n et de dimension d sous la forme

$$s(x_1, ..., x_d) = \sum_{k_1\in\mathbb{Z}} \dots \sum_{k_d\in\mathbb{Z}} c_{k_1 \dots k_d} \beta^n(x_1 - k_1) \dots \beta^n(x_d - k_d),$$

et sa dérivée par rapport à x_i est donc donnée par

$$\frac{\partial s}{\partial x_i}(x_1, ..., x_d) = \sum_{k_1\in\mathbb{Z}} ... \sum_{k_d\in\mathbb{Z}} c_{k_1 \dots k_d} \beta^n(x_1 - k_1) \dots \beta^n(x_{i-1} - k_{i-1}) D_i \beta^n(x_{i+1} - k_{i+1}) \dots \beta^n(x_d - k_d), \qquad (3.4)$$

où

$$D_i = \beta^{n-1}\Big(x_i - k_i + \frac{1}{2}\Big) - \beta^{n-1}\Big(x_i - k_i - \frac{1}{2}\Big).$$

3.1.4 Les B-splines cubiques

Dans ce travail, nous nous concentrons sur les B-splines d'ordre 3. Elles consistent en effet en un bon compromis entre temps d'exécution, qualité d'interpolation et complexité de calcul. De plus, elles sont deux fois continûment dérivables. Les B-splines d'ordre 3 sont d'ailleurs très utilisées dans les domaines du traitement d'images et du traitement du signal.

Calculons l'expression d'une B-spline d'ordre 3 via la définition 3.10 :

$$\begin{aligned}
\beta^3(x) &= \frac{1}{6}\,(x+2)^3\,\mu(x+2) + \frac{-2}{3}\,(x+1)^3\,\mu(x+1) + x^3\,\mu(x) \\
&\quad + \frac{-2}{3}\,(x-1)^3\,\mu(x-1) + \frac{1}{6}\,(x-2)^3\,\mu(x-2) \\
&= \begin{cases}
0 & \text{si } x < -2 \\
\frac{1}{6}\,(x+2)^3 & \text{si } -2 \le x < -1 \\
\frac{1}{6}\,(x+2)^3 - \frac{2}{3}\,(x+1)^3 & \text{si } -1 \le x < 0 \\
\frac{1}{6}\,(x+2)^3 - \frac{2}{3}\,(x+1)^3 + x^3 & \text{si } 0 \le x < 1 \\
\frac{1}{6}\,(x+2)^3 - \frac{2}{3}\,(x+1)^3 + x^3 - \frac{2}{3}\,(x-1)^3 & \text{si } 1 \le x < 2 \\
\frac{1}{6}\,(x+2)^3 - \frac{2}{3}\,(x+1)^3 + x^3 - \frac{2}{3}\,(x-1)^3 + \frac{1}{6}\,(x-2)^3 & \text{si } 2 \le x
\end{cases} \\
&= \begin{cases}
\frac{1}{6}\,(x+2)^3 & \text{si } -2 \le x < -1 \\
\dfrac{-3x^3 - 6x^2 + 4}{6} & \text{si } -1 \le x < 0 \\
\dfrac{3x^3 - 6x^2 + 4}{6} & \text{si } 0 \le x \le 1 \\
\frac{1}{6}\,(2-x)^3 & \text{si } 1 \le x < 2 \\
0 & \text{sinon.}
\end{cases}
\end{aligned}$$

La B-spline obtenue est représentée à la figure 3.2.

Utilisons à présent le théorème 3.7 pour déterminer l'expression de la dérivée d'une B-spline cubique. Par la définition 3.9, nous trouvons que

$$\begin{aligned}
\beta^2(x) &= \frac{1}{2}\left(x+\frac{3}{2}\right)^2 \mu\left(x+\frac{3}{2}\right) - \frac{3}{2}\left(x+\frac{1}{2}\right)^2 \mu\left(x+\frac{1}{2}\right) \\
&\quad + \frac{3}{2}\left(x-\frac{1}{2}\right)^2 \mu\left(x-\frac{1}{2}\right) - \frac{1}{2}\left(x-\frac{3}{2}\right)^2 \mu\left(x-\frac{3}{2}\right),
\end{aligned}$$

et donc, par le théorème 3.7,

$$\begin{aligned}
\frac{\partial \beta^3(x)}{\partial x} &= \beta^2\left(x+\frac{1}{2}\right) - \beta^2\left(x-\frac{1}{2}\right) \\
&= \frac{1}{2}(x+2)^2\mu(x+2) - 2(x+1)^2\mu(x+1) + 3x^2\mu(x) \\
&\quad -2(x-1)^2\mu(x-1) + \frac{1}{2}(x-2)^2\mu(x-2).
\end{aligned}$$

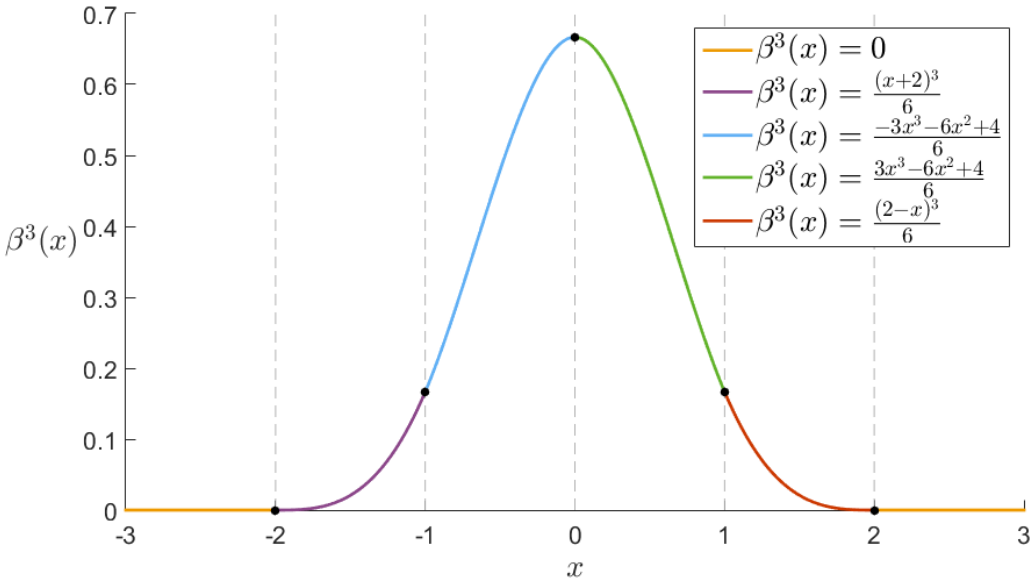


FIGURE 3.2 – B-spline cubique

En développant, nous trouvons finalement

$$\frac{\partial \beta^3(x)}{\partial x} = \begin{cases} \frac{1}{2}(x+2)^2 & \text{si } -2 \leq x < -1 \\ \frac{x(-3x-4)}{2} & \text{si } -1 \leq x < 0 \\ \frac{x(3x-4)}{2} & \text{si } 0 \leq x \leq 1 \\ \frac{-1}{2}(x-2)^2 & \text{si } 1 \leq x < 2 \\ 0 & \text{sinon.} \end{cases}$$

La figure 3.3 illustre la dérivée d'une B-spline cubique.

3.2 Interpolation par B-splines pour les images

Dans cette section, nous allons voir comment interpoler optimalement une image à partir des B-splines d'ordre 3. Comme nous l'avons déjà rappelé en début de chapitre, le but est de déterminer une fonction f de la forme (3.2). Grâce à l'introduction des B-splines à la section précédente, le problème peut maintenant s'écrire

$$f(x) = \sum_{k \in \mathbb{Z}^d} c(k) \beta^n(x-k) \quad \forall\, x \in \mathbb{R}^d,$$

où seule la fonction c est inconnue, i.e., les coefficients c_k définis par (3.1). Cependant, nous avons vu que les B-splines respectaient la propriété de séparabilité. Une interpolation d-dimensionnelle peut donc être divisée en d interpolations unidimensionnelles, ce qui allège à la fois les calculs et le coût. La procédure concrète peut être expliquée pour le cas 2D via l'équation suivante, provenant

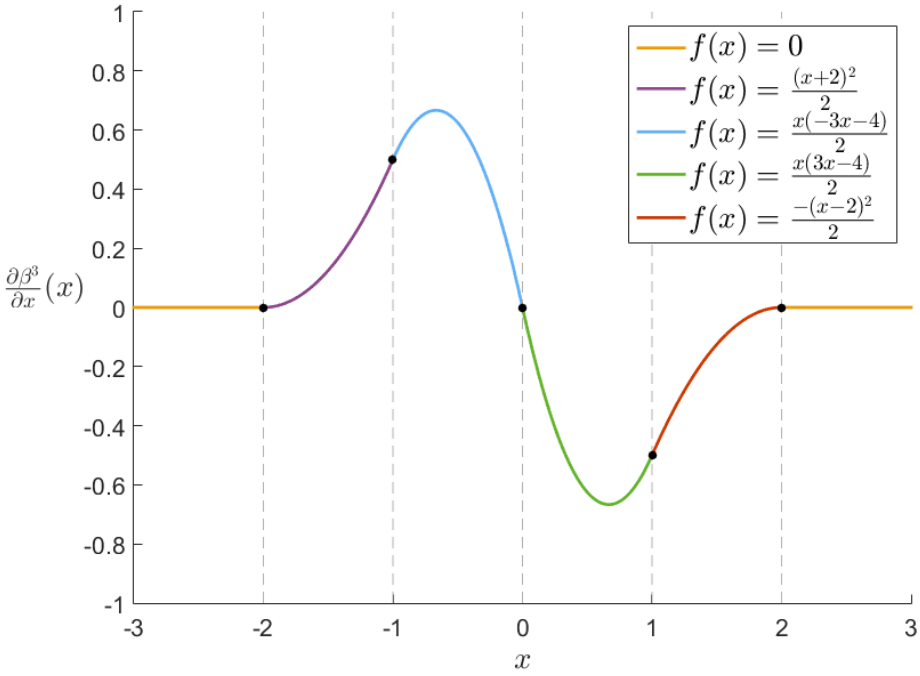


FIGURE 3.3 – Dérivée d'une B-spline cubique

de la propriété de séparabilité des B-splines :

$$\begin{aligned} s(x,y) &= \sum_{i\in\mathbb{Z}}\sum_{j\in\mathbb{Z}} c_{ij}\beta^n(x-i)\beta^n(y-j) \\ &= \sum_{i\in\mathbb{Z}} \beta^n(x-i)\sum_{j\in\mathbb{Z}} c_{ij}\beta^n(y-j) \\ &= \sum_{i\in\mathbb{Z}} c_i(y)\beta^n(x-i), \end{aligned}$$

où $c_i(y) = \sum_{j\in\mathbb{Z}} c_{ij}\beta^n(y-j)$. Il suffit alors de calculer les coefficients d'une interpolation 1D selon x et d'utiliser ensuite les valeurs obtenues comme signal connu pour interpoler en 1D selon la direction y :

$$c_i(y) = \sum_{j\in\mathbb{Z}} c_{ij}\beta^n(y-j).$$

Les coefficients trouvés sont ainsi les c_{ij} que l'on cherchait au départ [45]. Cet exemple 2D est généralisable à n'importe quelle dimension d.

Nous pouvons donc réduire le problème au cas unidimensionnel :

$$f(x) = \sum_{k\in\mathbb{Z}} c(k)\beta^n(x-k) \quad \forall\, x \in \mathbb{R}. \tag{3.5}$$

Le but de cette section est de décrire une méthode efficace pour le calcul des coefficients c_k, $k \in \mathbb{Z}$. L'approche que nous adoptons ici est basée sur le traitement du signal. La première partie de cette section introduit donc la théorie du signal, tandis que la deuxième reprend pas à pas la construction de la méthode d'interpolation par B-splines.

3.2.1 Introduction au traitement du signal

Pour construire la méthode d'interpolation que nous utiliserons dans ce travail, il nous faut utiliser les bases de la théorie du signal. Le but de cette sous-section est de les introduire, mais nous n'abordons pas toutes les notions incontournables de la théorie du signal, uniquement ce qui est indispensable pour la construction de notre méthode d'interpolation. L'essentiel de cette section est basé sur [24] et [44].

Dans un premier temps, nous voyons les notions basiques relatives aux signaux et aux systèmes. Nous introduisons ensuite ce qui a trait aux filtres et au domaine fréquentiel.

Signaux et systèmes

Avant toute chose, il nous faut préciser formellement ce qu'est un signal. Le mot *signal* est peu précis ; il peut être défini de beaucoup de façons différentes. La définition choisie ici est celle de M. Kunt [24] ; elle est extrêmement large.

M. Kunt définit simplement un *signal* comme étant le support physique d'une information. Les ondes de lumière, par exemple, sont des signaux permettant de nous apporter de l'information visuelle. Mathématiquement parlant, un signal est représenté par une fonction d'une (ou plusieurs) variable(s). Cette variable est très souvent le temps, mais pas toujours. L'exemple qui nous vient à l'esprit dans le cadre de ce travail est bien sûr l'image ; une image est un signal fonction de deux variables au moins dont aucune n'est le temps puisqu'il s'agit simplement de variables représentant le système de coordonnées de l'image.

A partir de là, il est possible d'imaginer différents types de signaux. Pour commencer, il nous faut remarquer que les variables dont dépend un signal peuvent être continues ou discrètes. Dans le premier cas, le signal est dit *analogique* ; dans le second, il est dit *échantillonné*. L'amplitude du signal peut également être continue ou discrète. Un signal analogique avec une amplitude discrète est ainsi dit *quantifié* tandis qu'un signal échantillonné ayant une amplitude également discrète est dit *numérique*. La figure 3.4 montre des exemples des quatre types de signaux envisagés.

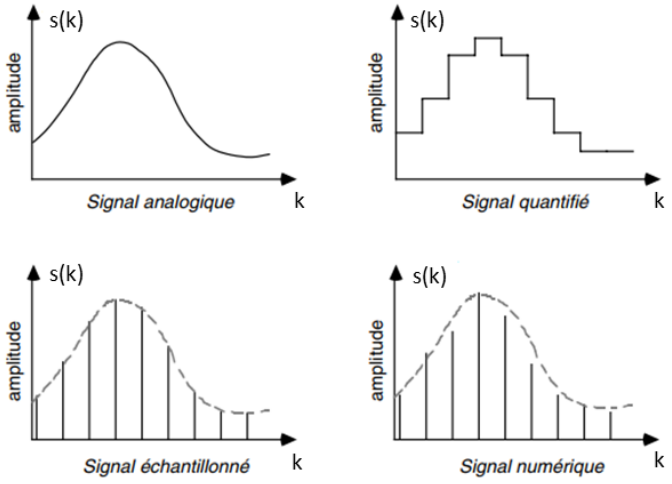


FIGURE 3.4 – Classification des signaux [9]

Notre exemple d'image est clairement celui d'un signal numérique. Lorsque nous évoquerons un signal par la suite, nous ferons donc concrètement référence à une suite de valeurs numériques. Les signaux numériques peuvent posséder différentes propriétés. Citons ici celles susceptibles de nous intéresser :

- Un signal peut être à valeurs réelles (signal *réel*) ou complexes (signal *complexe*).
- Un signal est dit *périodique* de période P si l'on a, pour toute valeur de k,

$$x(k) = x(k + P).$$

- Un signal est *à longueur limitée* s'il n'est défini qu'en un nombre fini N de points tel que, pour un certain entier k_0,

$$x(k) = \begin{cases} x(k) & \text{pour } k_0 \leq k \leq k_0 + K - 1 \\ 0 & \text{partout ailleurs.} \end{cases}$$

- Un signal est dit *déterministe* si son évolution peut être parfaitement prédite mathématiquement. Si ce n'est pas le cas, il est dit *aléatoire*.

Les signaux peuvent être générés de plusieurs manières. Il est tout d'abord possible de générer une suite de nombres et d'ensuite faire correspondre ceux-ci à un échantillon de la variable indépendante. Un signal numérique peut aussi être formé grâce à l'obtention de mesures d'une certaine grandeur physique effectuées à des intervalles réguliers de temps (ou fonctions d'une autre variable discrète bien sûr). Ensuite, les signaux déterministes peuvent bien sûr être représentés via l'utilisation d'une fonction mathématique, ou encore par une relation de récurrence qui y correspond. Par exemple, le signal représenté par

$$x(k) = 2x(k - 1)$$

avec $x(0) = 1$ est équivalent à

$$x(k) = \begin{cases} 2^k & \text{si } k \geq 0 \\ 0 & \text{sinon.} \end{cases}$$

Avant de continuer, donnons l'expression de deux signaux très courants :

– Le signal *impulsion* est donné par

$$d(k) = \begin{cases} 1 & \text{si } k = 0 \\ 0 & \text{sinon.} \end{cases}$$

– Le signal *saut unité* est défini par

$$w(k) = \begin{cases} 1 & \text{si } k \geq 0 \\ 0 & \text{sinon.} \end{cases}$$

Notons que le signal impulsion nous permet notamment d'écrire n'importe quel signal sous forme de la série

$$x(k) = \sum_{l=-\infty}^{+\infty} x(l)d(k - l), \tag{3.6}$$

pour tout entier k. En effet, par définition, l'impulsion $d(k-l)$ n'est non nulle que pour $l=k$.

Maintenant que nous avons défini plus précisément les signaux, nous pouvons aborder le *traitement du signal*. En effet, les signaux doivent généralement être traités pour que l'on puisse en extraire de l'information. Ces traitements sont effectués par ce que l'on appelle des *systèmes*. En pratique, un système opère sur un signal d'entrée et produit, en sortie, un signal sous une forme plus appropriée pour l'utilisation que l'on souhaite en faire. Ce signal de sortie est en fait appelé *réponse* du système. Une réponse bien connue est la *réponse impulsionnelle* ; celle-ci, notée g, est simplement la réponse du système pour un signal impulsion en entrée.

Les systèmes peuvent être classés parallèlement aux signaux. Ainsi, les systèmes numériques sont les systèmes agissant sur un signal numérique donné en entrée et produisant en sortie un signal numérique également. Mathématiquement parlant, un système est représenté par une transformation ou un opérateur fonctionnel S opérant sur un signal d'entrée $x(k)$ pour le transformer en signal de sortie $y(k)$:

$$y(k) = S[x(k)].$$

Plus concrètement, un opérateur est simplement une équation (ou système d'équations) représentant la relation entre $x(k)$ et $y(k)$.

Tout comme pour les signaux, il existe plusieurs façons de caractériser un système. Nous en donnons ici quatre :

- Un système est dit *causal* si l'opérateur S associé ne dépend que des valeurs passées et présentes du signal d'entrée. Par exemple,

  $$y(k) = x(k) + x(k-1)$$

  est causal mais

  $$y(k) = x(k) + x(k+1)$$

  ne l'est pas. Un système est *anticausal* s'il ne dépend que des valeurs futures du signal d'entrée.

- Un système est *linéaire* si l'opérateur S associé respecte la contrainte

  $$S[ax_1(k) + bx_2(k)] = aS[x_1(k)] + bS[x_2(k)],$$

  où a, b sont des constantes et x_1, x_2 sont des signaux d'entrée.

- Un système linéaire est dit *invariant* s'il ne varie pas sous translation, i.e., pour tout entier k,

  $$y(k) = x(k) \;\Rightarrow\; y(k-k_0) = x(k-k_0),$$

  pour k_0 un entier quelconque.

- Un système linéaire invariant est *stable* si sa réponse impulsionnelle $g(k)$ satisfait la condition suivante :

  $$\sum_{k=-\infty}^{+\infty} |g(k)| < \infty. \tag{3.7}$$

Ici, nous nous intéresserons particulièrement aux systèmes linéaires invariants car ce sont ceux que nous utiliserons pour l'interpolation par B-splines. Un tel système peut être écrit sous une forme

particulière grâce à l'expression (3.6). En effet, par linéarité puis par invariance d'un système S, nous avons

$$\begin{aligned} y(k) &= S\left[\sum_{l=-\infty}^{+\infty} x(l)d(k-l)\right] \\ &= \sum_{l=-\infty}^{+\infty} x(l)S[d(k-l)] \\ &= \sum_{l=-\infty}^{+\infty} x(l)g(k-l). \end{aligned} \tag{3.8}$$

Cette représentation d'un système linéaire invariant est très utile pour montrer différents résultats. Le théorème qui suit est d'ailleurs basé dessus, mais nous ne détaillons pas ici le processus de démonstration [24].

Théorème 3.8. *Tout système linéaire invariant peut être contraint à être causal.*

Ce théorème permet l'introduction d'une autre manière de représenter un système linéaire invariant. Celle-ci est donnée par l'équation aux différences, i.e.,

$$\sum_{n=0}^{N} a_n y(k-n) = \sum_{m=0}^{M} b_m x(k-m), \tag{3.9}$$

où les coefficients a_n, $n = 0, ..., N$, et b_m, $m = 0, ..., M$, sont constants et N, M sont des entiers positifs. A partir de là, nous pouvons définir l'*ordre* d'un système linéaire invariant par

$$\max(M, N).$$

Filtres numériques et domaine fréquentiel

Nous abordons ici les filtres numériques, qui sont des systèmes particuliers. Nous donnons aussi quelques définitions et résultats concernant les transformées en z et les fonctions de transfert, des fonctions définies dans le domaine fréquentiel et non le domaine temps [24].

Définition 3.11 (Filtre numérique)**.** *Un filtre numérique (ou digital) est un système numérique linéaire invariant utilisé pour modifier la distribution fréquentielle des composantes d'un signal selon des spécifications données, en utilisant des opérations arithmétiques de précision limitée.*

Si cette définition semble un peu complexe, elle peut être assez simplement résumée en disant qu'un filtre numérique est un système linéaire invariant opérant sur un signal discret dans le but d'en garder uniquement la partie qui nous intéresse, ou encore de le transformer en un signal ayant certaines propriétés voulues.

Les filtres numériques sont souvent classifiés en fonction de la durée de leur réponse impulsionnelle. Il y a dès lors deux types de filtres :

– Les filtres à *réponse impulsionnelle finie* (FIR) possèdent une réponse impulsionnelle de durée finie. Celle-ci est donc un signal à durée limitée, la réponse impulsionnelle finit par s'annuler.

– Les filtres à *réponse impulsionnelle infinie* (IIR) possèdent une réponse impulsionnelle de durée infinie. Il existe donc une infinité de valeurs k pour lesquelles $g(k) \neq 0$.

Théorème 3.9. *Soit un filtre FIR et g sa réponse impulsionnelle. Si pour tout entier k,*

$$|g(k)| < +\infty,$$

alors ce filtre est stable.

En pratique, les filtres FIR sont quasiment tous stables.

Nous introduisons maintenant des fonctions très intéressantes dans le cadre du filtrage : les transformées en z.

Définition 3.12 (Transformée en z)**.** *La transformée en z, X(z), d'un signal $x(k)$, est définie par*

$$X(z) = \sum_{k=-\infty}^{+\infty} x(k) z^{-k}, \tag{3.10}$$

où z est une variable complexe et $X(z)$ une fonction complexe de z. La notation

$$X(z) = Z[x(k)]$$

est utilisée.

Propriété 3.2. *La transformée en z respecte les propriétés suivantes :*

1. *Linéarité : $Z[ax_1(k) + bx_2(k)] = aX_1(z) + bX_2(z)$.*
2. *Décalage : $Z[x(k-j)] = z^{-j}X(z)$.*
3. *Convolution : $Z[x_1 * x_2(k)] = X_1(z)X_2(z)$.*

Un autre outil essentiel, basé sur la transformée en z, est la fonction de transfert. Nous énonçons également un résultat sur cette fonction dont nous aurons l'utilité lors de la construction de la méthode d'interpolation par B-splines [24].

Définition 3.13 (Fonction de transfert)**.** *Soit un système linéaire invariant transformant un signal d'entrée $x(k)$ en un signal de sortie $y(k)$, donné par l'équation (3.8), i.e.,*

$$\sum_{l=-\infty}^{+\infty} x(l) g(k-l). \tag{3.11}$$

En passant au domaine fréquentiel via les transformées en z et leur propriété de convolution, nous avons

$$Y(z) = X(z)G(z).$$

La fonction $G(z)$ est appelée fonction de transfert du système et est donc définie comme étant la transformée en z de la réponse impulsionnelle.

Théorème 3.10. *La fonction de transfert d'un filtre IIR décrit par (3.9) est de la forme*

$$G(z) = \frac{\sum_{m=0}^{M} b_m z^{-m}}{\sum_{n=0}^{N} a_n z^{-n}}. \tag{3.12}$$

Nous avons à présent en main toute la théorie nécessaire pour construire notre méthode d'interpolation par filtrage. Cependant, avant d'y arriver, appliquons une partie de la théorie vue ici à un exemple intéressant que nous pourrons réutiliser dans la section suivante.

Exemple 3.1. *Calculons la fonction de transfert du filtre causal de premier ordre donné par*

$$y(k) = x(k) + ay(k-1),$$

où a est une constante quelconque. Calculons sa réponse impulsionnelle $g(k)$. Celle-ci est simplement trouvée en considérant $y(k)$ pour $x(k) = d(k)$. Nous avons $g(k) = 0$ pour $k < 0$. Ensuite, la réponse impulsionnelle est donnée par

$$\begin{aligned} g(0) &= x(0) + ay(-1) = 1 + 0 = 1 \\ g(1) &= x(1) + ay(0) = 0 + a = a \\ g(2) &= x(2) + ay(1) = 0 + a^2 = a^2 \\ &\vdots \end{aligned}$$

et donc, $\forall\, k \in \mathbb{Z}$,

$$g(k) = a^k \mu(k).$$

La fonction de transfert du filtre est ensuite donnée par la transformée en z du résultat obtenu, à savoir

$$G(z) = Z[g(k)] = \sum_{k=-\infty}^{+\infty} a^k \mu(k) z^{-k} = \sum_{k=0}^{+\infty} a^k z^{-k} = \sum_{k=0}^{+\infty} (az^{-1})^k = \frac{1}{1 - az^{-1}}.$$

Calculons à présent la fonction de transfert d'un filtre anticausal de premier ordre donné par

$$y(k) = ax(k+1) + ay(k+1),$$

où a est une constante quelconque. Le filtre peut être réécrit sous la forme

$$y(k) = \sum_{j=1}^{+\infty} a^j x(k+j) = \sum_{j=-\infty}^{1} a^{-j} x(k-j) = \sum_{j=-\infty}^{+\infty} a^{-j} \mu(-j-1) x(k-j),$$

et sa réponse impulsionnelle est $y(k)$ pour $x(k) = d(k)$, i.e.,

$$g(k) = a^{-k} \mu(-k-1).$$

La fonction de transfert associée à ce filtre anticausal est donc donnée par

$$G(z) = Z[g(k)] = \sum_{k=-\infty}^{+\infty} a^{-k} \mu(-k-1) z^{-k} = \sum_{k=-\infty}^{-1} a^{-k} z^{-k} = \sum_{k=1}^{+\infty} (az)^k = \frac{1}{1 - az} - 1 = \frac{az}{1 - az}.$$

3.2.2 Filtrage digital par B-splines

Dans cette partie, nous effectuons tout le chemin jusqu'à l'obtention d'une formule nous permettant, en pratique, d'effectuer notre interpolation ([45], [46], [47], [48]).

Rappelons d'abord à nouveau que nous souhaitons interpoler un échantillon de données à l'aide de B-splines de manière à obtenir une spline

$$s(x) = \sum_{i\in\mathbb{Z}} c(i)\beta^n(x-i) \quad \forall\, x \in \mathbb{R},$$

cf. Section 3.2, équation (3.5). Notons que nous avons ici remplacé l'indice de sommation k par i afin d'éviter toute confusion avec les notations de la théorie du signal. Pour un signal discret et réel $\{f(k)\}$ défini pour $k = -\infty, \ldots, +\infty$ définissant les intensités connues de l'image, la spline interpolante devra notamment vérifier

$$f(k) = s(k) = \sum_{i\in\mathbb{Z}} c(i)\beta^n(x-i) \quad \forall\, k \in \mathbb{Z}. \tag{3.13}$$

Bien sûr en pratique, $f(k)$ est un signal à durée limitée ; nous reviendrons sur la question plus loin.

Comme le laisse présager l'équation (3.13), nous allons devoir utiliser des B-splines discrètes (à noeuds uniformément espacés). Nous définissons ci-dessous un type particulier de B-splines discrètes dont nous aurons besoin.

Définition 3.14 (B-spline discrète avec facteur d'élargissement)**.** *Une B-spline discrète d'ordre n avec un facteur d'élargissement m est donnée $\forall\, k \in \mathbb{Z}$ par*

$$\begin{aligned} b_m^n(k) &= \beta^n\left(\frac{k}{m}\right) \\ &= \sum_{i=0}^{n+1} \frac{(-1)^i}{n!} \binom{n+1}{i} \left(\frac{k}{m} + \frac{(n+1)}{2} - i\right)^n \mu\left(\frac{k}{m} + \frac{(n+1)}{2} - i\right) \\ &= \frac{1}{m^n} \sum_{i=0}^{n+1} \frac{(-1)^i}{n!} \binom{n+1}{i} \left(k + m\left(\frac{(n+1)}{2} - i\right)\right)^n \mu\left(k + m\left(\frac{(n+1)}{2} - i\right)\right). \end{aligned}$$

Pour construire la méthode d'interpolation dont nous avons besoin, nous allons passer au domaine fréquentiel afin de caractériser au mieux les B-splines. La formule (3.13) est en fait une convolution discrète. Elle peut donc être réécrite sous la forme

$$f(k) = b_1^n * c\,(k).$$

En passant au domaine fréquentiel avec les transformées en z, nous obtenons, par la propriété 3.2,

$$F(z) = B_1^n(z)C(z),$$

où F, B_1^n et C sont les transformées en z respectives de f, b_1^n et c. Nous obtenons alors

$$C(z) = B_1^n(z)^{-1}F(z). \tag{3.14}$$

Il semblerait donc que nous puissions déterminer les coefficients c par filtrage inverse grâce au filtre $B_1^n(z)^{-1}$, dit *filtre spline direct d'ordre n*. Celui-ci est donné par définition de la transformée en z :

$$B_1^n(z)^{-1} = \frac{1}{\sum\limits_{k\in\mathbb{Z}} b_1^n(k)z^{-k}}. \tag{3.15}$$

Nous observons par le théorème 3.10 que $B_1^n(z)^{-1}$ correspond à la fonction de transfert d'un filtre à réponse impulsionnelle infinie. Cela amène donc la question de sa stabilité. En effet, pour être utilisable en pratique, un filtre doit être stable.

Comme dans ce travail nous travaillons par la suite avec des splines cubiques, nous nous contentons ici d'étudier le cas $n = 3$. Calculons donc $B_1^3(z)^{-1}$:

$$\begin{aligned}
B_1^3(z)^{-1} &= \frac{1}{\sum\limits_{k\in\mathbb{Z}} b_1^3(k)z^{-k}} \\
&= \frac{1}{\sum\limits_{k\in\mathbb{Z}} \beta^3(k)z^{-k}} \\
&= \left(0 + \ldots + 0 + \frac{1}{6}z^{-1} + \frac{2}{3}z^0 + \frac{1}{6}z^1 + 0 + \ldots + 0\right)^{-1} \\
&= \frac{6}{z + 4 + z^{-1}} \\
&= \frac{6z}{z^2 + 4z + 1}.
\end{aligned}$$

Les pôles de cette fonction sont $-2 \pm \sqrt{3}$. Notons $\alpha = -2 + \sqrt{3}$. Le deuxième pôle est l'inverse du premier. Dès lors, nous pouvons écrire

$$\begin{aligned}
B_1^3(z)^{-1} &= \frac{6z}{(z-\alpha)(z-\alpha^{-1})} \\
&= \frac{6}{(1-\alpha z^{-1})(\frac{\alpha z - 1}{\alpha})} \\
&= \frac{-6\alpha}{(1-\alpha z^{-1})(1-\alpha z)}.
\end{aligned} \tag{3.16}$$

En redéveloppant l'expression obtenue, nous pouvons voir qu'il s'agit en fait de la fonction de transfert d'un filtre à réponse impulsionnelle infinie. En effet,

$$\begin{aligned}
B_1^3(z)^{-1} &= \frac{-6\alpha}{1 - \alpha z - \alpha z^{-1} + \alpha^2} \\
&= \frac{-6\alpha z^{-1}}{-\alpha z^0 + (\alpha^2 + 1)z^{-1} - \alpha z^{-2}}
\end{aligned}$$

Nous retrouvons donc bien une expression de la forme (3.12).

Via une décomposition en fractions simples, il est également possible de redévelopper $B_1^3(z)^{-1}$ pour obtenir une expression plus claire :

$$B_1^3(z)^{-1} = \frac{-6\alpha}{(1-\alpha^2)}\left(\frac{1}{1-\alpha z^{-1}} + \frac{\alpha z}{1-\alpha z}\right). \tag{3.17}$$

Grâce à l'exemple 3.1 donné dans la sous-section précédente, introduisant la théorie du signal, nous remarquons tout de suite que le terme

$$\frac{1}{1-\alpha z^{-1}}$$

correspond à la fonction de transfert d'un filtre causal de premier ordre, tandis que le terme

$$\frac{\alpha z}{1-\alpha z}$$

est clairement la fonction de transfert d'un filtre anticausal de premier ordre. Leurs réponses impulsionnelles, toujours via l'exemple , sont respectivement

$$\alpha^k \mu(k) \quad \text{et} \quad \alpha^{-k}\mu(-k-1).$$

Leur somme est donc $\alpha^{|k|}$ et la réponse impulsionnelle totale liée à $B_1^3(z)^{-1}$ est dès lors donnée par

$$g(k) = \frac{-6\alpha}{(1-\alpha^2)}\,\alpha^{|k|}.$$

Cette fonction est une exponentielle qui décroît avec $|k|$ puisque $|\alpha| < 1$. La figure 3.5 l'illustre d'ailleurs. Par définition, la stabilité du filtre spline direct est donc assurée.

Repartons à présent de l'équation (3.14) en redéveloppant grâce aux résultats que nous venons d'obtenir :

$$\begin{aligned} C(z) &= B_1^3(z)^{-1}\,F(z) \\ &= \frac{-6\alpha}{(1-\alpha^2)}\left(\frac{F(z)}{1-\alpha z^{-1}} + \frac{F(z)}{1-\alpha z} - F(z)\right). \end{aligned}$$

Pour retrouver la fonction discrète décrivant les coefficients c, il nous faut revenir dans le domaine temps. Nous allons pour cela calculer les transformées en z inverses de chacun des trois termes du facteur de droite puisque la transformée en z respecte la propriété de linéarité (cf. Propriété 3.2).

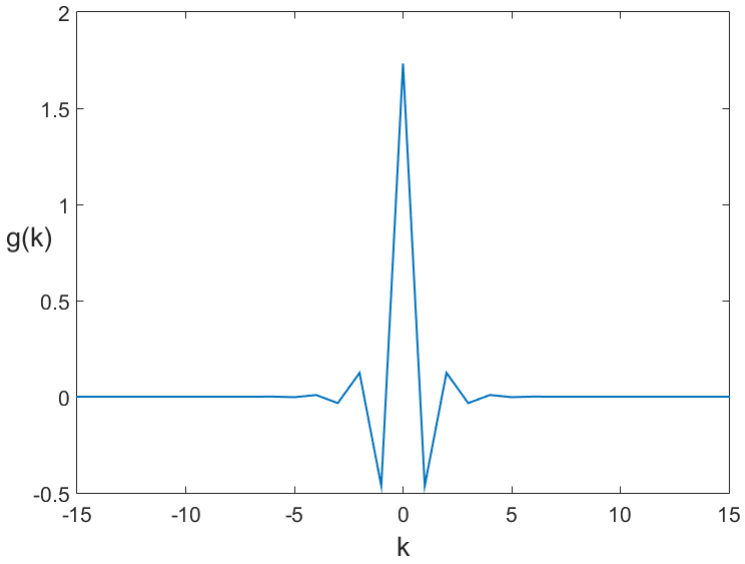


FIGURE 3.5 – Réponse impulsionnelle du filtre spline direct

- Transformée en z inverse de $F(z)$:

  Nous noterons celle-ci $f(k)$.

- Transformée en z inverse de $G_1(z) := \dfrac{F(z)}{1-\alpha z^{-1}}$:

  Observons tout d'abord que
  $$G_1(z) = F(z).\frac{1}{1-\alpha z^{-1}}$$
  et que donc la transformée en z inverse de $G_1(z)$, notée $g_1(k)$, sera le produit de convolution entre les transformées en z inverses respectives de $F(z)$ et $\frac{1}{1-\alpha z^{-1}}$. Notons la première $f(k)$ et calculons la seconde. Nous voyons d'abord que
  $$\frac{1}{1-\alpha z^{-1}} = \sum_{i=0}^{+\infty}(\alpha z^{-1})^i = \sum_{i=0}^{+\infty}\alpha^i z^{-i} = \sum_{i=-\infty}^{+\infty}\alpha^i \mu(i) z^{-i}.$$
  Dès lors, sa transformée en z inverse est simplement $\alpha^k\mu(k)$ et nous obtenons ainsi
  $$\begin{aligned} g_1(k) &= \alpha^k\mu(k) * f(k) \\ &= \sum_{i=-\infty}^{+\infty}\alpha^{k-i}\mu(k-i)f(i) \\ &= \sum_{i=-\infty}^{k}\alpha^{k-i}f(i). \end{aligned}$$

- Transformée en z inverse de $G_2(z) := \dfrac{F(z)}{1-\alpha z}$:

  Le raisonnement est similaire au précédent. Nous avons cette fois
  $$\frac{1}{1-\alpha z} = \sum_{i=0}^{+\infty}(\alpha z)^i = \sum_{i=0}^{+\infty}\alpha^i z^i = \sum_{i=-\infty}^{0}\alpha^{-i}z^{-i} = \sum_{i=-\infty}^{+\infty}\alpha^{-i}\mu(-i)z^{-i},$$
  ce qui correspond à la transformée en z de $\alpha^{-k}\mu(-k)$. Nous obtenons ainsi
  $$\begin{aligned} g_2(k) &= \alpha^{-k}\mu(-k) * f(k) \\ &= \sum_{i=-\infty}^{+\infty}\alpha^{-(k-i)}\mu(-(k-i))f(i) \\ &= \sum_{i=k}^{+\infty}\alpha^{i-k}f(i). \end{aligned}$$

Il s'ensuit que la fonction discrète décrivant les coefficients de la spline interpolante que nous cherchons à déterminer est donnée par
$$c(k) = \frac{-6\alpha}{(1-\alpha^2)}\left(\sum_{i=-\infty}^{k}\alpha^{k-i}f(i) + \sum_{i=k}^{+\infty}\alpha^{i-k}f(i) - f(i)\right). \tag{3.18}$$
Néanmoins, comme en pratique nous ne disposons pas d'un signal $f(k)$ de longueur infinie, nous allons uniquement considérer la suite $\{f(1), \ldots, f(K)\}$ pour un certain $K \in \mathbb{N}$. La formule (3.18) devient ainsi
$$c(k) = \frac{-6\alpha}{(1-\alpha^2)}\left(\sum_{i=1}^{k}\alpha^{k-i}f(i) + \sum_{i=k}^{K}\alpha^{i-k}f(i) - f(i)\right). \tag{3.19}$$

Cette formule peut ensuite être exprimée récursivement via la création de deux variables intermédiaires :

$$\begin{cases} c^+(k) = f(k) + \alpha c^+(k-1) & (k = 2, \dots, K) \\ c^-(k) = f(k) + \alpha c^-(k+1) & (k = K-1, \dots, 1) \\ c(k) = \dfrac{-6\alpha}{(1-\alpha^2)}(c^+(k) + c^-(k) - f(k)) \end{cases},$$

où la variable $c^+(k)$ est l'expression du filtre causal donné par la première somme de l'expression (3.19), tandis que la variable $c^-(k)$ correspond au filtre anticausal, à savoir la deuxième somme de (3.19).

Il faut à présent définir des conditions frontières pour c^+ et c^-, c'est-à-dire choisir les valeurs $c^+(1)$ et $c^-(K)$. Une pratique courante en traitement de l'image est d'étendre le signal en miroir, i.e.,

$$\begin{cases} f(1-k) = f(k+1) & (k = 0, \dots, K-1) \\ f(k) = f(2K-k) & (k = K, \dots, 2K-1). \end{cases} \tag{3.20}$$

Nous pouvons dès lors exprimer les conditions initiales sur base de ce nouveau signal. Pour $c^+(1)$, nous avons par définition

$$\begin{aligned} c^+(1) &= f(1) + \alpha c^+(0) \\ &= f(1) + \alpha f(0) + \alpha^2 c^+(-1) \\ &\vdots \\ &= \sum_{k=1}^{+\infty} \alpha^{k-1} f(2-k) \\ &= \sum_{k=1}^{+\infty} \alpha^{k-1} f(k), \end{aligned}$$

par (3.20), en remarquant que la condition $f(1-k) = f(k+1)$ $(k = 0, \dots, K-1)$ est équivalente à la condition $f(2-k) = f(k)$ $(k = 1, \dots, K)$. La somme obtenue peut être calculée jusqu'à un certain $k = k_0$, choisi de manière à ce que α^{k_0} respecte un certain niveau de précision. Similairement, nous avons

$$\begin{aligned} c^-(K) &= f(K) + \alpha c^-(K+1) \\ &= f(K) + \alpha f(K+1) + \alpha^2 c^-(K+2) \\ &\vdots \\ &= \sum_{k=0}^{+\infty} \alpha^k f(K+k) \\ &= \sum_{k=0}^{+\infty} \alpha^k f(K-k), \end{aligned}$$

par (3.20), en remarquant que la condition $f(k) = f(2K-k)$ $(k = K, \dots, 2K-1)$ est équivalente à la condition $f(K+k) = f(K-k)$ $(k = 0, \dots, K-1)$. Il est facile de voir que la condition initiale ici obtenue correspond en fait à $c^+(K)$.

Nous pouvons maintenant conclure en donnant ici la formule finale permettant de calculer les coefficients c. Pour un signal étendu donné par (3.20), nous avons

$$\begin{cases} c^+(k) = f(k) + \alpha c^+(k-1) & (k = 2, \dots, K) \\ c^-(k) = f(k) + \alpha c^-(k+1) & (k = K-1, \dots, 1) \\ c(k) = \dfrac{-6\alpha}{(1-\alpha^2)}(c^+(k) + c^-(k) - f(k)) & \\ c^+(1) = \sum\limits_{k=1}^{k_0} \alpha^{k-1} f(k) & \\ c^-(K) = c^+(K). & \end{cases} \tag{3.21}$$

Une autre expression est cependant également souvent utilisée. Donnée par le théorème qui suit [45], elle est basée sur la décomposition du filtre spline direct en produit et non en somme, c'est-à-dire sur l'expression (3.16) et non (3.17). Cela signifie donc qu'au lieu de sommer un filtre causal et un filtre anticausal, nous appliquons un filtre anticausal après l'application d'un filtre causal.

Théorème 3.11. *Pour un signal $\{f(0), ..., f(K-1)\}$, $K \in \mathbb{N}$, étendu en miroir, la fonction c décrivant les coefficients de la spline interpolante peut être exprimée sous la forme*

$$\begin{cases} c^+(k) = f(k) + \alpha c^+(k-1) & (k = 1, \dots, K-1) \\ c^-(k) = \alpha(c^-(k+1) - c^+(k)) & (k = K-2, \dots, 0) \\ c(k) = 6c^-(k) & \end{cases} \tag{3.22}$$

avec

$$c^-(K-1) = \frac{\alpha}{\alpha^2 - 1}(c^+(K-1) + \alpha c^+(K-2))$$

et

$$c^+(0) = \begin{cases} \sum\limits_{k=0}^{k_0} \alpha^k f(k) & \text{si } k_0 \leq K-1 \\ \dfrac{1}{1-\alpha^{2K-2}} \sum\limits_{k=0}^{2K-3} \alpha^k f(k) & \text{sinon,} \end{cases}$$

où

$$k_0 > \frac{\log \epsilon}{\log \alpha},$$

avec ϵ la précision désirée.

3.3 Conclusion

Ce chapitre a porté sur la construction d'une méthode d'interpolation utilisant les fonctions B-splines.

Dans la première section, nous avons découvert les B-splines ainsi que les propriétés justifiant leur usage. Citons notamment celle de séparabilité, qui permet de réaliser plusieurs interpolations unidimensionnelles au lieu d'une interpolation multidimensionnelle, réduisant ainsi énormément le coût de calcul. Nous nous sommes particulièrement intéressés aux B-splines cubiques à noeuds uniformément espacés, qui constituent un bon outil pour l'interpolation d'images.

La seconde partie de ce chapitre a traité de la construction d'une méthode d'interpolation par filtrage numérique utilisant ces B-splines, basée sur le fait que l'interpolant discret

$$f(k) = s(k) = \sum_{i \in \mathbb{Z}} c(i)\beta^m(x - i) \quad \forall\, k \in \mathbb{Z},$$

était en fait une convolution discrète. Tout le problème résidant dans la découverte des coefficients c_i, $i \in \mathbb{Z}$, une formule explicite pour le calcul de ceux-ci a été trouvée.

Le prochain chapitre s'intéresse à l'algorithme de recalage utilisé dans ce mémoire : l'algorithme des ensembles de niveau.

Chapitre 4

Ensembles de niveau pour le recalage d'images

Ce mémoire a pour but l'étude de l'interpolation par B-splines pour le recalage d'images non rigide qui, pour rappel, était décrit au Chapitre 2, Section 2.3. La méthode de recalage choisie, à laquelle nous avons appliqué l'interpolation par B-splines, est appelée méthode des ensembles de niveau. Ce chapitre a pour but de la décrire.

Reprenons les notations du Chapitre 2. L'algorithme des ensembles de niveau est une méthode *non paramétrique*, c'est-à-dire que la transformation $\hat{T}$ recherchée pour résoudre le problème de recalage

$$\hat{T} = \arg \max_{T \in \mathcal{T}} S(I, T(J))$$

n'est pas paramétrisée comme le serait par exemple clairement un algorithme recherchant une translation optimale. La méthode est de plus *iconique* (cf. Chapitre 2, Section 2.1) et recherche une transformation non linéaire, i.e., une déformation. De plus, de par sa construction, elle est adaptée aux cas où les images à recaler diffèrent par des déformations *locales*.

Plus concrètement, l'algorithme des ensembles de niveau utilise la théorie de l'évolution des courbes et sa formulation en termes d'ensembles de niveau afin de trouver une transformation optimale pour l'image mouvante. Le champ de déformation est en fait la solution d'une équation aux dérivées partielles (EDP) décrivant le mouvement de l'image. Le critère de similarité est en quelque sorte implicite et ancré dans l'algorithme : c'est lui qui dirige intrinsèquement l'évolution du mouvement et ne peut donc être modifié. Il n'y a donc pas usage d'une méthode d'optimisation, tout le processus étant basé sur une EDP de mouvement créée pour tendre à minimiser le critère de similarité [35].

La première section de ce chapitre est consacrée à l'étude de la méthode des courbes et ensembles de niveau dans un cas général. La section suivante illustre ensuite comment appliquer ces notions pour obtenir un processus de recalage d'images. Nous effectuons ensuite quelques corrections sur les équations obtenues afin d'obtenir un algorithme plus robuste et plus stable. Enfin, une résolution numérique concrète des équations décrivant le recalage d'images est envisagée. L'ensemble de ce chapitre est principalement basé sur [28], [39] et [49].

4.1 Courbes et ensembles de niveau

Considérons une famille de courbes planaires fermées données par

$$C(p,t) : S^1 \times [0,r) \longrightarrow \mathbb{R}^2, \tag{4.1}$$

où S^1 est le cercle unité. Ces courbes évoluent dans le temps selon l'équation d'évolution

$$\frac{\partial C}{\partial t} = \vec{F}(p,t), \tag{4.2}$$

où $C(p,0) = C_0(p)$ est une courbe initiale quelconque et

$$\vec{F}(p,t) : S^1 \times [0,r) \longrightarrow \mathbb{R}^2$$

est un certain champ de vitesse. L'équation est donc physiquement assez logique ; elle exprime simplement que la vitesse d'un point sur une courbe est donnée par la dérivée de son déplacement.

Ce champ de vitesse $\vec{F}$ peut être décomposé en une composante normale à la courbe et une composante tangente à la courbe. L'équation (4.2) peut donc se réécrire

$$\frac{\partial C}{\partial t} = \alpha\vec{T} + \gamma\vec{N}, \tag{4.3}$$

où α est la composante tangentielle de $\vec{F}$, γ sa composante normale, $\vec{T}$ est le vecteur tangent unité et $\vec{N}$ est le vecteur normal unité à la courbe C. Notons que ce dernier peut être dirigé soit vers l'intérieur, soit vers l'extérieur de la courbe. Cela dépend du sens dans lequel on considère que la courbe évolue ; c'est une question de convention.

Théorème 4.1 (Epstein-Gage)**.** *Si le paramètre γ dans l'équation (4.3) ne dépend pas de la paramétrisation de la courbe, alors l'évolution d'une courbe décrite par (4.2) est la même que celle décrite par*

$$\frac{\partial C}{\partial t} = \gamma\vec{N}. \tag{4.4}$$

Le paramètre γ représente ainsi à lui seul la vitesse de la courbe [49]. Nous pouvons donc considérer uniquement les équations d'évolution de la forme (4.4).

Formulons à présent le problème en termes d'ensembles de niveau. Commençons par définir ceux-ci.

Définition 4.1 (Ensemble de niveau)**.** *Soit un naturel d et une fonction*

$$f : \mathbb{R}^d \longrightarrow \mathbb{R}.$$

L'ensemble de niveau c de f est donné par

$$L_c(f) = \{(x_1, ..., x_d) | f(x_1, ..., x_d) = c\},$$

où c est une constante quelconque.

Il est ainsi possible de définir une courbe planaire fermée comme étant un ensemble de niveau d'une fonction U. La figure 4.1 l'illustre.

Supposons donc que $C(p,t)$ forme l'ensemble de niveau zéro d'une fonction

$$U : \mathbb{R}^2 \times [0,r) \longrightarrow \mathbb{R}.$$

$C(p,t)$ satisfait donc

$$U(C,t) = 0. \tag{4.5}$$

En dérivant cette équation par rapport à t via la règle de dérivation en chaine, nous obtenons

$$\frac{\partial U}{\partial C}\frac{\partial C}{\partial t} + \frac{\partial U}{\partial t} = 0,$$

c'est-à-dire

$$\nabla U C_t + U_t = 0.$$

C_t n'est rien d'autre que le champ de vitesse (4.2) et nous avons vu que celui-ci peut s'exprimer uniquement via sa composante normale $\vec{N}$. Nous avons donc

$$\nabla U \gamma \vec{N} + U_t = 0. \tag{4.6}$$

Afin de trouver l'expression de la normale à C en termes d'ensembles de niveau, dérivons à nouveau l'équation (4.5), mais cette fois-ci selon la variable spatiale p. Nous obtenons

$$\frac{\partial U}{\partial C}\frac{\partial C}{\partial p} = 0,$$

ce qui donne, en notant $\cdot$ le produit scalaire,

$$\nabla U \cdot \vec{T} = 0,$$

puisque la tangente à la courbe correspond à sa dérivée spatiale. Cette équation entraîne alors que la tangente à C est perpendiculaire à ∇U et, pour obtenir un vecteur unitaire pointant ici vers l'intérieur de la courbe, nous prenons

$$\vec{N} = \frac{-\nabla U}{\|\nabla U\|}. \tag{4.7}$$

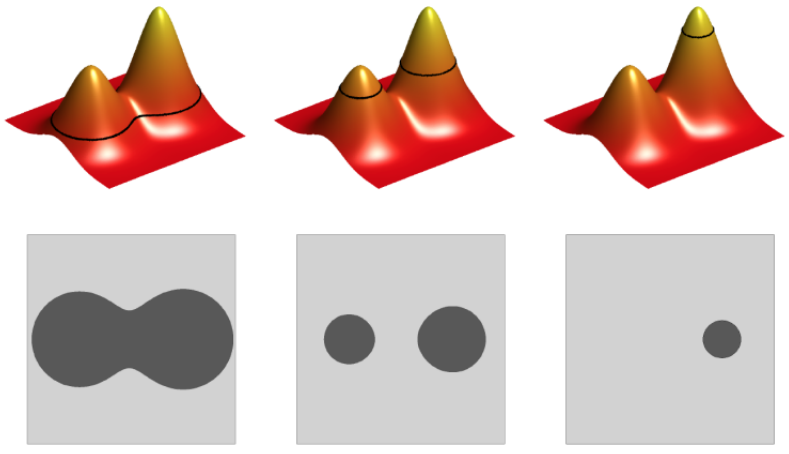


FIGURE 4.1 – Illustration de la représentation d'une courbe par un ensemble de niveau d'une fonction donnée [15]

En remplaçant dans l'équation (4.6), nous obtenons

$$\gamma \nabla U \frac{(-\nabla U)}{\|\nabla U\|} + U_t = 0,$$

et l'expressions finale est donc

$$U_t = \gamma \|\nabla U\|, \tag{4.8}$$

dite équation des ensembles de niveau et où γ représente la vitesse d'évolution de la courbe définie via l'ensemble de niveau zéro de la fonction U. Cette formulation permet donc de poser le problème d'évolution d'une courbe en termes d'ensembles de niveau.

4.2 Application au recalage d'images

Tentons à présent de considérer notre problème de recalage comme un problème d'évolution de courbes (cas 2D) ou de surfaces (cas 3D). Nous pouvons alors imaginer que recaler deux images correspond à faire évoluer les ensembles de niveau de la fonction image mouvante en les ensembles de niveau de la fonction image fixe. Il faut dès lors chercher un mapping envoyant les ensembles de niveau de l'image mouvante sur ceux de l'image fixe.

Soit $I_1(X)$, l'image mouvante que l'on souhaite recaler sur une image fixe, $I_2(X)$, avec X bidimensionnel ou tridimensionnel. La formulation du recalage d'images que nous avions introduite au Chapitre 2, équation (2.1), devient alors

$$\hat{T} = \arg\max_{T \in \mathcal{T}} S(I_2, T(I_1)).$$

Nous voulons que $I_1(X)$ évolue jusqu'à devenir $I_2(X)$. L'équation des ensembles de niveau de ce problème peut donc être écrite sous la forme

$$I_t(X,t) = \gamma \|\nabla I(X,t)\|, \text{ avec } I(X,0) = I_1(X), \tag{4.9}$$

où γ est le terme de vitesse. Il nous faut maintenant choisir un γ approprié. Comme l'évolution de l'image $I_1(X)$ doit s'arrêter lorsque celle-ci sera exactement $I_2(X)$, le terme de vitesse doit s'annuler lorsque les deux images sont exactement les mêmes. Un bon choix semble donc être donné par

$$\gamma = I_2(X) - I(X,t),$$

et l'équation (4.9) devient donc

$$I_t(X,t) = (I_2(X) - I(X,t)) \|\nabla I(X,t)\|, \text{ avec } I(X,0) = I_1(X). \tag{4.10}$$

Cette formulation de l'équation des ensembles de niveau ne permet cependant pas de trouver explicitement la transformation géométrique à appliquer à l'image mouvante.

Soit $\vec{V} = (u,v)^T$ le vecteur de déplacement en X ; $\vec{V}(X) = (x+u, y+v)^T$. Pour le cas 3D, nous avons similairement $\vec{V} = (u,v,w)^T$ et $\vec{V}(X) = (x+u, y+v, z+w)^T$. Le déplacement correspond en fait simplement à un ensemble de niveau de la fonction image mouvante I_1. Dès lors, son évolution peut être formulée similairement à l'évolution (4.4) de la courbe C dans la section précédente, c'est-à-dire

$$\vec{V}_t = \gamma \vec{N}.$$

Le déplacement doit s'annuler lorsque l'image mouvante devient l'image fixe. Prenons donc

$$\gamma = \big(I_2(X) - I_1(\vec{V}(X))\big).$$

En remplaçant de plus $\vec{N}$ par l'expression (4.7) trouvée à la section précédente, nous obtenons

$$\vec{V}_t = \big(I_2(X) - I_1(\vec{V}(X))\big)\frac{-\nabla I_1(\vec{V}(X))}{\|\nabla I_1(\vec{V}(X))\|} \text{ avec } \vec{V}(X,0) = \vec{0}.$$

Pour se débarrasser du signe négatif, nous pouvons redéfinir $\vec{V}(X)$ par $(x-u, y-v)^T$ en 2D ou $(x-u, y-v, z-w)^T$ en 3D. L'équation précédente devient ainsi

$$\vec{V}_t = \big(I_2(X) - I_1(\vec{V}(X))\big)\frac{\nabla I_1(\vec{V}(X))}{\|\nabla I_1(\vec{V}(X))\|} \text{ avec } \vec{V}(X,0) = \vec{0}. \tag{4.11}$$

4.3 Correction des équations d'évolution

Avant de passer à la résolution des équations d'évolution obtenues à la section précédente, il nous faut envisager quelques petites modifications dans les équations (4.10) et (4.11) afin de rendre l'algorithme plus robuste et plus stable [49].

En effet, l'équation (4.10) a deux possibilités pour s'annuler : soit le terme de vitesse $(I_2(X) - I(X,t))$ tombe à zéro, soit le gradient de I s'annule. Dans le cas présent, nous voulons que le processus stoppe uniquement si l'image $I_1(X)$ devient l'image fixe $I_2(X)$. Nous allons donc modifier l'équation (4.10) de manière à s'assurer que ce soit bien le cas :

$$I_t(X,t) = (I_2(X) - I(X,t))\sqrt{\|\nabla I(X,t)\|^2 + \epsilon}, \text{ avec } I(X,0) = I_1(X), \tag{4.12}$$

où ϵ est un petit nombre positif. L'auteur choisit lui $\epsilon = 0.1$.

Pour l'équation aux dérivées partielles (4.11), un problème de stabilité risque de se poser si le gradient de l'image mouvante approche zéro. On peut néanmoins pallier à ce problème en ajoutant un terme α au dénominateur :

$$\vec{V}_t = \big(I_2(X) - I_1(\vec{V}(X))\big)\frac{\nabla I_1(\vec{V}(X))}{\|\nabla I_1(\vec{V}(X))\| + \alpha}, \text{ avec } \vec{V}(X,0) = \vec{0},$$

où α est une petite constante positive.

Ensuite, pour l'équation (4.10) aussi bien que pour l'équation (4.11), il est conseillé d'effectuer un lissage gaussien sur l'image mouvante avant d'en calculer le gradient. En effet, le calcul du gradient est souvent très sensible au bruit dans l'image. En pratique, il faut convoluer l'image mouvante avec le filtre gaussien, donné par

$$G_\sigma(X) = \frac{1}{(\sqrt{2\pi}\sigma)^n}\, e^{-\frac{\|X\|^2}{2\sigma^2}},$$

où σ est l'écart-type du filtre gaussien et $X \in \mathbb{R}^n$. Plus σ augmente, plus l'image résultante est floutée. Nos équations aux dérivées partielles deviennent alors

$$I_t(X,t) = (I_2(X) - I(X,t))\sqrt{\|\nabla(G_\sigma * I(X,t))\|^2 + \epsilon}, \text{ avec } I(X,0) = I_1(X) \tag{4.13}$$

et

$$\vec{V}_t = \big(I_2(X) - I_1(\vec{V}(X))\big)\frac{\nabla(G_\sigma * I_1(\vec{V}(X)))}{\|\nabla(G_\sigma * I_1(\vec{V}(X)))\| + \alpha}, \text{ avec } \vec{V}(X,0) = \vec{0}. \tag{4.14}$$

4.4 Résolution numérique

Les calculs effectués ici sont pour le cas 2D mais que l'extension au cas 3D est assez simple. Nous notons les coordonnées d'un pixel (x, y) [49].

Avant de commencer, adaptons nos notations au cas discret. Notons I_{ij}^k la fonction image au pixel (i, j) à l'itération k. L'itération k est ainsi l'équivalent discret du temps continu t. Pour ce qui est de $\vec{V}$, notons

$$\begin{pmatrix} u_{ij}^k \\ v_{ij}^k \end{pmatrix}$$

le déplacement du pixel (i, j) à l'itération k. Définissons également

$$\begin{aligned} E_{ij} &:= G_\sigma * I_{ij}, \\ C_{ij} &:= G_\sigma * I_{i-u,j-v}. \end{aligned}$$

Au vu des équations (4.13) et (4.14), nous avons besoin d'une méthode efficace pour calculer le gradient de l'image mouvante. Commençons par définir les traditionnelles différences finies forward et backward d'une fonction discrète quelconque F au point (i, j) :

$$\begin{aligned} D_x^+ F_{ij} &= \frac{F_{i+1,j} - F_{ij}}{\Delta x}, \\ D_y^+ F_{ij} &= \frac{F_{i,j+1} - F_{ij}}{\Delta y}, \\ D_x^- F_{ij} &= \frac{F_{ij} - F_{i-1,j}}{\Delta x}, \\ D_y^- F_{ij} &= \frac{F_{ij} - F_{i,j-1}}{\Delta y}, \end{aligned}$$

où les deux premières définissent les différences forward selon x et y et les deux dernières les différences backward selon x et y.

Les auteurs de l'article [49] présentant la méthode affirment, après expérimentations de différents schémas de différences finies, que les différences finies *minmod* donnent un gradient d'une exactitude acceptable pour l'équation (4.14), tandis que les différences finies *upwind* sont adaptées pour le calcul du gradient dans (4.13).

La fonction *minmod* est donnée par

$$m(x, y) = \begin{cases} \operatorname{sign}(x) \min(|x|, |y|) & \text{si } xy > 0 \\ 0 & \text{sinon,} \end{cases}$$

où la fonction $\operatorname{sign}(x)$ donne le signe de x. Les dérivées de C au pixel (i, j) selon x et y sont alors données par

$$\begin{cases} (C_x)_{ij} = m(D_x^+ C_{ij}, D_x^- C_{ij}), \\ (C_y)_{ij} = m(D_y^+ C_{ij}, D_y^- C_{ij}). \end{cases}$$

Les différences finies *upwind* de la fonction E_{ij} sont elles données par

$$\begin{cases} (E_x)_{ij} = \left(\max(D_x^- E_{ij}, 0)^2 + \min(D_x^+ E_{ij}, 0)^2 \right)^{1/2}, \\ (E_y)_{ij} = \left(\max(D_y^- E_{ij}, 0)^2 + \min(D_y^+ E_{ij}, 0)^2 \right)^{1/2}. \end{cases}$$

Nous avons à présent tous les outils en main pour résoudre numériquement les équations aux dérivées partielles (4.13) et (4.14). Pour cela, nous allons utiliser un schéma de résolution par différences finies. Le principe est d'approximer la dérivée partielle selon t par une différence finie forward. Les équations (4.13) et (4.14) deviennent donc, en remplaçant également les gradients,

$$\frac{I_{ij}^{k+1} - I_{ij}^{k}}{\Delta t} = \big((I_2)_{ij} - I_{ij}^{k}\big)\sqrt{R},$$

où

$$R = \max(D_x^- E_{ij}, 0)^2 + \min(D_x^+ E_{ij}, 0)^2 + \max(D_y^- E_{ij}, 0)^2 + \min(D_y^+ E_{ij}, 0)^2 + \epsilon,$$

et

$$\frac{\begin{pmatrix} u_{ij}^{k+1} \\ v_{ij}^{k+1} \end{pmatrix} - \begin{pmatrix} u_{ij}^{k} \\ v_{ij}^{k} \end{pmatrix}}{\Delta t} = \frac{(I_2)_{ij} - (I_1)_{i-u_{ij}^k, j-v_{ij}^k}}{\sqrt{m^2(D_x^+ C_{ij}, D_x^- C_{ij}) + m^2(D_y^+ C_{ij}, D_y^- C_{ij})} + \alpha} \begin{pmatrix} m(D_x^+ C_{ij}, D_x^- C_{ij}) \\ m(D_y^+ C_{ij}, D_y^- C_{ij}) \end{pmatrix}.$$

Les équations finales de mise à jour sont donc

$$I_{ij}^{k+1} = I_{ij}^{k} + \Delta t\big((I_2)_{ij} - I_{ij}^{k}\big)\sqrt{R}$$

et

$$\begin{pmatrix} u_{ij}^{k+1} \\ v_{ij}^{k+1} \end{pmatrix} = \begin{pmatrix} u_{ij}^{k} \\ v_{ij}^{k} \end{pmatrix} + \Delta t \frac{(I_2)_{ij} - (I_1)_{i-u_{ij}^k, j-v_{ij}^k}}{\sqrt{m^2(D_x^+ C_{ij}, D_x^- C_{ij}) + m^2(D_y^+ C_{ij}, D_y^- C_{ij})} + \alpha} \begin{pmatrix} m(D_x^+ C_{ij}, D_x^- C_{ij}) \\ m(D_y^+ C_{ij}, D_y^- C_{ij}) \end{pmatrix},$$

avec les conditions initiales respectives

$$I_{ij}^{0} = (I_1)_{ij} \quad \text{et} \quad \begin{pmatrix} u_{ij}^{0} \\ v_{ij}^{0} \end{pmatrix} = \vec{0}.$$

Nous obtenons ainsi une mise à jour explicite du vecteur de déformation à appliquer en un pixel ainsi qu'une expression d'évolution de l'image mouvante. En pratique, il faudra choisir de travailler soit par image *transformée*, soit par image *évoluée*. On travaillera alors avec l'une ou l'autre des deux équations établies ci-dessus.

4.5 Conclusion

Dans ce chapitre, nous avons abordé la méthode des ensembles de niveau permettant l'étude de l'évolution des courbes et surfaces. Nous avons ensuite vu pourquoi et comment cette théorie s'appliquait particulièrement bien au recalage d'images. Cela nous a permis de déterminer deux équations aux dérivées partielles distinctes menant chacune à l'évolution de l'image mouvante vers l'image fixe. L'une fonctionne avec la géométrie de l'image tandis que l'autre agit directement sur son intensité. Après de petites modifications menant à des équations plus stables et plus robustes, nous avons vu comment résoudre celles-ci numériquement de manière à obtenir une méthode implémentable.

Il est important de remarquer que la méthode étudiée est construite de façon à ne pas devoir faire usage d'un algorithme d'optimisation externe. En effet, la phase d'optimisation est en quelque sorte comprise dans l'EDP définissant la méthode, qui optimise implicitement un critère de similarité donné, à savoir le terme de vitesse donné par la différences entre images.

Le prochain chapitre aborde l'implémentation de cette méthode de recalage d'images dans ITK, une librairie du C++. Nous y découvrons notamment l'implémentation de la phase d'interpolation et celle du gradient, sujets qui nous intéressent dans le cadre de ce travail.

Chapitre 5

Outils et implémentation

Maintenant que nous avons abordé toutes les thématiques théoriques associées au sujet de ce mémoire, il nous faut parler de la mise en pratique du processus de recalage décrit.

Dans la première section de ce chapitre, nous abordons le langage et la librairie utilisée pour l'implémentation du code. Nous voyons ensuite la construction et l'architecture générale du code. Un intérêt particulier est alors porté à l'implémentation de la méthode des ensembles de niveau et de la phase d'interpolation. Enfin, nous abordons le calcul du gradient de l'image.

5.1 La librairie ITK

Pour tester la méthode décrite plus tôt dans ce manuscrit, nous sommes partis d'un code existant provenant d'ITK (*Insight Segmentation and Registration Toolkit* [21]). ITK est une librairie open-source du C++ dont le développement a commencé en 1999 grâce au financement de l'Institut National de Santé américain. Les principaux participants au projet sont à la fois des entreprises et des universités. Citons entre autre GE Corporate R&D, Kitware Inc., MathSoft, l'Université de Caroline du Nord, l'Université de Pennsylvanie et l'Université du Tennessee. Comme beaucoup de librairies, ITK est en évolution constante. A l'heure d'aujourd'hui, énormément de scientifiques à travers le monde participent à son développement.

ITK offre toutes sortes d'algorithmes de traitement d'images dont principalement des méthodes de recalage et de segmentation. Elle consiste en des dossiers de codes, la plupart étant des classes d'objets. Dans la suite de cette section, nous abordons brièvement l'architecture générale d'ITK ainsi que son utilisation en pratique.

5.1.1 Architecture générale d'ITK

Comme déjà mentionné, la librairie ITK est faite d'un ensemble de dossiers contenant des codes. Pour la plupart, il s'agit de classes d'objets, qui ne peuvent donc pas être utilisées toutes seules. Ici, seuls deux de ces dossiers nous ont intéressés. Nous allons donc les passer brièvement en revue et voir, en pratique, comment combiner les différents codes de la librairie.

Explicitons tout d'abord un peu la notion de classe. Chaque classe d'ITK est constituée d'un fichier *.h* et d'un fichier *.hxx*. Le fichier *.h* contient généralement des définitions de types, de variables et de fonctions. Le fichier *.hxx*, lui, contient les fonctions en elles-mêmes.

Dans la version 4.10.0 que nous avons téléchargée, ITK contient notamment les dossiers *Modules* et *Examples* :

– Le dossier *Modules* est le dossier principal de la librairie. Divisé en sous-dossiers, il contient à lui seul tous les éléments clés des méthodes de traitement d'images proposées par la librairie. Une classe ne correspond cependant pas à une méthode de traitement d'images. Concentrons-nous ici sur le recalage. Pour implémenter un tel algorithme avec ITK, il faut lier différentes classes entre elles. En effet, chaque classe correspond à une composante du processus de recalage. Il y a par exemple une classe par méthode d'optimisation, une classe par méthode d'interpolation, une classe par type de transformation. Pour construire un code, il faut donc choisir chaque composante du processus de recalage et les assembler. Notons que, dans notre cas, il n'y aura pas de classe pour l'optimisation ou pour le type de transformation recherchée puisque tout cela est intrinsèque à la méthode des ensembles de niveau.

– Le dossier *Examples*, lui, reprend des fichiers *.cxx*, c'est-à-dire des programmes du langage C++. Il donne ainsi des exemples d'algorithmes qu'il suffit de lancer pour obtenir un résultat. Pour le recalage d'images, il propose notamment toute une série de codes permettant d'effectuer un recalage déformable. Ceux-ci se trouvent dans le sous-dossier *RegistrationITKv4*. Cela nous intéresse particulièrement puisque l'un de ces codes nous permettra d'exécuter l'algorithme des ensembles de niveau précédemment étudié. Celui-ci est le fichier *DeformableRegistration5.cxx*. Nous reviendrons plus en détails sur ce code dans la section suivante.

5.1.2 Utilisation d'ITK

Pour pouvoir utiliser ITK, il faut d'abord configurer la librairie. Pour cela, il faut télécharger *CMake* ainsi qu'un compilateur C++. *CMake* est un outil open-source destiné à construire, compiler et tester des logiciels [20]. La procédure exacte pour configurer ITK est décrite dans le premier livre du *Software Guide* d'ITK [23].

Une fois la librairie configurée, nous pouvons commencer à l'utiliser. Faire tourner un code d'ITK n'est cependant pas immédiat ; la compilation requiert à nouveau *CMake*. Décrivons rapidement la procédure utilisée pour compiler un code d'ITK sur Windows. Tout d'abord, il faut créer un fichier nommé *CMakeLists.txt* comprenant les paramètres et informations nécessaires à la construction de l'exécutable. Ce fichier doit être placé dans un dossier avec le programme principal que l'on souhaite compiler. CMake permet ensuite de construire des fichiers qui, ensemble, constituent un projet. Celui-ci peut être ouvert via un IDE tel que Visual Studio C++ et permet dans cet IDE la génération de l'exécutable. La procédure est décrite plus en détails dans [23].

5.2 Implémentation générale de l'algorithme étudié

Nous allons ici nous tenter d'expliquer la structure générale de l'algorithme testé. Cependant, les codes d'ITK possèdent une structure complexe qu'il n'est pas toujours évident de comprendre complètement. Cela est dû au fait que, lorsqu'on regarde un code d'ITK, la plupart des lignes ne sont pas explicites car elles font appel à des méthodes implémentées dans une autre classe. Un même algorithme peut ainsi faire appel à des centaines de classes différentes, ce qui rend la lecture et la compréhension du code difficiles. La plupart du temps, il sera donc sage de regarder uniquement le nom de la fonction pour tenter de comprendre ce qu'elle fait.

Pour commencer, jetons un oeil au premier algorithme donné en annexe. Il s'agit du code *DeformableRegistration5.cxx* qui constitue le programme principal faisant appel à la méthode de recalage qui nous intéresse. Décrivons brièvement les différentes étapes de ce code.

- **Etape 1 : Inclusion des fichiers nécessaires**
  Il faut, pour commencer, inclure les classes utilisées par le code. On inclut pour cela le fichier header *.h* de chaque classe nécessaire au programme principal.

- **Etape 2 : Vérification du nombre d'arguments donnés en entrée**
  La première étape du programme `main` est la vérification des arguments entrés par l'utilisateur. En effet, les algorithmes de recalage d'ITK requièrent, pour la plupart, de donner en entrée l'image fixe, l'image mouvante et le nom que l'on souhaite donner à l'image de sortie. Dans notre cas nous pouvons aussi, si nous le souhaitons, rajouter un argument qui sera un nom de fichier pour enregistrer la déformation finale à appliquer à l'image.

- **Etape 3 : Initialisations liées aux images et opérations préliminaires**
  ITK fonctionne énormément avec le mot-clé `typedef`. Celui-ci permet de définir des nouveaux types pour alléger la lecture du code et le rendre plus compréhensible. Ici, on définit notamment des nouveaux types `FixedImageType` et `MovingImageType` pour l'image fixe et l'image mouvante. La dimension des images à recaler est également fixée. On initialise ensuite des objets permettant de lire le contenu des images fixes et mouvantes. Enfin, un nouveau type d'images est défini : `InternalImageType`. Ce type d'images est constitué de pixels de type `float`, plus pratique pour retravailler l'image. Des filtres pour transformer (*caster*) une image à pixels entiers en une image à pixels réels sont ensuite définis, initialisés et lancés.

- **Etape 4 : Pré-recalage**
  Avant d'appliquer le recalage par ensembles de niveau, l'exemple *DeformableRegistration5.cxx* effectue un pré-recalage en tentant de mettre en correspondance les histogrammes des images, qui pour rappel avaient été introduits au Chapitre 1, Section 1.2.7.

- **Etape 5 : Initialisations liées au champ de déformation recherché**
  ITK considère les champs de déformation comme des images dont chaque pixel est un vecteur. On définit donc ici un type `VectorPixelType` pour les pixels et un type `DeformationFieldType` pour le champ de déformation qui est un type d'images composé de pixels sous formes de vecteurs.

- **Etape 6 : Configuration et lancement du filtre de recalage**
  On commence ici par créer un type de filtres pour la méthode des ensembles de niveau. On y lie ensuite la classe déclarée au début du code : `CommandIterationUpdate`. Cela permet, en bref, de surveiller le processus de recalage en affichant des mesures à l'écran. Dans notre cas, nous affichons la valeur de la fonction de similarité. Le filtre de recalage est ensuite configuré (nombre maximum d'itérations, images à recaler, etc). Le processus de recalage est enclenché via la méthode `Update()`. Nous ne voyons donc dans le programme principal aucun calcul lié au recalage. Concrètement, le procédé permettant d'arriver aux fonctions clés du recalage est très complexe et fait appel à énormément de classes différentes ; nous n'entrerons ici pas plus dans les détails de sa programmation.

- **Etape 7 : Application de la déformation finale à l'image mouvante**
  On définit tout d'abord un filtre de déformation et une méthode d'interpolation. Le filtre est ensuite configuré (interpolateur, image à déformer, paramètres de l'image), puis lancé.

- **Etape 8 : Ecriture de la nouvelle image**
  Avant d'écrire l'image mouvante de type `MovingImageType`, il faut la transformer en un nouveau type `OutputImageType`. Des types pour les pixels de l'image de sortie et pour l'image de

sortie sont donc définis. Un filtre de *cast* est ensuite initialisé et lancé. On initialise également un filtre permettant l'écriture de l'image dans un fichier dont le nom a été donné par l'utilisateur (étape 2). Il ne reste alors plus qu'à écrire l'image.

- **Etape 9 (optionnelle) : Ecriture du champ de déformation final**
  Si un quatrième argument a été donné au programme, c'est que l'on souhaite sauvegarder la déformation finale dans un fichier du nom de cet argument. Pour cela, on crée donc un filtre permettant d'écrire un champ de déformation. On le configure (champ à écrire et nom du fichier de sortie), et on l'enclanche.

5.3 Implémentation de la méthode des ensembles de niveau

Nous avons vu au chapitre précédent que le processus de recalage d'images pouvait notamment être exprimé via une équation aux dérivées partielles dont la solution numérique était donnée par

$$\begin{pmatrix} u_{ij}^{k+1} \\ v_{ij}^{k+1} \end{pmatrix} = \begin{pmatrix} u_{ij}^{k} \\ v_{ij}^{k} \end{pmatrix} + \Delta t \frac{(I_2)_{ij} - (I_1)_{i-u_{ij}^k, j-v_{ij}^k}}{\sqrt{m^2(D_x^+ C_{ij}, D_x^- C_{ij}) + m^2(D_y^+ C_{ij}, D_y^- C_{ij}) + \alpha}} \begin{pmatrix} m(D_x^+ C_{ij}, D_x^- C_{ij}) \\ m(D_y^+ C_{ij}, D_y^- C_{ij}) \end{pmatrix},$$

avec la condition initiale suivante :

$$\begin{pmatrix} u_{ij}^{0} \\ v_{ij}^{0} \end{pmatrix} = \vec{0}.$$

Une autre EDP, fonctionnant directement par évolution des intensités de l'image mouvante, avait également été établie ; sa solution était

$$I_{ij}^{k+1} = I_{ij}^{k} + \Delta t\big((I_2)_{ij} - I_{ij}^{k}\big)\sqrt{R},$$

avec $I_{ij}^0 = (I_1)_{ij}$ et

$$R = \max(D_x^- E_{ij}, 0)^2 + \min(D_x^+ E_{ij}, 0)^2 + \max(D_y^- E_{ij}, 0)^2 + \min(D_y^+ E_{ij}, 0)^2 + \epsilon.$$

Nous sommes donc ici en possession de deux possibilités pour faire évoluer les ensembles de niveau de l'image mouvante. La première agit sur la géométrie de l'image (image *transformée*), la deuxième sur son intensité (image *évoluée*). Dans la librairie ITK, c'est la première solution qui a été implémentée.

Avant de passer cette implémentation en revue, il nous faut aborder un dernier point important : le choix de la valeur de Δt. En effet, ce choix peut avoir un impact important sur la stabilité du schéma numérique de résolution de l'EDP. Ici, nous allons nous contenter d'étudier le cas où nous travaillons avec l'image *transformée* puisque c'est l'implémentation utilisée par ITK. Une condition nécessaire bien connue sur Δt est la restiction de Courant-Friedrichs-Levy (CFL), que nous énonçons ci-dessous [8].

Proposition 5.1 (Restriction CFL)**.** *Une condition nécessaire pour obtenir la stabilité d'un schéma numérique de résolution d'une EDP est donnée, pour le cas bidimensionnel, par*

$$\Delta t\left(\frac{|s_x|}{\Delta x} + \frac{|s_y|}{\Delta y}\right) \leq C_{max},$$

où s_x et s_y représentent respectivement le terme de vitesse selon x et y, Δx et Δy sont les pas spatiaux, Δt est le pas temporel et C_{max} est dit nombre de Courant.

En pratique, le nombre de Courant est souvent pris égal à 1. Nous obtenons alors

$$\Delta t \leq \frac{1}{\frac{|s_x|}{\Delta x} + \frac{|s_y|}{\Delta y}}.$$

Appliquons cette condition à notre expression discrète de la mise à jour de $\vec{V} = (u, v)^T$ obtenue ci-dessus. Comme il s'agit d'un déplacement, le terme de vitesse est clairement

$$\begin{pmatrix} s_x \\ s_y \end{pmatrix} = \frac{(I_2)_{ij} - (I_1)_{i-u_{ij}^k, j-v_{ij}^k}}{\sqrt{m^2(D_x^+ C_{ij}, D_x^- C_{ij}) + m^2(D_y^+ C_{ij}, D_y^- C_{ij})} + \alpha} \begin{pmatrix} m(D_x^+ C_{ij}, D_x^- C_{ij}) \\ m(D_y^+ C_{ij}, D_y^- C_{ij}) \end{pmatrix}.$$

Concrètement, nous prendrons en fait

$$\Delta t = \frac{1}{\max\left\{\frac{|s_x|}{\Delta x} + \frac{|s_y|}{\Delta y}\right\}}$$

afin d'assurer un maximum la stabilité. Il faudra donc calculer

$$\frac{|s_x|}{\Delta x} + \frac{|s_y|}{\Delta y}$$

en chaque pixel de l'image et garder, en fin d'itération, la valeur maximale. Notons que, ici, Δx et Δy correspondent au pas choisi pour les différences finies *minmod*.

Voyons à présent comment se passe la résolution numérique de l'EDP d'évolution dans ITK. Ce n'est pas compliqué : la librairie implémente simplement l'équation donnant la solution de cette EDP. En réalité, ITK possède toute une série d'algorithmes de recalage d'images procédant par résolution numérique d'une EDP via différences finies. Ce cadre de résolution est appelé *Finite Difference Solver* (FDS). On trouve ainsi dans la librairie une série de classes liées à la résolution par différences finies, communes à tous ces algorithmes. Celles-ci sont situées dans le dossier `\Modules\Core\FiniteDifference\include` d'ITK.

Les deux classes principales du framework FDS sont *itkFiniteDifferenceFilter* et *itkFiniteDifferenceFunction*. Les autres sont des sous-classes de celles-ci mais chacune a son utilité bien précise. La classe *itkFiniteDifferenceFilter*, ainsi que ses sous-classes, sont appelées des objets *solvers*. Ceux-ci prennent une image en entrée et produisent une image en sortie. La classe *itkFiniteDifferenceFunction* et ses sous-classes sont elles destinées aux calculs plus fins en un pixel de l'image [21].

De manière générale, beaucoup d'algorithmes d'ITK sont divisés en deux classes : ces deux classes possèdent le même nom, si ce n'est que l'une finit par *Filter* et l'autre par *Function*, comme nous venons de le voir pour le framework FDS. C'est notamment le cas de l'algorithme des ensembles de niveau, qui est implémenté dans les classes *itkLevelSetMotionRegistrationFilter* et *itkLevelSetMotionRegistrationFunction*, situées dans le dossier `\Modules\Registration\PDEDeformable\include`. C'est la seconde qui nous intéressera principalement puisque c'est celle qui effectue les calculs précis, pixel après pixel. Le fichier *.hxx* est repris en annexe, commenté, afin de retrouver le processus théorique décrit plus haut. La méthode `ComputeUpdate()` est particulièrement importante ; c'est dans cette fonction que l'on gère à la fois la mise à jour du déplacement, l'interpolation, le calcul du gradient et le calcul de la métrique de similarité. Celle-ci, pour l'algorithme des ensembles de niveau d'ITK, est la somme des différences au carré moyenne, c'est-à-dire la somme des différences au carré divisée par le nombre total de pixels :

$$S(I_2, T(I_1)) = \frac{1}{P} \sum_{s \in \Omega} \big(I_2(s) - T(I_1(s))\big)^2,$$

où Ω est l'intersection des domaines de l'image fixe, I_2, et de l'image mouvante transformée, $T(I_1)$, et P est le nombre de pixels des images. On considère donc ainsi que les intensités des images sont liées par une relation identité, comme vu au Chapitre 2, Section 2.2. Cela semble raisonnable puisque l'algorithme fonctionne par évolution des ensembles de niveau.

5.4 Phase d'interpolation

Nous avons vu, au Chapitre 3, comment l'interpolation par B-splines est effectuée en pratique. Nous allons maintenant décrire les deux classes d'ITK implémentant la méthode.

La première est la classe *itkBSplineDecompositionImageFilter*. Son but est de calculer les coefficients c_k de l'interpolation. Elle prend une image en entrée et renvoie une image dont les pixels correspondent aux coefficients d'interpolation. La deuxième est *itkBSplineInterpolateImageFunction*. Elle calcule l'interpolant et l'évalue en un point donné.

Tout ce qui touche à l'interpolation par B-splines est en réalité implémentée sur trois niveaux différents. Une première partie concerne la déclaration et l'initialisation de l'interpolateur. Cette étape n'est réalisée qu'une seule fois sur tout le processus de recalage d'images. Une deuxième partie gère l'image à interpoler ainsi que le calcul des coefficients c_k de l'interpolant. Cette étape a lieu au début de chaque itération de l'algorithme. Enfin, une dernière partie s'occupe de l'évaluation de l'interpolant en un point précis de l'image. Si l'image possède P pixels, cette étape est donc réalisée P fois par itération.

Les figures 5.1, 5.2 et 5.3 modélisent l'imbrication des fonctions principales pour chaque partie. Les fonctions sont données sous le format `NomDeLaClasse::NomDeLaFonction`. Les arguments des fonctions sont omis pour alléger la lecture, mais toutes ces fonctions sont reprises dans les annexes pour plus de détails. A chaque schéma est également associée une explication plus précise de la procédure.

Le premier niveau de l'interpolation est décrit à la figure 5.1. La première fonction de ce schéma est le constructeur de la classe *itkBSplineInterpolateImageFunction*. Ce dernier construit un objet qui, ici, est un interpolateur pour images utilisant des B-splines. Il initialise les attributs de l'objet, puis fait appel au constructeur de la classe *BSplineDecompositionImageFilter*. Un deuxième objet est donc créé (le filtre calculant les coefficients d'interpolation) et ses attributs initialisés. Ce constructeur fait ensuite appel à la fonction `BSplineDecompositionImageFilter::SetSplineOrder` afin de fixer l'ordre des B-splines à utiliser pour le calcul des coefficients. Cette fonction fait ensuite elle-même appel à la méthode `BSplineDecompositionImageFilter::SetPoles`, qui fixe simplement les pôles du filtre spline direct, cf. Chapitre 3, Section 3.2.2. On retombe ensuite dans le constructeur initial, celui de l'interpolateur. Celui-ci fixe également l'ordre des B-splines à utiliser dans l'interpolant via la fonction `BSplineInterpolateImageFunction::SetSplineOrder`. L'interpolateur est ainsi prêt à l'usage.

Le deuxième niveau de la phase d'interpolation est illustré par le schéma 5.2. Chaque itération de l'algorithme commence avec la fonction `InitializeIteration` de la classe *itkLevelSetMotionRegistrationFunction*. Elle effectue toutes les opérations touchant à l'image entière et non à un pixel particulier, c'est-à-dire le lissage de l'image avant calcul du gradient ainsi que l'initialisation des interpolants (l'un pour l'image lissée, l'autre pour l'image non lissée). Pour ce faire, elle fait appel pour chaque interpolant à une fonction `SetInputImage` qui fixe l'image à interpoler à la fois pour l'interpolateur et pour le filtre calculant les coefficients. Ensuite, elle enclanche le calcul des coefficients via la fonction `Update`, qui déclenche tout un pipeline de méthodes pour arriver

FIGURE 5.1 – Enchainement des appels de fonctions pour la déclaration et l'initialisation de l'interpolateur - Niveau 1 de l'interpolation

finalement à la classe *itkBSplineDecompositionImageFilter*, dans la fonction `GenerateData`. Cette méthode est présente dans tous les filtres et son but est principalement d'allouer de la mémoire pour les données de sortie. Elle fait ensuite appel à `DataToCoefficientsND` qui est chargée du calcul général des coefficients : par exemple, pour une image 2D dont la définition est $m_1 \times m_2$, cette méthode calcule les coefficients de l'interpolation en faisant appel $m_1 + m_2$ fois à la fonction

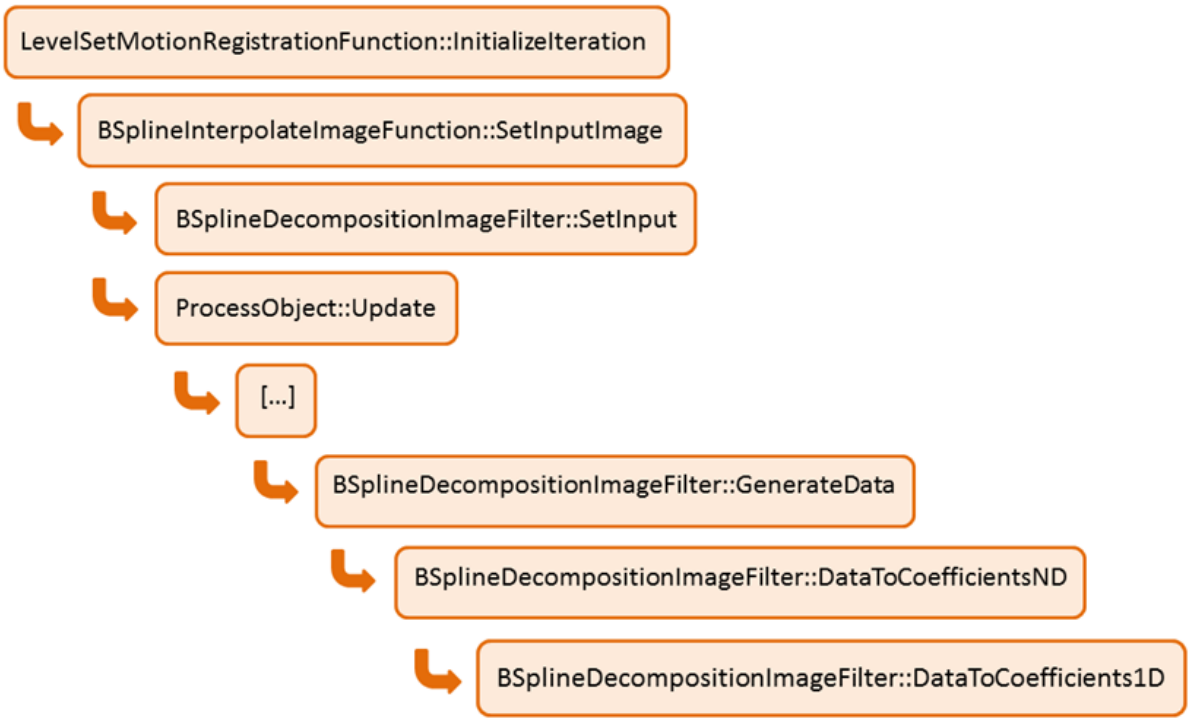


FIGURE 5.2 – Enchainement des appels de fonctions pour le choix de l'image à interpoler et le calcul des coefficients - Niveau 2 de l'interpolation

`DataToCoefficients1D`, chargée du calcul des coefficients d'une interpolation 1D. Concrètement, ITK initialise l'image d'output destinée aux coefficients d'interpolation avec l'image à interpoler. A l'aide de celle-ci, elle effectue ensuite m_2 calculs de coefficients 1D selon la direction x, stocke les résultats dans la même image et recommence pour la direction y (m_1 fois). C'est donc pour cela que la classe gérant les coefficients d'interpolation est un filtre : elle prend en entrée l'image à interpoler et donne en sortie une image de même définition avec, en chaque pixel, le coefficient d'interpolation associé. Notons que la procédure 2D décrite ici peut facilement être étendue aux cas 3D ou plus.

Finalement, le troisième niveau de l'interpolation est schématisé à la figure 5.3. La fonction de départ est celle qui calcule une itération de l'algorithme. Ainsi, lorsqu'elle a besoin de l'intensité en un point, elle fait appel à `BSplineInterpolateImageFunction::Evaluate` qui transforme le point en lequel on souhaite évaluer l'interpolant en index continu et passe ensuite la main à la fonction `EvaluateAtContinuousIndex` de la même classe. Celle-ci déclare des variables de travail et les passe, ainsi que l'index, à `EvaluateAtContinuousIndexInternal`, la fonction au coeur de l'interpolation. Celle-ci commence par fixer la région de support des B-splines à utiliser pour interpoler au point demandé via la fonction `DetermineRegionOfSupport`. Ensuite, elle évalue les B-splines sur la région de support grâce à `SetInterpolationWeights`. Un exemple illustre cela à la figure 5.4 pour le cas des B-splines cubiques. On observe les poids de l'interpolation en orange et les points de la région de support sont entourés en bleu. Enfin, l'interpolateur fait appel à la fonction `ApplyMirrorBoundaryConditions` qui modifie la région de support dans le cas où les index qu'elle contient sont en dehors des index de l'image. Cela est fait de manière miroir puisque les B-splines sont symétriques. L'évaluation de l'interpolant est ensuite simplement réalisée via une double boucle, l'une portant sur les points servant à interpoler, l'autre portant sur la dimension de l'image.

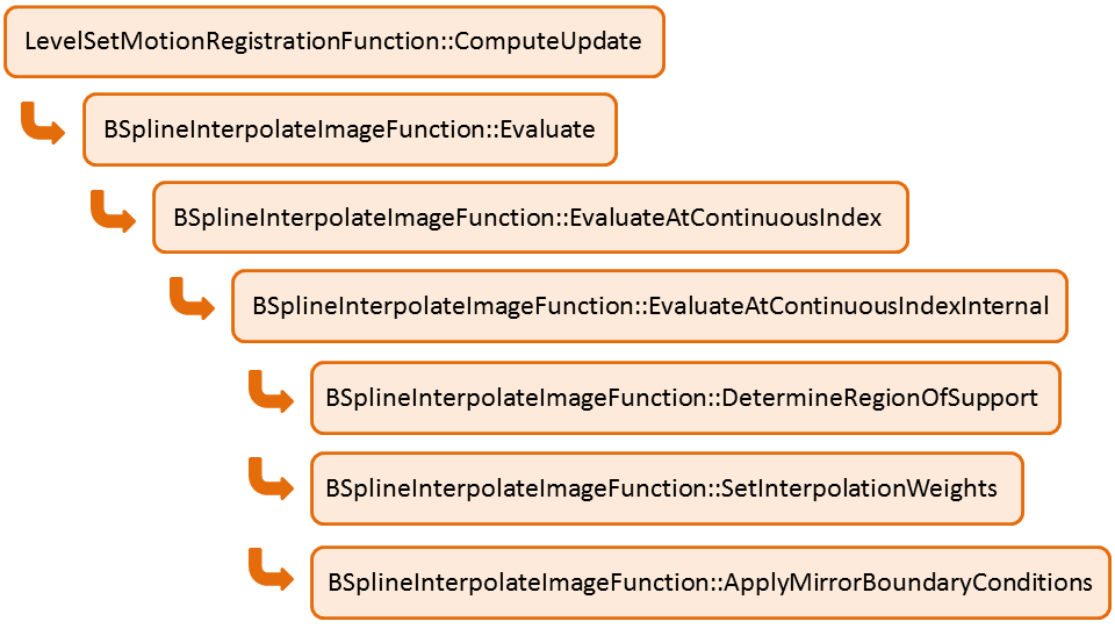


FIGURE 5.3 – Enchainement des appels de fonctions pour l'étape d'évaluation de l'interpolant en un pixel précis - Niveau 3 de l'interpolation

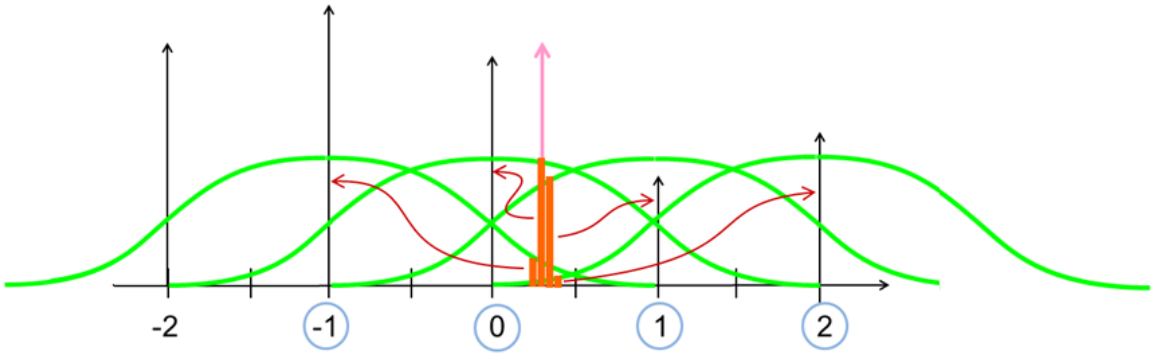


FIGURE 5.4 – B-splines intervenant dans l'interpolation en un point, modifié de [22]

5.5 Calcul du gradient de l'image

Nous avons déjà vu que, pour calculer le gradient de l'image dans l'algorithme des ensembles de niveau, ITK utilise des différences finies *minmod*. Le but de ce mémoire est de comparer cette implémentation avec une implémentation par dérivées exactes. Le problème est que l'image est discrète ; il faut donc dériver l'interpolant, qui est lui continu.

Nous verrons donc ici premièrement comment ITK effectue son calcul de différences finies, avant d'introduire une implémentation alternative par dérivées exactes et de vérifier numériquement que le codage a été bien effectué.

5.5.1 Implémentation d'ITK par différences finies

Rappelons tout d'abord que les différences finies *minmod* de $C_{ij} = G_\sigma * I_{i-u,j-v}$ sont données par

$$\begin{cases} (C_x)_{ij} = m(D_x^+ C_{ij}, D_x^- C_{ij}), \\ (C_y)_{ij} = m(D_y^+ C_{ij}, D_y^- C_{ij}), \end{cases}$$

où

$$m(x,y) = \begin{cases} \operatorname{sign}(x) \min(|x|,|y|) & \text{si } xy > 0 \\ 0 & \text{sinon.} \end{cases}$$

La librairie ITK réalise ce calcul dans la fonction `ComputeUpdate` de la classe *itkLevelSetMotionRegistrationFunction*, donnée en annexe. Cette fonction implémente une itération de l'algorithme pour un pixel précis. Le pseudo-code du processus de dérivation est repris à l'algorithme 5.1, où I_{G_σ} fait référence à l'image lissée dont on calcule le gradient.

Notons qu'ITK utilise par défaut le spacing de l'image comme pas pour les différences finies. Cela peut donc mener à des approximations du gradient de mauvaise qualité lorsque le spacing est assez éloigné de zéro.

Algorithme 5.1 Calcul des différences finies *minmod* en un point $\vec{V}(P)$

$forwardD = \dfrac{I_{G_\sigma}(\vec{V}(P) + spacing) - I_{G_\sigma}(\vec{V}(P))}{spacing}$ par rééchantillonnage de l'image lissée

$backwardD = \dfrac{I_{G_\sigma}(\vec{V}(P)) - I_{G_\sigma}(\vec{V}(P) - spacing)}{spacing}$ par rééchantillonnage de l'image lissée

if $forwardD * backwardD > 0$ **then**
  $a = \min(|forwardD|, |backwardD|)$
  $gradient = a * sign(forwardD)$
else
  $gradient = 0$
end if

5.5.2 Calcul du gradient par dérivées exactes

Rappelons qu'au Chapitre 3 nous avions vu que la dérivée d'un interpolant $s(x)$ d'ordre n et de dimension d était donnée par l'équation (3.4), i.e.,

$$\frac{\partial s}{\partial x_i}(x_1, ..., x_d) = \sum_{k_1 \in \mathbb{Z}} ... \sum_{k_d \in \mathbb{Z}} c_{k_1 ... k_d} \beta^n(x_1 - k_1) ... \beta^n(x_{i-1} - k_{i-1}) D_i \beta^n(x_{i+1} - k_{i+1}) ... \beta^n(x_d - k_d),$$

où les c_k sont les coefficients de l'interpolant s et

$$D_i = \beta^{n-1}\Big(x_i - k_i + \frac{1}{2}\Big) - \beta^{n-1}\Big(x_i - k_i - \frac{1}{2}\Big).$$

Nous pouvons donc reprendre le même calcul des coefficients que lors de l'interpolation, les mêmes évaluations de B-splines pour toutes les composantes sauf x_i, pour laquelle il nous faut évaluer deux B-splines de l'ordre inférieur.

En réalité, une grande partie du travail est déjà effectuée par ITK puisque la classe *itkBSplineInterpolateImageFunction* contient une fonction `EvaluateDerivative`. Le processus est alors similaire à celui de l'évaluation de l'interpolant, si ce n'est qu'au lieu d'évaluer

$$s(x_1, ..., x_d) = \sum_{k_1 \in \mathbb{Z}} ... \sum_{k_d \in \mathbb{Z}} c_{k_1 ... k_d} \beta^n(x_1 - k_1) \dots \beta^n(x_d - k_d),$$

on évalue (3.4). C'est d'ailleurs comme cela que fonctionne ITK.

Le schéma de la pile des appels des fonctions principales est donné à la figure 5.5. Similairement au niveau 3 de la phase d'interpolation, la fonction principale pour l'évaluation du gradient est `EvaluateDerivativeAtContinuousIndexInternal`. Celle-ci crée la région de support des B-splines à utiliser et évalue ces B-splines au point d'intérêt via la formule que nous avions établie au Chapitre 3, Section 3.1.4. Elle applique également si nécessaire des conditions miroir sur les index à utiliser. Les dérivées selon chaque composante sont ensuite calculées via (3.4).

Pour s'assurer que ce calcul de la dérivée exacte de l'image est correctement effectué, nous pouvons nous intéresser à la borne d'erreur lorsque l'on approxime le gradient par des différences finies [36]. Supposons que nous cherchions à calculer le gradient de l'interpolant I d'une image à d dimensions et que cet interpolant soit deux fois continûment différentiable. Ce sera bien notre cas grâce à la propriété 3.1 des B-splines. Le théorème de Taylor ([36]) nous dit que, si $x, p \in \mathbb{R}^d$,

$$I(x + p) - I(x) - \nabla I(x)^T p = \frac{1}{2} p^T \nabla^2 I(x + tp) p \quad \text{pour } t \in (0, 1).$$

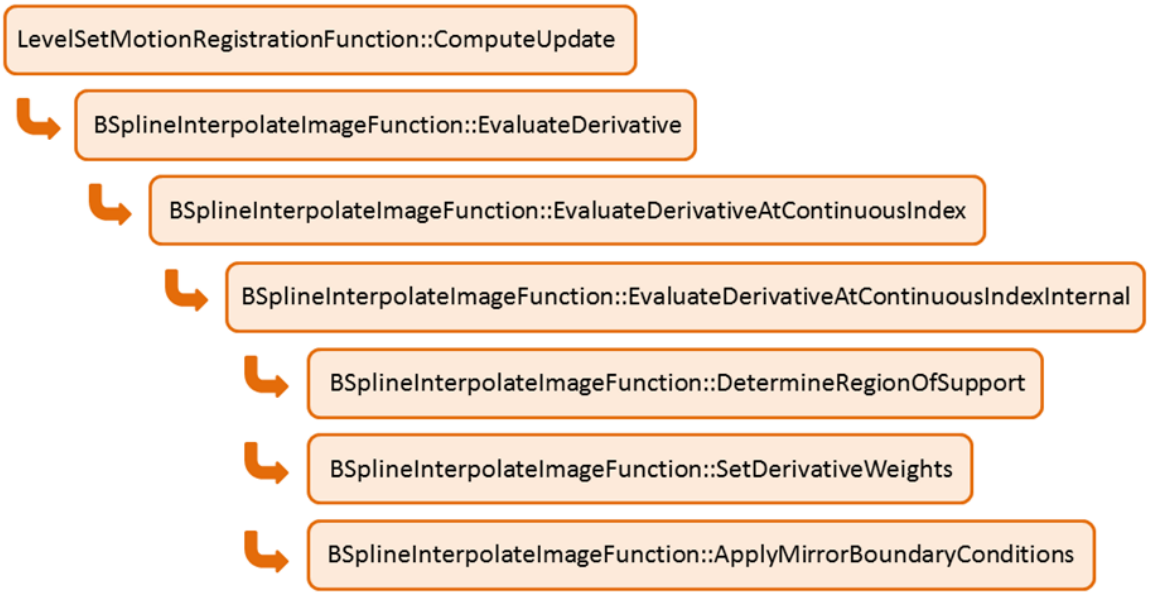


FIGURE 5.5 – Enchainement des appels de fonctions pour le calcul de la dérivée de l'interpolant en un point

Soit $L = \sup_{y \in [x, x+p]} \|\nabla^2 I(y)\|$, nous avons dès lors

$$|I(x+p) - I(x) - \nabla I(x)^T p| \leq \frac{L}{2}\|p\|^2.$$

Si nous choisissons $p = \epsilon e_i$ pour obtenir une dérivée de I par rapport à x_i, où les vecteurs e_i $(i = 1, ..., d)$ sont les vecteurs de la base canonique de $\mathbb{R}^d$, nous obtenons

$$|I(x + \epsilon e_i) - I(x) - \nabla I(x)^T \epsilon e_i| \leq \frac{L}{2}\epsilon^2,$$

i.e.,

$$\left| \frac{I(x + \epsilon e_i) - I(x)}{\epsilon} - \frac{\partial I}{\partial x_i} \right| \leq \frac{L}{2}\epsilon.$$

La marge d'erreur pour l'approximation du gradient de I par une différence finie forward est donc donnée par

$$\frac{L}{2}\epsilon, \tag{5.1}$$

où ϵ est le pas choisi. Cette borne est identique pour une différence backward. Dans notre cas, nous nous intéressons aux différences finies *minmod*. Celles-ci sont données soit par une différence finie forward, soit par une différence finie backward, soit par zéro. Ce dernier cas n'arrive cependant que lorsque les approximations forward et backward sont de signe opposé, c'est-à-dire lorsqu'on est proche d'un extremum local. L'approximation du gradient par zéro donnera donc une erreur moindre qu'au moins une des deux différences finies possibles. Nous pouvons donc garder la borne (5.1) comme marge d'erreur générale pour les différences finies *minmod*.

Il nous faut cependant également tenir compte des erreurs numériques survenant lors du calcul des gradients. On estime en général que l'erreur relative commise lors de l'évaluation d'une fonction

est de l'ordre de la précision machine u, c'est-à-dire 10^{-16} en double précision [36]. En faisant cette hypothèse et en définissant une borne $L_I = \sup_{y\in[x,x+\epsilon e_i]} |I(y)|$, nous avons alors

$$\begin{aligned} |\text{comp}(I(x)) - I(x)| &\leq uL_I, \\ |\text{comp}(I(x+\epsilon e_i)) - I(x+\epsilon e_i)| &\leq uL_I, \end{aligned}$$

où comp(.) désigne la valeur calculée. De plus, en définissant $M = \sup_{y\in[x,x+\epsilon e_i]} \left|\frac{\partial I}{\partial x_i}(y)\right|$, nous avons également

$$\left|\text{comp}\left(\frac{\partial I}{\partial x_i}\right) - \frac{\partial I}{\partial x_i}\right| \leq uM,$$

ce qui implique ainsi finalement que

$$\left|\frac{I(x+\epsilon e_i) - I(x)}{\epsilon} - \frac{\partial I}{\partial x_i}\right| \leq \frac{L}{2}\epsilon + \frac{2uL_I}{\epsilon} + uM.$$

Notons cette marge d'erreur $E(\epsilon)$. Afin de trouver le pas ϵ optimal, nous pouvons minimiser E, ce qui donne

$$\epsilon^2 = \frac{4L_I u}{L}.$$

En supposant que le problème est bien échelonné, c'est-à-dire qu'un petit changement de x ne produit pas un grand changement de I, on peut choisir un pas de l'ordre de $\sqrt{u}$. Fixons donc $\epsilon = \sqrt{u}$; considérant le terme uM comme négligeable, l'erreur totale devient alors inférieure à

$$\begin{aligned} E(\sqrt{u}) &= \frac{L}{2}\sqrt{u} + 2\sqrt{u}L_I \\ &= \sqrt{u}\left(\frac{L}{2} + 2L_I\right). \end{aligned} \tag{5.2}$$

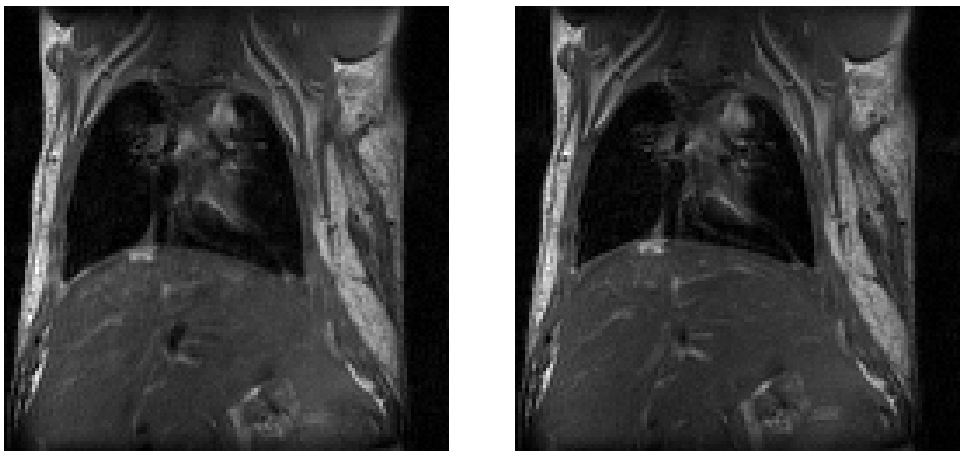
La borne d'erreur dépend donc de l'intensité de l'image et de la valeur de son hessien. En double précision pour la dérivation d'une image 8-bit, l'erreur maximale devrait être comprise, grosso modo, entre 10^{-8} et 10^{-6}.

Afin de vérifier l'implémentation du gradient exact, calculons le gradient des deux manières possibles pour la première itération de notre algorithme de recalage appliqué à deux images 2D provenant des exemples fournis par ITK. La figure 5.6 montre ces images, qui représentent la cage thoracique d'un rat. Elles sont définies sur 256 niveaux de gris. Ici, nous allons simplement calculer les gradients en chaque point de l'image mouvante puisque nous ne considérons qu'une seule itération. Comme la définition de l'image est de 128 × 128, cela nous donne déjà accès à 16384 évaluations du gradient.

Après calcul des deux gradients (exact et par différences finies *minmod* avec $\epsilon = \sqrt{u}$), nous avons tracé les différences absolues entre eux, selon x puis selon y. Les résultats sont donnés à la figure 5.7 avec une échelle logarithmique. On y observe que la plupart des points sont de l'ordre de 10^{-8}, 10^{-7} ou 10^{-6}. Cela semble assez normal, mais il nous faut tout de même vérifier que les points d'un ordre supérieur à $\sqrt{u}$ sont bien dûs au facteur

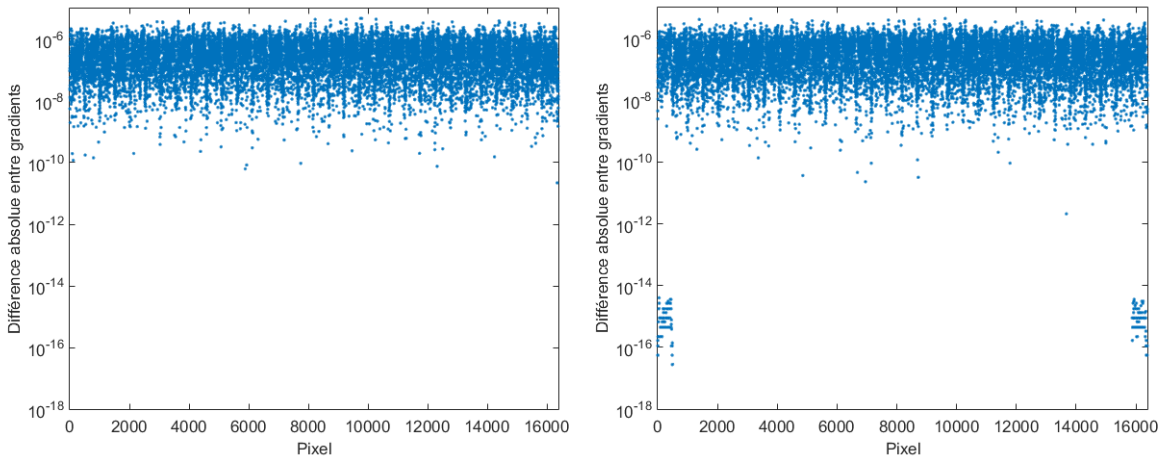
$$\frac{L}{2} + 2L_I$$

de l'expression (5.2). Pour se faire une première idée, nous pouvons visualiser la différence absolue entre les deux gradients sous forme d'image (logarithmique) ; un pixel est ainsi donné par la différence absolue entre gradients évalués au pixel correspondant de l'image mouvante. Nous observons



(a) (b)

FIGURE 5.6 – (a) Image fixe (b) Image mouvante [21]



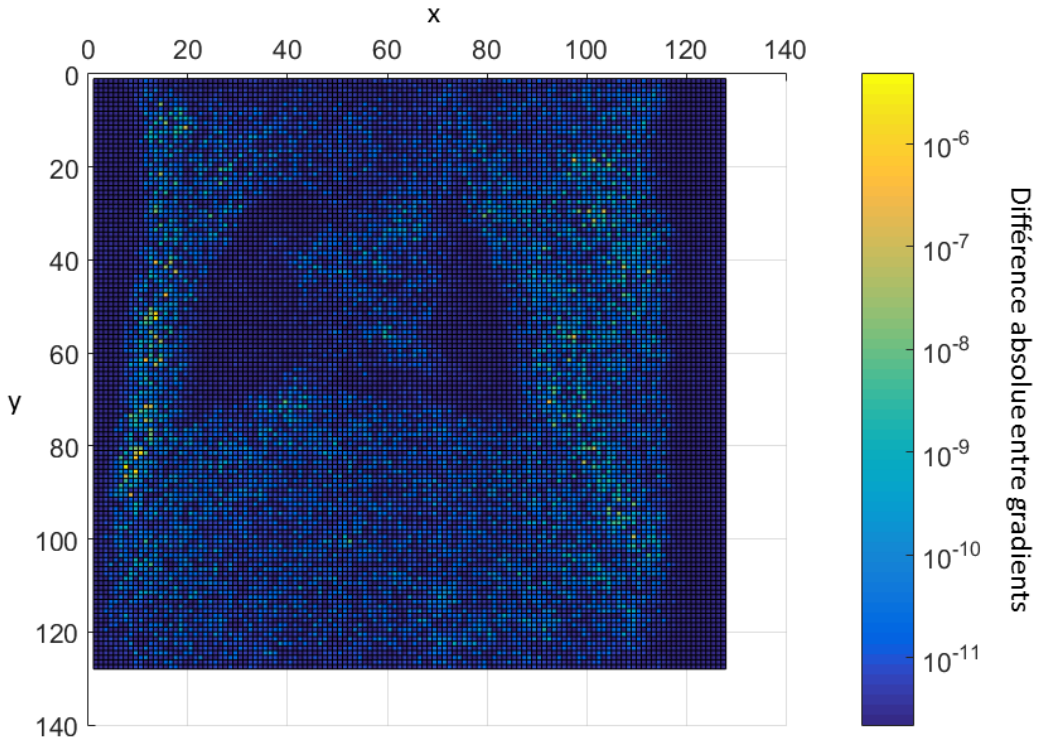
(a) (b)

FIGURE 5.7 – (a) Différence absolue entre gradients selon x (b) Différence absolue entre gradients selon y

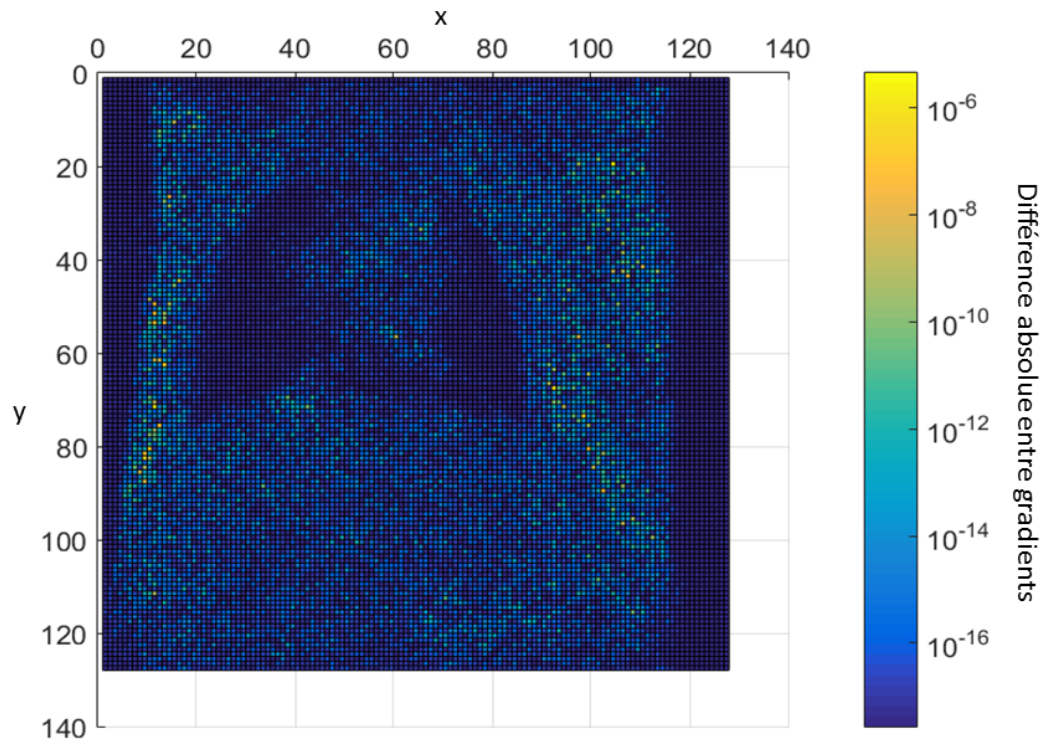
ainsi à la figure 5.8 que la plupart des pixels sont dans les tons bleus, ce qui indique un écart entre gradients inférieur à 10^{-9}. Seuls quelques pixels sont jaunes, indiquant une différence de l'ordre de 10^{-7} ou 10^{-6}. Si l'on compare avec l'image initiale, figure 5.6(b), on remarque que ces importants écarts sont dans des zones ayant, de base, une forte intensité. Rappelons en effet que plus le pixel est clair, plus la valeur de la fonction image est élevée. Au vu de la marge (5.2), cela pourrait donc expliquer l'écart plus élevé entre gradients à ces endroits puisque la borne L_I serait alors de l'ordre de 10^1 ou 10^2.

Pour nous en assurer, repartons de l'équation (5.2). Nous avions

$$\left| \frac{I(x + \sqrt{u}e_i) - I(x)}{\epsilon} - \frac{\partial I}{\partial x_i} \right| \leq \sqrt{u}\left(\frac{L}{2} + 2L_I\right).$$



(a)



(b)

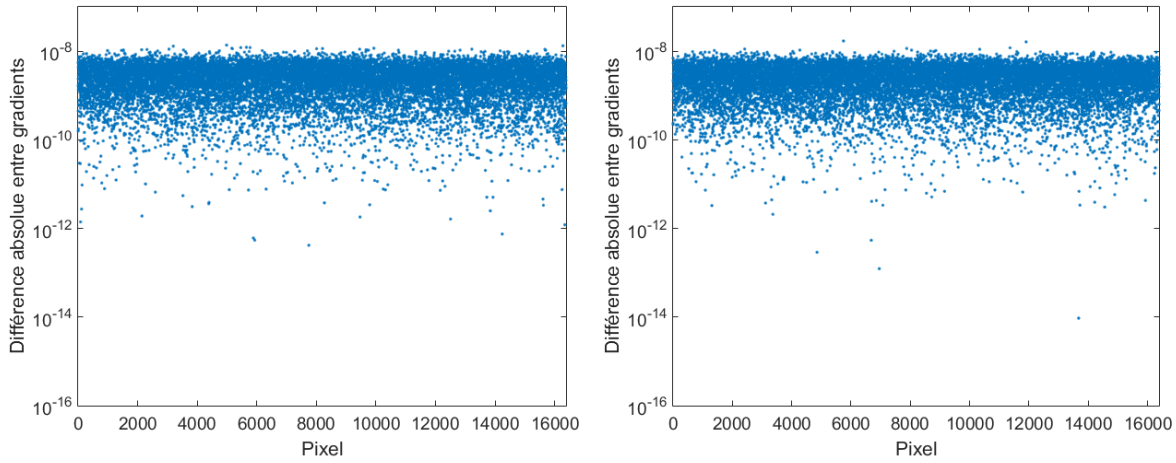
FIGURE 5.8 – (a) Graphe logarithmique de la différence absolue des deux gradients par rapport à x en chaque pixel (b) Graphe logarithmique de la différence absolue des deux gradients par rapport à y en chaque pixel

Nous pouvons donc simplement calculer

$$D := \frac{\left| \frac{I(x + \sqrt{u}e_i) - I(x)}{\epsilon} - \frac{\partial I}{\partial x_i} \right|}{\frac{L}{2} + 2L_I}$$

et vérifier que le nombre obtenu est bien de l'ordre de $\sqrt{u}$ ou moins. Comme nous ne connaissons ni L ni L_I, nous allons les choisir comme étant respectivement la valeur de l'interpolant et de la norme du hessien en chaque point de l'image. En effet, si le problème est bien échelonné, l'ordre obtenu devrait être le bon. De plus, le dénominateur ne devrait jamais être négatif ; après vérification c'est d'ailleurs bien le cas. Notons que nous avons ici calculé le hessien par différences finies forward.

Afin de visualiser les résultats, affichons les nombres obtenus sur des graphes à échelle logarithmique (figure 5.9). On observe que toutes les valeurs sont bien d'un ordre maximal de 10^{-8} ; le calcul du gradient par dérivées exactes semble donc finalement bien correct.



(a) (b)

FIGURE 5.9 – (a) Nombre D évalué en chaque pixel avec gradients par rapport à x (b) Nombre D évalué en chaque pixel avec gradients par rapport à y

5.6 Conclusion

Dans ce chapitre, nous avons tout d'abord introduit l'outil utilisé pour tester l'algorithme décrit dans les chapitres précédents. Il s'agit d'ITK, une librairie du C++.

L'algorithme qui nous intéresse était déjà implémenté dans ITK, et nous avons ici tenté d'en expliquer la structure générale, assez complexe. Nous avons ainsi abordé l'implémentation de la mise à jour d'un champ de déplacement pour l'image mouvante de manière explicite ainsi que celle de la phase d'interpolation.

Enfin, nous avons abordé un point clé de ce travail qu'est le calcul du gradient de l'image avec ITK. La librairie utilise des différences finies *minmod*, que nous avons de notre côté remplacées par des dérivées exactes de l'interpolant. Nous avons ensuite vérifié notre implémentation via des tests sur une image, comparant les différences théoriques et expérimentales entre les deux implémentations différentes du gradient.

Chapitre 6

Expérimentations et résultats

Ce chapitre présente les différentes expérimentations de l'algorithme des ensembles de niveau pour le recalage d'images ainsi que les résultats obtenus en fonction de la méthode de calcul du gradient utilisée. La première section explique la démarche effectuée pour choisir les différents paramètres de l'algorithme. Nous exposons ensuite les résultats obtenus pour des images médicales 2D avant d'étudier un cas 3D. Notons que tous les résultats présentés ici ont été obtenus avec un processeur Intel Core i5-2410M 2301 MHz.

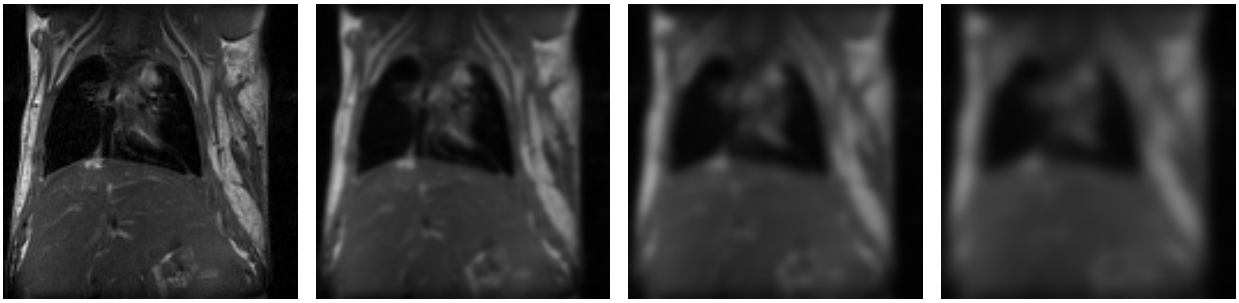
6.1 Choix des paramètres

L'algorithme des ensembles de niveau utilise peu de paramètres. Le recalage étant un processus devant souvent être réalisé rapidement, nous ne pouvons nous permettre de laisser le choix des paramètres être fonction des images à recaler. Il nous faut donc établir une fois pour toutes ces valeurs. Les paramètres en question sont l'écart-type σ du lissage gaussien et la constante α stabilisant le déplacement en cas de gradient proche de zéro, intervenant dans les équations d'évolution corrigées à la Section 5.3.1. Nous devons également penser à fixer le pas temporel Δt du schéma de résolution numérique de l'EDP principale devant respecter la restriction CFL donnée par le théorème 5.1 ainsi que le nombre maximal d'itérations de l'algorithme.

Commençons par ce dernier. Nous choisissons ici de fixer la limite du nombre d'itérations à 300. En effet, c'est déjà beaucoup sachant que le recalage doit souvent se faire rapidement.

Passons maintenant au choix d'α. Cette constante doit être petite et positive afin de stabiliser la solution dans le cas où la norme du gradient serait très faible. Les auteurs de la méthode de recalage d'images ne proposant aucune valeur spécifique, nous choisissons simplement ici de garder la valeur par défaut proposée par ITK : $\alpha = 0.1$.

Intéressons-nous à présent à la valeur de l'écart-type dans le lissage gaussien de l'image avant calcul du gradient. Cette étape est destinée à éliminer le bruit de l'image afin de ne pas obtenir de grandes variations dans le gradient qui ne seraient pas significatives. Il faut cependant conserver un maximum l'information réelle de l'image. Pour se faire une idée, nous pouvons appliquer un lissage gaussien à l'image mouvante 5.6(b) utilisée dans le chapitre précédent pour la vérification du calcul du gradient exact. Prenons $\sigma = 1$, $\sigma = 2$ et $\sigma = 3$ et comparons les images obtenues avec l'image mouvante originale. A la vue de la figure 6.1, il est clair que choisir σ égal à 3 est trop ; on perd beaucoup de détails de l'image. De plus, prendre $\sigma = 1$ donne déjà selon nous un bon débruitage là où $\sigma = 2$ provoque déjà quelques pertes d'information. Nous garderons donc $\sigma = 1$ pour tous les tests que nous réaliserons, qui est d'ailleurs la valeur utilisée par l'auteur de



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

FIGURE 6.1 – (a) Image originale, (b) Image lissée avec $\sigma = 1$, (c) Image lissée avec $\sigma = 2$, (d) Image lissée avec $\sigma = 3$

la méthode des ensembles de niveaux pour le recalage d'images.

Enfin, il nous faut choisir un pas temporel Δt pour la résolution de l'EDP décrivant l'évolution du champ de déformation. Cependant, les valeurs que nous choisissons en pratique sont celles à donner à Δx et Δy, les pas spatiaux intervenant dans le calcul par différences finies du gradient. Rappelons en effet que par la restriction CFL, le pas Δt doit être majoré comme suit :

$$\Delta t \leq \frac{1}{\frac{|s_x|}{\Delta x} + \frac{|s_y|}{\Delta y}},$$

et qu'en pratique nous prenons

$$\Delta t = \frac{1}{\max\left\{\frac{|s_x|}{\Delta x} + \frac{|s_y|}{\Delta y}\right\}}.$$

Ainsi, plus nous choisirons Δx et Δy grands, plus Δt sera grand.

Pour l'implémentation par différences finies, ITK fixe par défaut Δx et Δy à 1 (pour le calcul de Δt uniquement), comme le propose l'auteur de la méthode dans son article [49]. Nous laisserons donc cela tel quel. Remarquons que cela ne correspond donc pas forcément au spacing de l'image et donc pas forcément non plus aux pas utilisés dans le calcul du gradient.

Pour l'implémentation avec dérivées exactes, par contre, Δx et Δy sont supposés infiniment petits par définition ; il est donc probable que les fixer à 1 provoque une certaine instabilité dans la solution. A nouveau, effectuons différents tests sur l'image mouvante 5.6(b) afin de fixer les paramètres concernés. Nous affichons ici la valeur de la métrique qui, pour rappel, est la somme des différences au carré moyenne (MSD). Nous avons fixé successivement Δx et Δy égaux à 1, 0.5, 0.3, 0.2 et 0.1 en effectuant maximum 300 itérations. Les résultats se trouvent à la figure 6.2. Nous observons que plus les paramètres Δx et Δy sont grands, plus la métrique décroît vite. Cela semble assez logique puisque plus Δt est grand, plus vite on converge vers la solution. Cependant, le graphe de droite nous permet de nous rendre compte de l'instabilité engendrée par le choix de certains pas. C'est notamment frappant pour le choix de la valeur 1. Bien sûr, le choix de Δx et Δy à 0.1 est celui provoquant le moins d'instabilité, mais il ne permet par contre pas de minimiser la métrique autant que les autres. Cela pourrait s'améliorer en ajoutant des itérations, mais 300 constitue déjà un nombre important d'itérations et, encore une fois, le recalage est souvent limité par le temps. Ici, un bon compromis nous semble être $\Delta x = \Delta y = 0.3$, qui engendre peu d'instabilité (même moins que pour 0.2 étonnamment) tout en convergeant assez rapidement vers la solution. Pour tous les tests que nous réaliserons par la suite, nous fixerons donc $\Delta x = \Delta y = 0.3$.

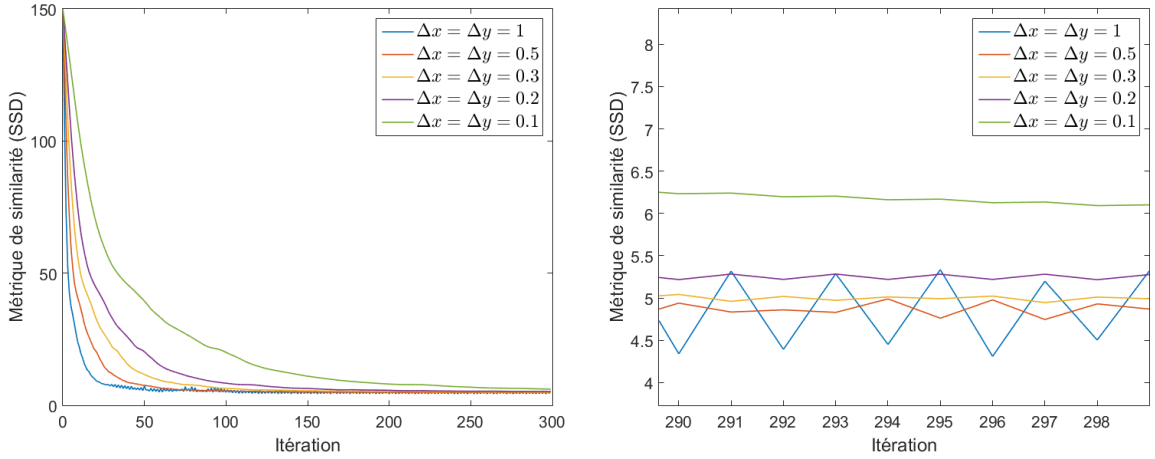


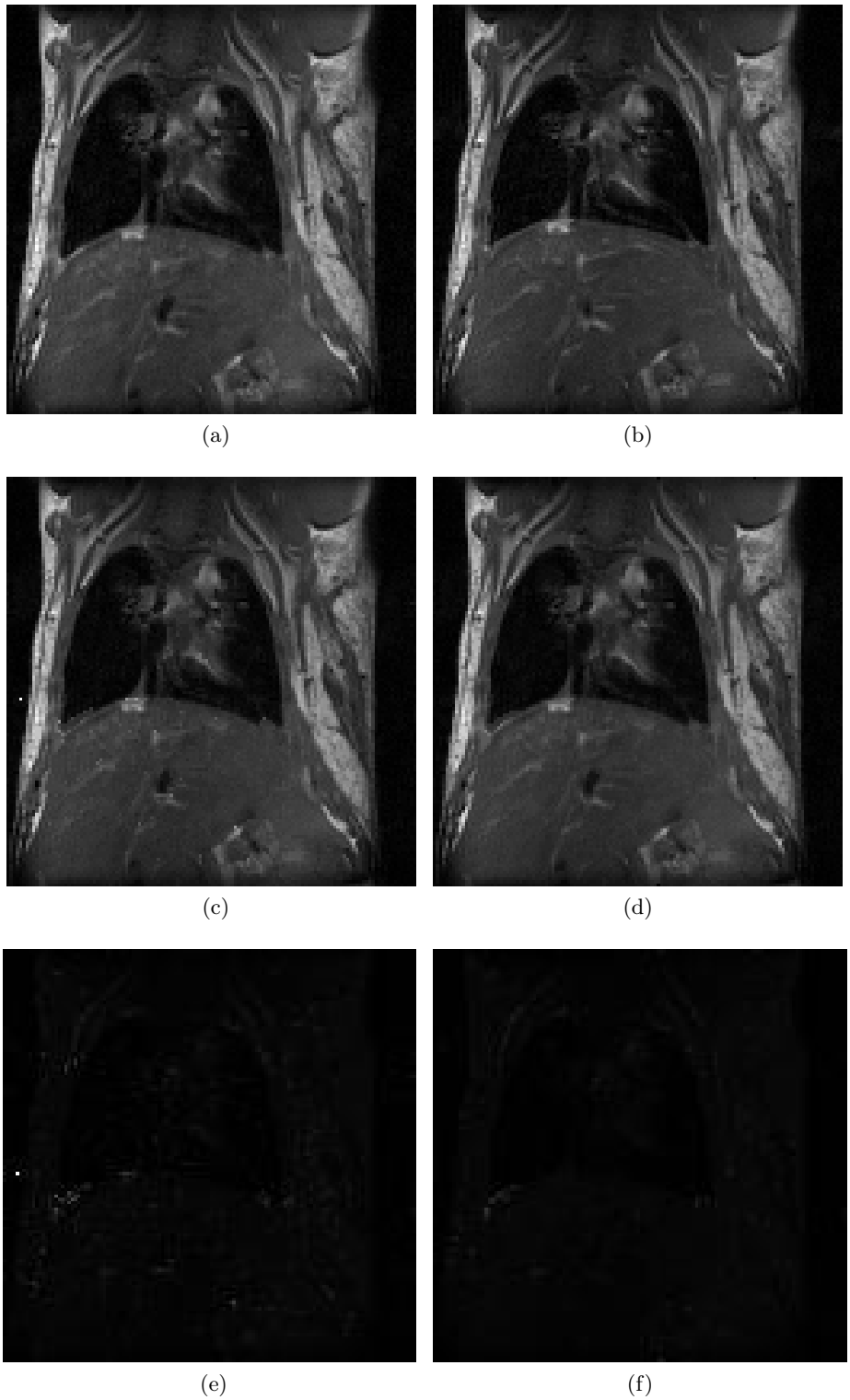
FIGURE 6.2 – (a) Valeur du critère de similarité MSD en fonction de l'itération (b) Zoom sur les dernières itérations

6.2 Application aux images médicales 2D

Nous avons commencé par appliquer l'algorithme des ensembles de niveau à quatres paires d'images 2D. Pour chaque jeu de données, nous avons comparé l'évolution du critère de similarité ainsi que le temps d'exécution. Nous avons de plus comparé les images montrant la différence absolue entre image fixe et image mouvante recalée. Le premier jeu de données provient des exemples fournis par la librairie ITK, tandis que les trois autres proviennent de la librairie FAIR (*Flexible Algorithms for Image Registration*), un outil de recalage d'images pour *Matlab* [34].

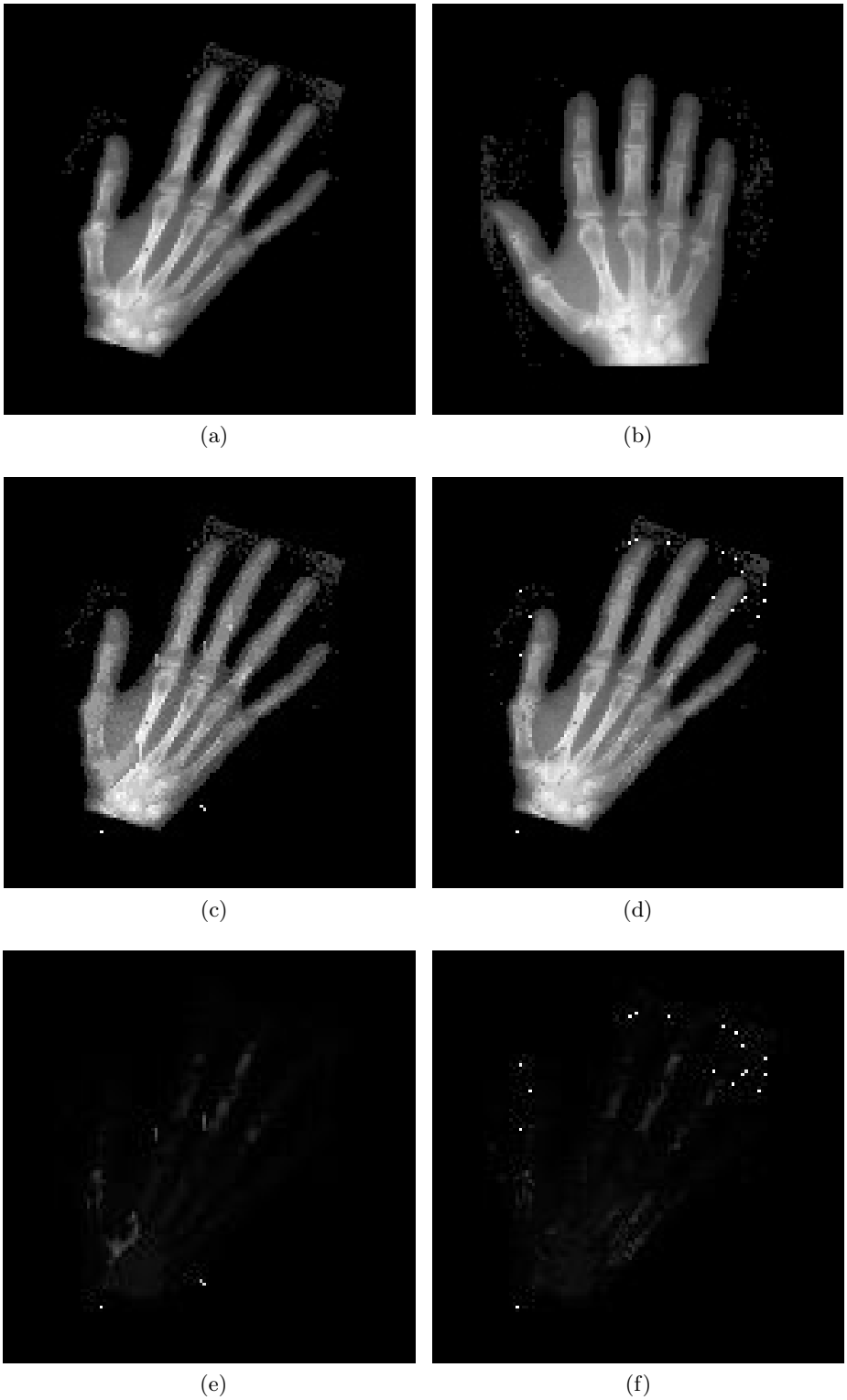
La première paire d'images est le CT des poumons d'un rat ; la dimension des images est 128×128. Comme le montrent les figures 6.3(a) et 6.3(b), les images à recaler sont déjà assez semblables avant le recalage. Le but ici est donc de corriger l'image mouvante par des petites déformations locales. Les images obtenues par les deux méthodes étudiées, i.e., utilisant un gradient approximé ou exact, sont montrées aux figures 6.3(c) et 6.3(d). Il n'est pas facile de les comparer ; la différence absolue entre images apporte ici plus d'informations sur la qualité du recalage. Nous remarquons en effet aux figures 6.3(e) et 6.3(f) que la différence absolue entre image fixe et image recalée obtenue en calculant le gradient par différences finies (figure 6.3(e)) possède quelques points de plus forte intensité à certains endroits de l'image, notamment sur la gauche, que la figure obtenue en utilisant un gradient exact (figure 6.3(f)). Cela semble indiquer un recalage plus uniforme pour la méthode utilisant un gradient exact.

Le deuxième jeu de données, à la figure 6.4, montre un CT d'une main ; la dimension des images est 128×128. Cette fois, la différence entre les images de départ est importante. Les images recalées semblent cependant toutes deux assez proches de l'image fixe et il est difficile de les départager. Cependant, l'image obtenue avec un calcul exact du gradient est globalement plus bruitée et une petite anomalie est observée à la base de l'index et du majeur. La base de la main, sur la gauche, est également moins semblable à l'image fixe que le résultat obtenu avec un gradient exact à la figure 6.4(d). Par contre, celle-ci présente un petit défaut sur la droite de la main, dans la continuation de l'auriculaire et comporte un peu de bruit autour des doigts de la main. Ces observations se confirment d'ailleurs à la vue de la différence absolue entre image fixe et image déformée pour chaque méthode, aux figures 6.4(e) et 6.4(f).



(a) (b) (c) (d) (e) (f)

FIGURE 6.3 – (a) Image fixe, (b) Image mouvante, (c) Image mouvante déformée - Différences finies, (d) Image mouvante déformée - Gradient exact, (e) Différence absolue entre (a) et (c), (f) Différence absolue entre (a) et (d)



(a) (b) (c) (d) (e) (f)

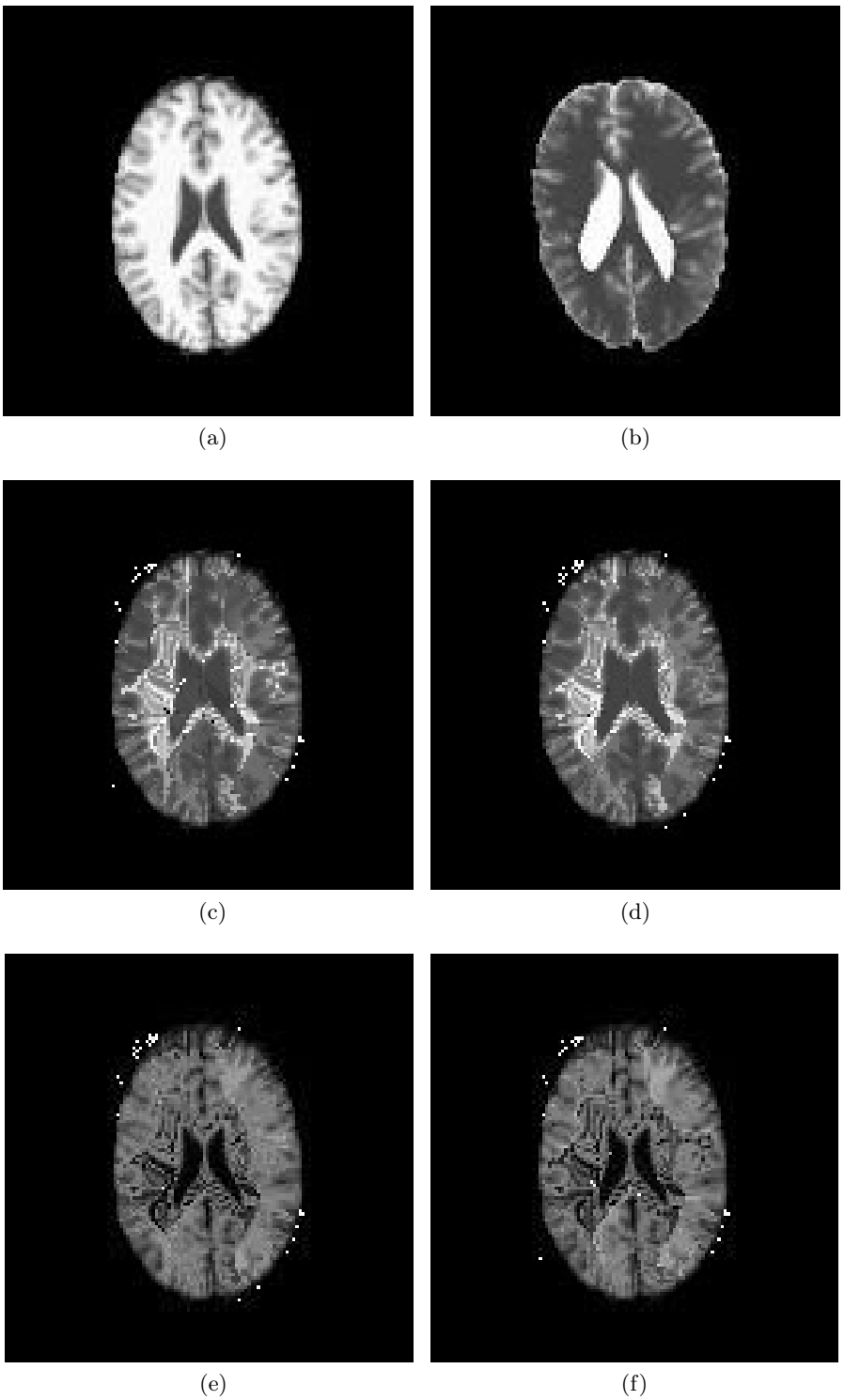
FIGURE 6.4 – (a) Image fixe, (b) Image mouvante, (c) Image mouvante déformée - Différences finies, (d) Image mouvante déformée - Gradient exact, (e) Différence absolue entre (a) et (c), (f) Différence absolue entre (a) et (d)

La troisième paire d'images sur laquelle nous avons testé l'algorithme des ensembles de niveau consiste en des IRM du cerveau (figures 6.5(a) et 6.5(b)). Les dimensions des images de base sont 128×128. Ici, comme l'illustrent les figures 6.5(c), 6.5(d), 6.5(e) et 6.5(f), nous obtenons un recalage de mauvaise qualité. C'est en réalité normal ; l'algorithme des ensembles de niveau a pour principe de faire évoluer les ensembles de niveau de l'image mouvante vers les ensembles de niveau de l'image fixe. Cela présuppose donc que deux points homologues sont de même intensité, ce qui n'est pas le cas ici. Ces images ne sont donc pas adaptées à l'algorithme étudié, ce qui explique un si mauvais résultat.

Enfin, le dernier jeu de données utilisé représente à nouveau un cerveau, mais cette fois-ci les images sont de taille 256×256. La principale différence entre l'image fixe et l'image mouvante, présentées aux figures 6.6(a) et 6.6(b), consiste en une déformation dans la partie supérieure de la représentation du cerveau. Celle-ci semble assez bien corrigée par le recalage, quelle que soit la méthode utilisée. Les résultats, visibles aux figures 6.6(c), 6.6(d), 6.6(e) et 6.6(f), sont cette fois difficilement différenciables ; nous reviendrons sur la qualité de ce recalage.

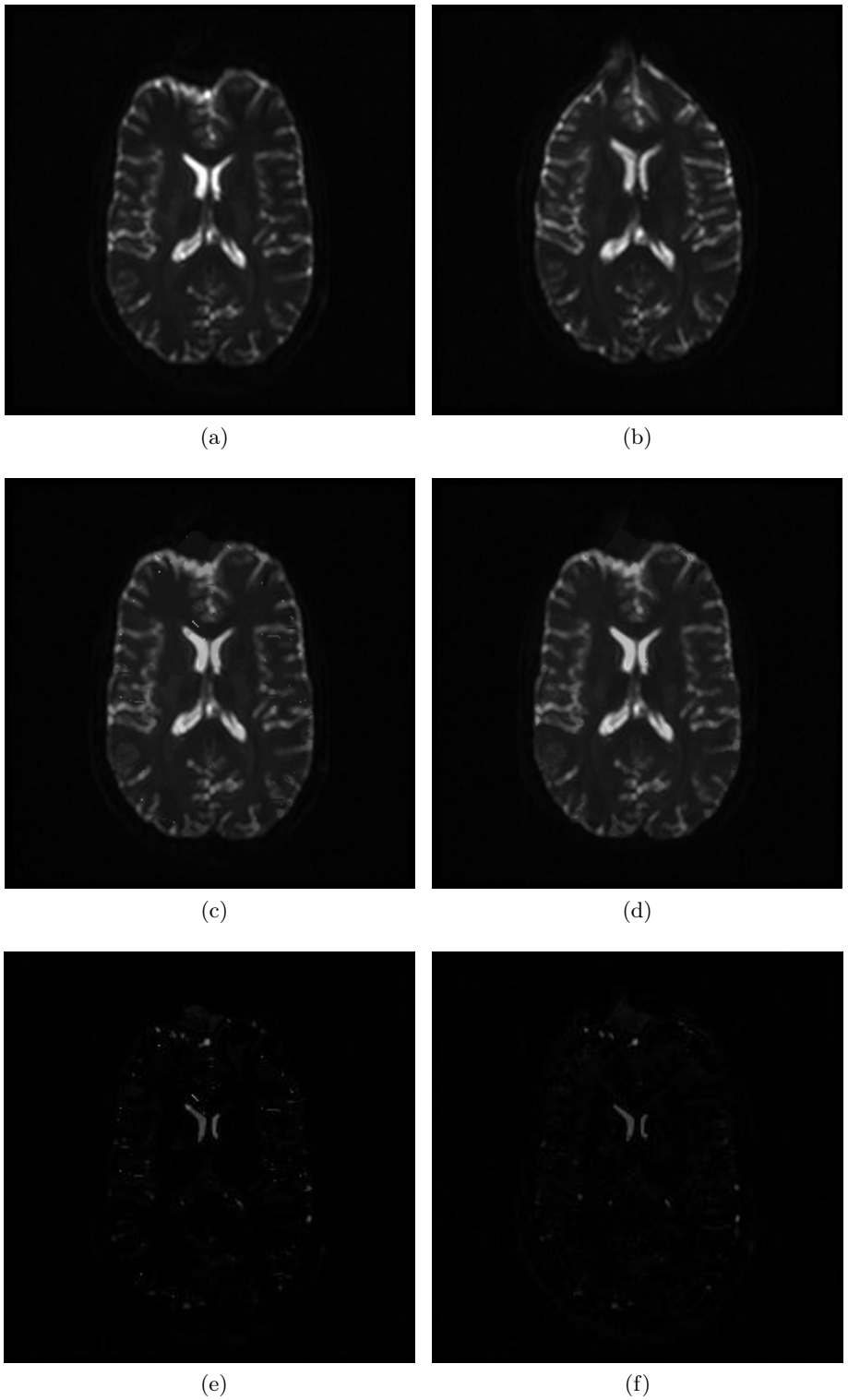
A présent, comparons les deux méthodes de calcul du gradient dans l'algorithme des ensembles de niveau de manière plus mathématique. Traçons, pour les quatre jeux de données, les courbes d'évolution de la métrique pour chaque implémentation ainsi qu'un zoom sur les dernières itérations. Pour rappel, la métrique utilisée est la somme des différences au carré moyenne (MSD). Les résultats sont présentés aux figures 6.7 à 6.10. La table 6.1, elle, reprend le temps d'exécution nécessaire à chaque méthode pour chaque paire d'images. Nous n'étudions pas ici les résultats en termes de nombre d'itérations car le nombre d'itérations maximal fixé est toujours atteint. En effet, il n'y a pas de critère d'arrêt numérique sur la valeur de la métrique pour cet algorithme. C'est dû au fait que la métrique approche rarement zéro et que la qualité du recalage dépend énormément de la différence entre image fixe et image mouvante avant recalage. Fixer un critère d'arrêt commun à toutes les images serait donc difficile. Ici, l'algorithme n'a ainsi que deux possibilités pour s'arrêter : soit il atteint le nombre maximal d'itérations fixé par l'utilisateur, soit la valeur du critère de similarité n'évolue plus (i.e., celle-ci est identique pour l'itération courante et pour l'itération précédente).

A la vue de la figure 6.7, nous constatons tout d'abord que l'implémentation du gradient exact permet ici d'atteindre une bien meilleure précision de recalage que la méthode utilisant des différences finies *minmod*. En effet, la somme des écarts au carré moyenne descend jusque 5 pour le gradient exact, contre environ 19 pour le gradient approximé. Cela est probablement dû au fait que le pas des différences finies est égal au spacing de l'image, i.e., 1 ici, ce qui rend l'approximation très grossière. Cependant, le critère de similarité décroît plus vite pour la méthode utilisant un gradient calculé par différences finies ; c'est simplement dû au fait que nous avons choisi $\Delta t = 1$ pour les différences finies mais $\Delta t = 0.3$ pour le gradient exact. Avec ces images-ci, le choix des différences finies ne semble utile que dans le cas où nous souhaitons effectuer un recalage demandant moins d'une quarantaine d'itérations. Le facteur temps doit néanmoins également être pris en compte. Nous observons à la table 6.1 que le temps nécessaire pour effectuer 300 itérations de l'algorithme est environ double pour l'utilisation de différences finies. Sur base de ce jeu de données, il semblerait donc qu'utiliser des différences finies, comme le fait ITK de base, serait une mauvaise idée. Avant de passer à l'analyse du deuxième recalage effectué, remarquons aussi que la figure 6.7(b) nous montre une métrique légèrement instable pour la méthode utilisant un gradient approximé. Cela illustre que la restriction CFL donnée par la proposition 5.1 est une condition nécessaire mais pas suffisante. Ici, le choix du pas temporel Δt respecte cette restriction mais la métrique semble tout de même quelque peu instable, ce qui indique donc une solution également instable de l'EDP à résoudre.



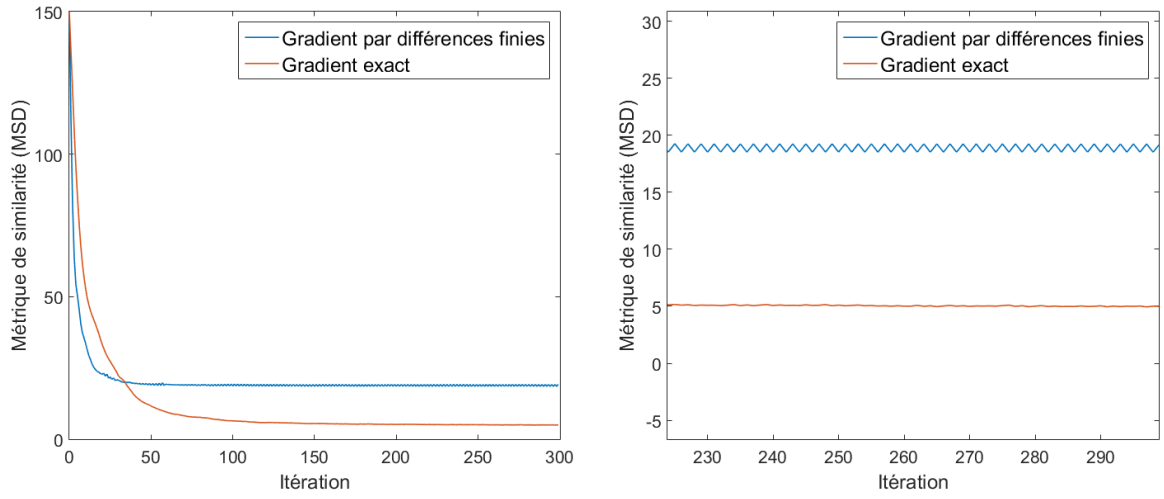
(a) (b) (c) (d) (e) (f)

FIGURE 6.5 – (a) Image fixe, (b) Image mouvante, (c) Image mouvante déformée - Différences finies, (d) Image mouvante déformée - Gradient exact, (e) Différence absolue entre (a) et (c), (f) Différence absolue entre (a) et (d)



(a) (b) (c) (d) (e) (f)

FIGURE 6.6 – (a) Image fixe, (b) Image mouvante, (c) Image mouvante déformée - Différences finies, (d) Image mouvante déformée - Gradient exact, (e) Différence absolue entre (a) et (c), (f) Différence absolue entre (a) et (d)

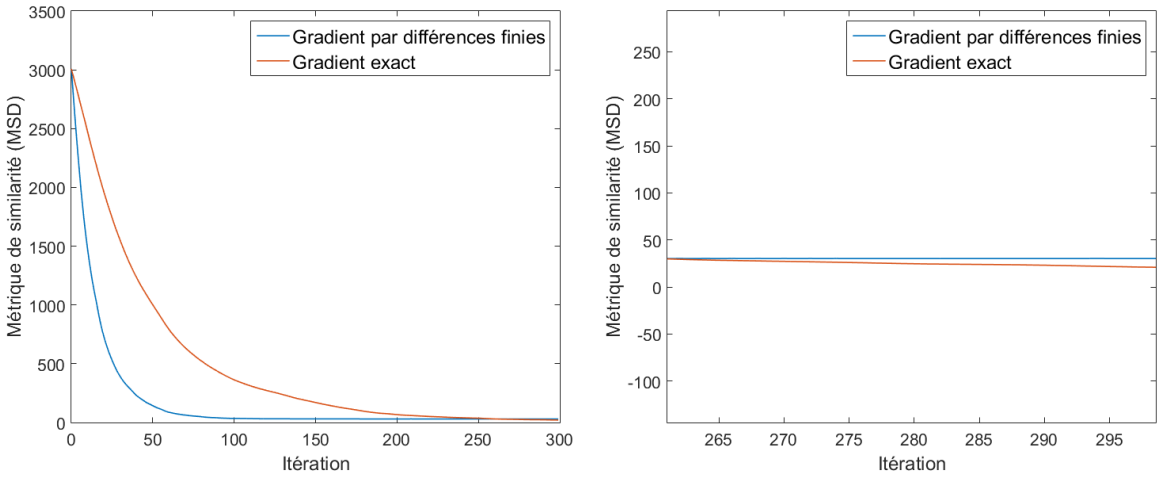


(a) (b)

FIGURE 6.7 – (a) Evolution du critère de similarité MSD pour le premier jeu de données (poumons d'un rat), (b) Zoom sur les dernières itérations

La métrique obtenue par recalage des images représentant une main est elle extrêmement lisse ; le problème de l'instabilité n'apparaît pas. Notons que la valeur de la MSD est très élevée au début du recalage puisque les images sont très différentes, contrairement au cas précédent. Par contre, nous observons à nouveau une amélioration de la qualité du recalage en utilisant un gradient exact, mais c'est léger. La figure 6.8(b) indique cependant que la valeur de la métrique pourrait encore descendre en autorisant un plus grand nombre d'itérations. De plus, le temps d'exécution étant à nouveau double pour la méthode calculant le gradient par différences finies, celle-ci n'a d'intérêt que si on souhaite un temps maximal de recalage très court.

Pour le troisième jeu de données, nous ne nous attardons pas trop sur les résultats puisque nous



(a) (b)

FIGURE 6.8 – (a) Evolution du critère de similarité MSD pour le deuxième jeu de données (main), (b) Zoom sur les dernières itérations

avons déjà vu que ces images ne se prêtent pas à un algorithme tel que les ensembles de niveau. Nous observons cependant à la figure 6.9 des courbes assez similaires à celles obtenues pour les paires d'images précédentes.

Enfin, le dernier jeu de données montre des résultats semblables à ceux obtenus par le recalage du deuxième jeu de données. Les métriques sont stables et la méthode du gradient exact permet d'obtenir une meilleure précision au-delà d'un certain nombre d'itérations. De plus, les temps d'exécution de l'algorithme sont à nouveau du simple au double. Notons cependant que les temps d'exécution sont ici beaucoup plus élevés que pour les autres images testées. Cela est simplement dû au fait que ces images-ci sont de taille 256×256 au lieu de 128×128. Il y a donc quatre fois plus de pixels à traiter à chaque itération, ce qui explique des temps d'exécution si élevés.

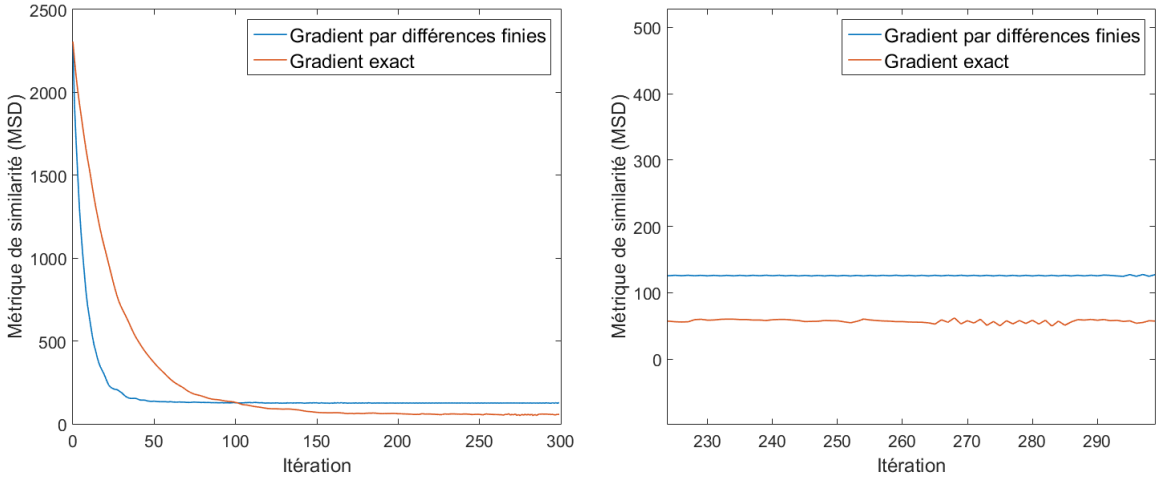


FIGURE 6.9 – (a) Evolution du critère de similarité MSD pour le troisième jeu de données (IRM du cerveau), (b) Zoom sur les dernières itérations

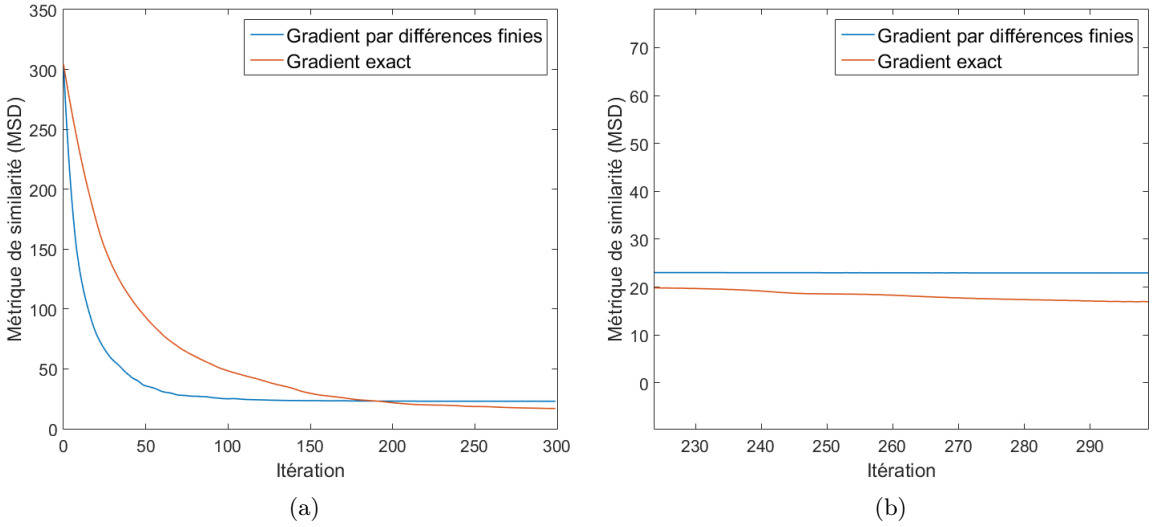


FIGURE 6.10 – (a) Evolution du critère de similarité MSD pour le quatrième jeu de données (cerveau), (b) Zoom sur les dernières itérations

	Différences finies	**Gradient exact**
Paire d'images 1 - Poumons de rat	10742 ms	6027 ms
Paire d'images 2 - Main	10859 ms	5881 ms
Paire d'images 3 - IRM du cerveau	10589 ms	6205 ms
Paire d'images 4 - Cerveau	34988 ms	17361 ms

TABLE 6.1 – Temps d'exécution (en millisecondes) du recalage pour les quatre jeux de données présentés : comparaison des méthodes par gradient approximé et exact

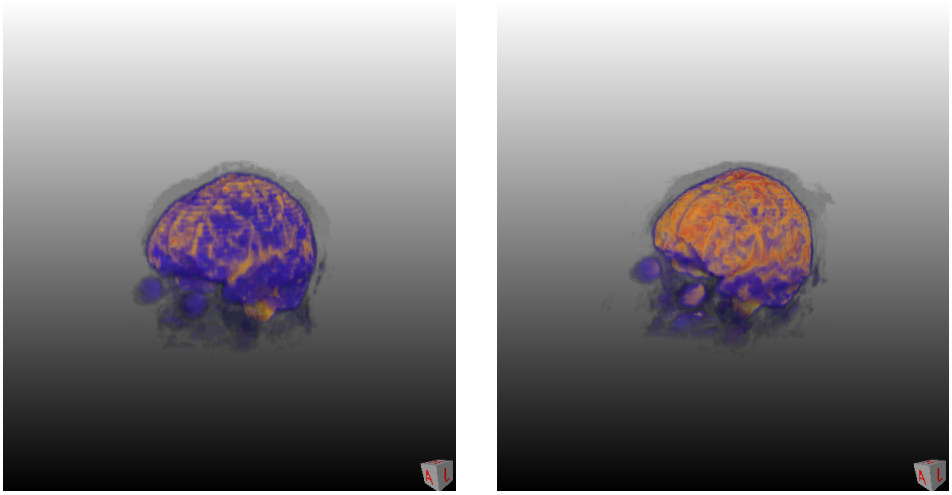
La différence de temps d'exécution entre les deux méthodes est assez simple à expliquer. En effet, nous avons vu au chapitre précédent qu'obtenir le gradient de l'interpolant en un point est extrêmement similaire à obtenir l'intensité de l'image en un point, puisque la dérivée de l'interpolant est preque identique à la formule d'interpolation. Dès lors, une dérivation en un point consiste principalement en la recherche des coefficients d'interpolation et en l'évaluation d'un certain nombre de B-splines, tout comme l'interpolation. Or une dérivation par différences finies demande trois interpolations : une première pour déterminer l'intensité du point d'intérêt et deux autres pour déterminer l'intensité des pixels voisins, permettant ainsi le calcul des différences finies forward et backward. D'autres petits calculs (minimes par rapport au coût de l'interpolation) sont de plus également nécessaires pour obtenir une différence finie *minmod*. Une dérivation exacte en un point, par contre, ne demande que l'équivalent d'une interpolation. Le calcul du gradient approximé est donc à peu près trois fois plus coûteux que celui du gradient exact.

6.3 Etude d'un cas 3D

Cette section a pour but de présenter les résultats obtenus lors du test de l'algorithme des ensembles de niveau sur une paire d'images 3D. La paire d'images en question est composée d'IRM du cerveau 16-bit dont la dimension est $154 \times 154 \times 40$ et nous a été fournie par le CHU de Dinant Godinne. Rappelons qu'une image 3D est une pile d'images 2D, cf. Chapitre 1, Section 1.2.1. Nous pouvons ici considérer notre image soit comme une pile de 154 images 2D frontales, soit comme une pile de 154 images 2D sagittales, soit comme une pile de 40 images transversales. Pour rappel, ces termes avaient été introduits au Chapitre 1, à la Section 1.2.6. Une vue 3D de chacune des images est proposée à la figure 6.11. Celles-ci ont été colorées à l'aide de *MevisLab* pour une meilleure visualisation.

Pour appliquer l'exemple d'ITK à ces images, nous avons dû transformer chacune des deux images 3D, qui consistaient en une image Dicom sous forme de pile, en un seul et même ficher. Nous avons pour cela utilisé le code *DicomSeriesReadImageWrite2* de la librairie ITK et ainsi obtenu deux fichiers 3D. Nous avons également appliqué un filtre sur l'image mouvante afin de lui donner le même espace physique que celui de l'image fixe via le filtre *itkChangeInformationImageFilter*. Enfin, il a fallu changer le type de pixels utilisé pour écrire l'image de sortie, qui était prévu pour des images 8-bit.

Analyser visuellement les résultats d'un recalage d'images 3D est plus difficile que pour des images 2D. En effet, visualiser uniquement le volume comme à la figure 6.11 n'apporte pas assez de détails de l'image et il nous est impossible de visualiser ici chaque tranche de l'image. Nous avons donc ici choisi de comparer quatre tranches transversales de l'image ainsi que, bien sûr, l'évolution de la métrique au cours des itérations et les temps d'exécution respectifs de la méthode



(a) (b)

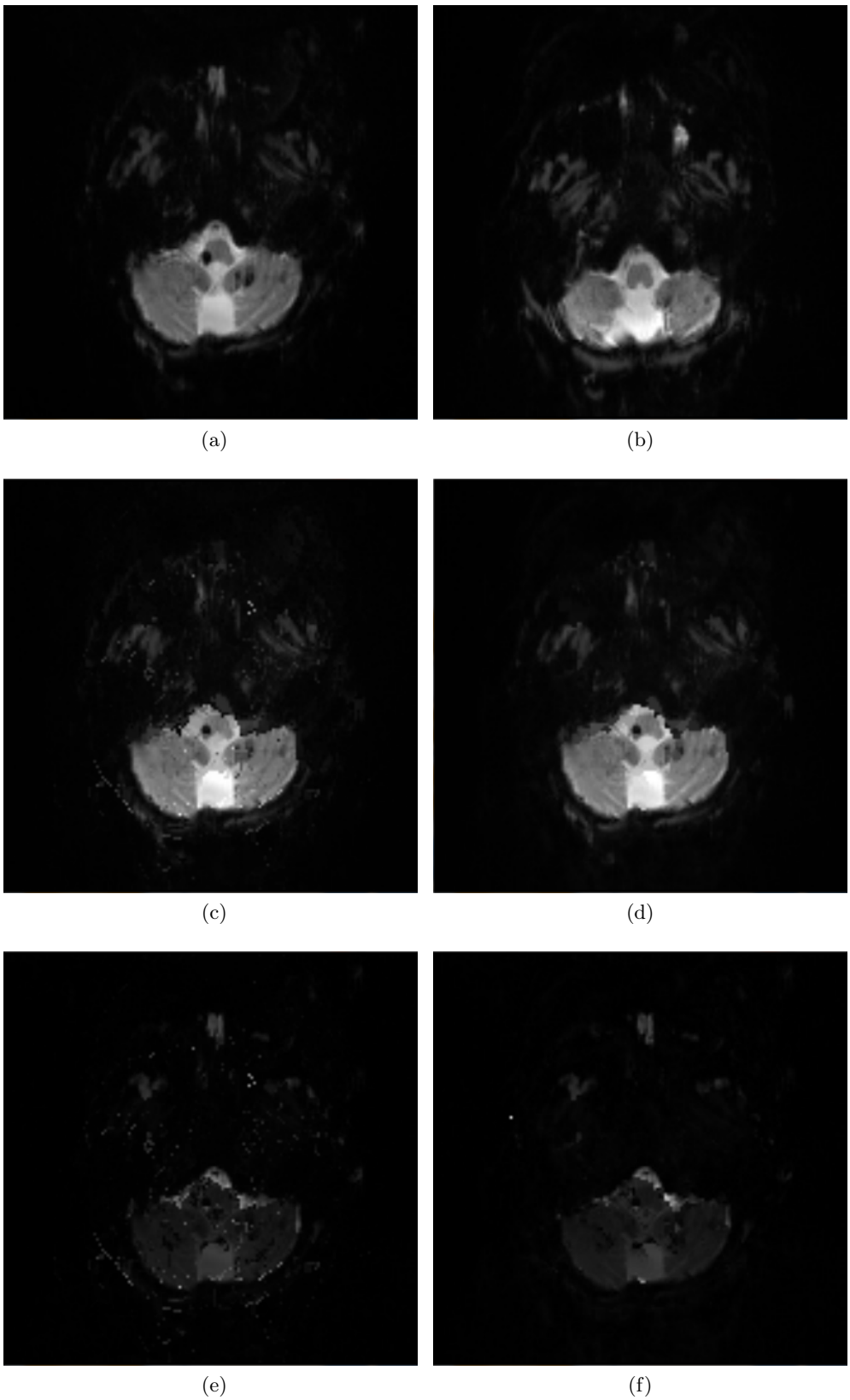
FIGURE 6.11 – (a) Image fixe (b) Image mouvante

utilisant un gradient exact et de celle utilisant un gradient approximé. Les tranches numéro 7, 12, 22 et 31 ont été choisies. Les résultats obtenus pour chacune des méthodes considérées sont visibles aux figures 6.12 à 6.15. La figure 6.16 montre l'évolution des métriques et, enfin, la table 6.2 présente les temps d'exécution pour les deux algorithmes.

Avant de passer aux analyses des différents résultats, nous souhaitons insister sur le fait que nous avons bien réalisé ici un recalage 3D et non quarante recalages 2D. Le champ de déplacement obtenu est donc tridimensionnel et les images recalées présentées dans les figures ci-dessous ne correspondent pas au recalage 2D des tranches initiales montrées.

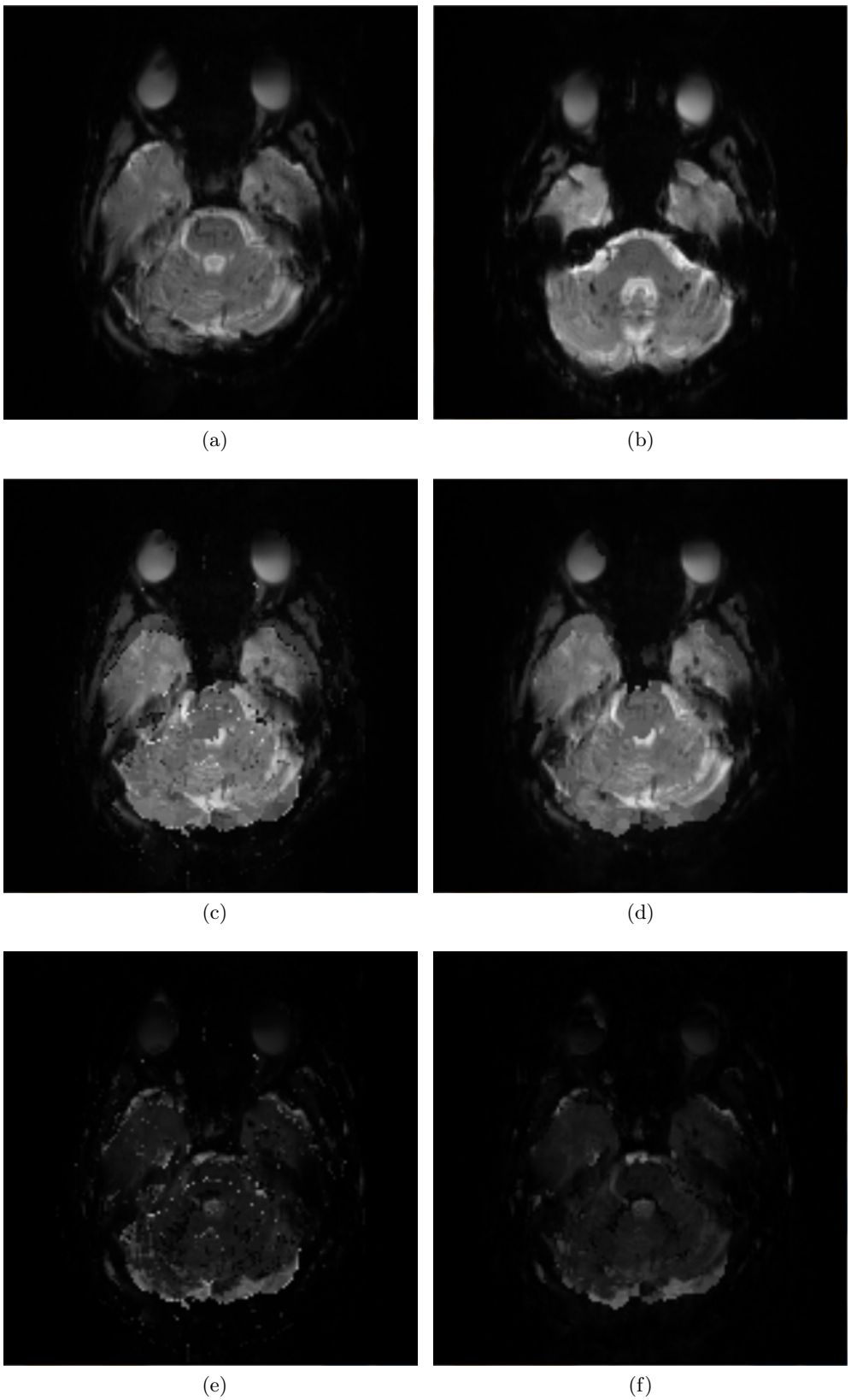
A la figure 6.12, nous tentons d'évaluer la qualité du recalage d'images via l'observation de la tranche 7. Pour les résultats obtenus via l'une ou l'autre méthode, aux figures 6.12(c) et 6.12(d), nous voyons une ressemblance générale avec la tranche 7 de l'image fixe (figure 6.12(a)). L'image est bien dans l'ensemble mais est loin d'être parfaite : plusieurs anomalies peuvent être observés quelle que soit la technique de calcul du gradient. Ainsi, l'image obtenue avec un gradient approximé est plus bruitée et moins lisse ; elle possède de surcroît un peu plus de petits défauts que la tranche 7 de l'image obtenue avec un calcul du gradient exact. Cela apparaît d'ailleurs aussi à travers les images présentant la différence absolue entre images fixes et images recalées, aux figures 6.12(e) et 6.12(f) : celle obtenue avec un gradient exact est plus sombre presque partout.

Pour la tranche 12 des images 3D, des remarques similaires sont à faire. Nous pouvons cependant de plus remarquer la mauvaise qualité du recalage lorsque les déformations nécessaires sont importantes. En effet, les figures 6.13(c) et 6.13(d) présentent toutes deux des contours peu lisses et mal dessinés à certains endroits ; citons par exemple le bord du cerveau dans la partie inférieure de l'image. L'algorithme des ensembles de niveau semble plus à même de réaliser des déformations légères, comme par exemple pour les bords du cerveau observés sur la gauche et la droite de l'image. Précisons cependant que cela semble lié à l'algorithme choisi et non au calcul du gradient puisque cette observation est valable pour la figure 6.13(c) aussi bien que pour la figure 6.13(d). Les différences absolues entre images mouvantes et images recalées, aux figures 6.13(e) et 6.13(f), semblent elles indiquer un recalage quelque peu meilleur pour la méthode utilisant un gradient exact.



(a) (b) (c) (d) (e) (f)

FIGURE 6.12 – (a) Image fixe - tranche 7, (b) Image mouvante - tranche 7, (c) Image mouvante déformée - Différences finies, (d) Image mouvante déformée - Gradient exact, (e) Différence absolue entre (a) et (c), (f) Différence absolue entre (a) et (d)



(a) (b)

(c) (d)

(e) (f)

FIGURE 6.13 – (a) Image fixe - tranche 12, (b) Image mouvante - tranche 12, (c) Image mouvante déformée - Différences finies, (d) Image mouvante déformée - Gradient exact, (e) Différence absolue entre (a) et (c), (f) Différence absolue entre (a) et (d)

La tranche 22 des images à recaler, figures 6.14(a) et 6.14(b), nous permet particulièrement bien de mettre en avant un important problème : le cerveau de l'image mouvante est situé bien plus bas que celui de l'image fixe. A cause de cela, l'algorithme doit faire d'importantes déformations de bas en haut et, comme nous l'avons déjà observé, cela ne se passe pas très bien. Ici, à nouveau, les contours du cerveau dans la partie supérieure et la partie inférieure de l'image sont fortement abîmés et très mal découpés (figures 6.14(c) et 6.14(d)). C'est simplement dû au fait que l'algorithme des niveau est prévu pour faire des corrections locales. Ici, le travail à effectuer est trop important. En revanche, l'intérieur du cerveau a été assez bien déformé et ce spécialement pour l'image obtenue en calculant le gradient de manière exacte qui ne possède pas de bruit. L'autre image est elle à nouveau fort bruitée et peut-être un peu moins bien déformée à certains endroits. C'est néanmoins léger.

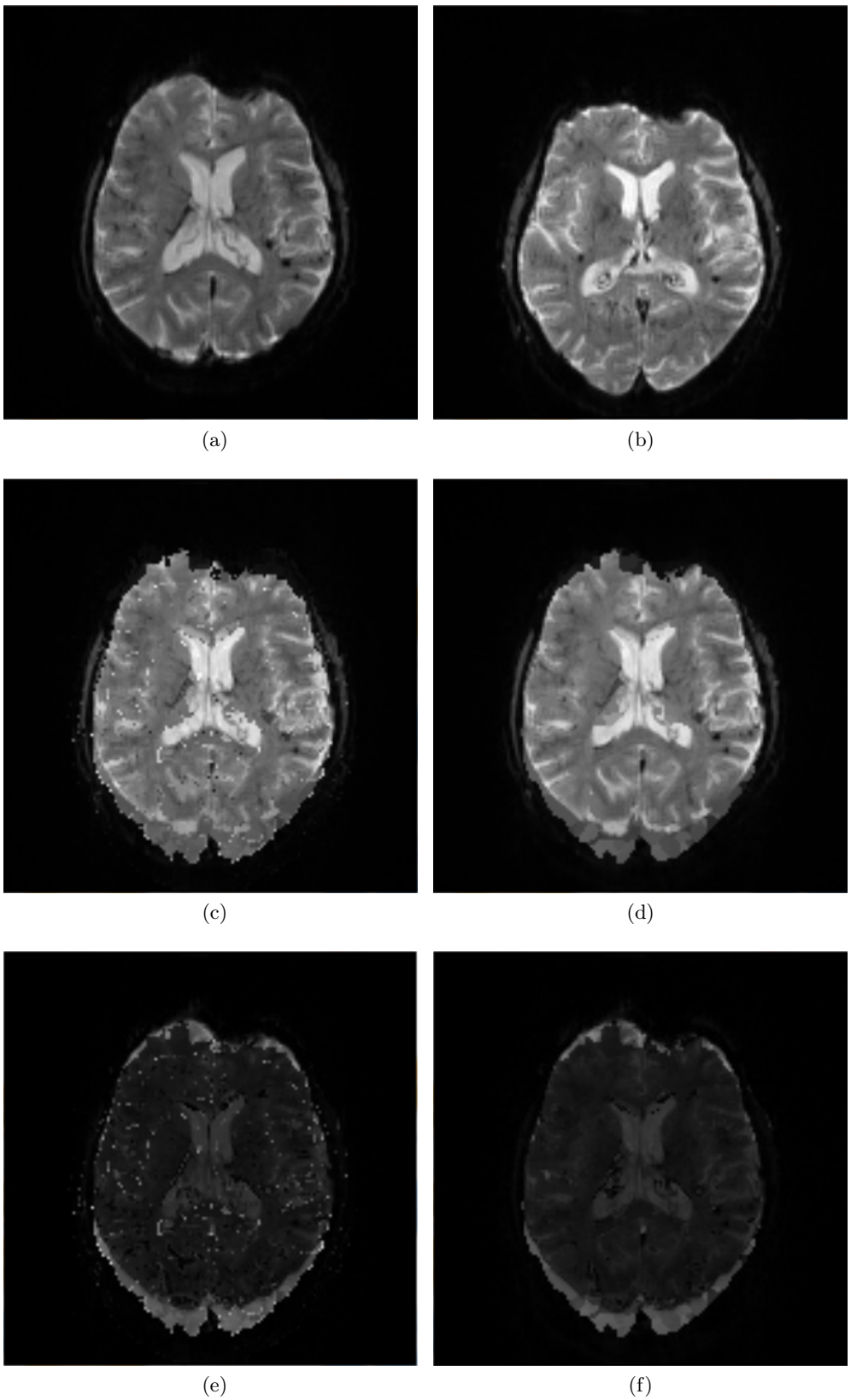
Enfin, la dernière tranche observée ne nous apprend rien de nouveau. Nous constatons toujours les mêmes problèmes dérangeants aux figures 6.15(c) et 6.15(d) : les contours du cerveau sont très mal corrigés en haut et en bas des images et l'image obtenue avec un gradient approximé est bruitée. L'intérieur du cerveau est lui assez bien déformé ; on l'observe notamment aux stries du cerveau.

De manière générale, il nous faut cependant observer que les tranche des images recalées sont plus claires que celles des images fixes. Cela est dû au fait que l'image mouvante est plus lumineuse dans l'ensemble que l'image fixe, comme l'indiquaient d'ailleurs déjà les figures 6.11(a) et 6.11(b). Or l'algorithme des ensembles de niveau suppose une même intensité pour des points correspondants, ce qui explique le petit décalage d'intensité. Notons que cette constatation explique que l'on visualise si bien les différences absolues entre images et que les valeurs de la métrique, comme nous allons le voir, soient si hautes.

Nous pouvons aussi remarquer qu'il pourrait être intéressant de coupler l'algorithme des ensembles de niveau au recalage d'images par transformation globale affine que nous avions évoqué au Chapitre 2, Section 2.3. Cela permettrait ainsi de faire une première correction affine très générale, réduisant les importantes déformations nécessaires à des plus petites. De plus, les recalages de ce type sont assez rapides ; le temps d'exécution général n'en serait probablement que peu accru.

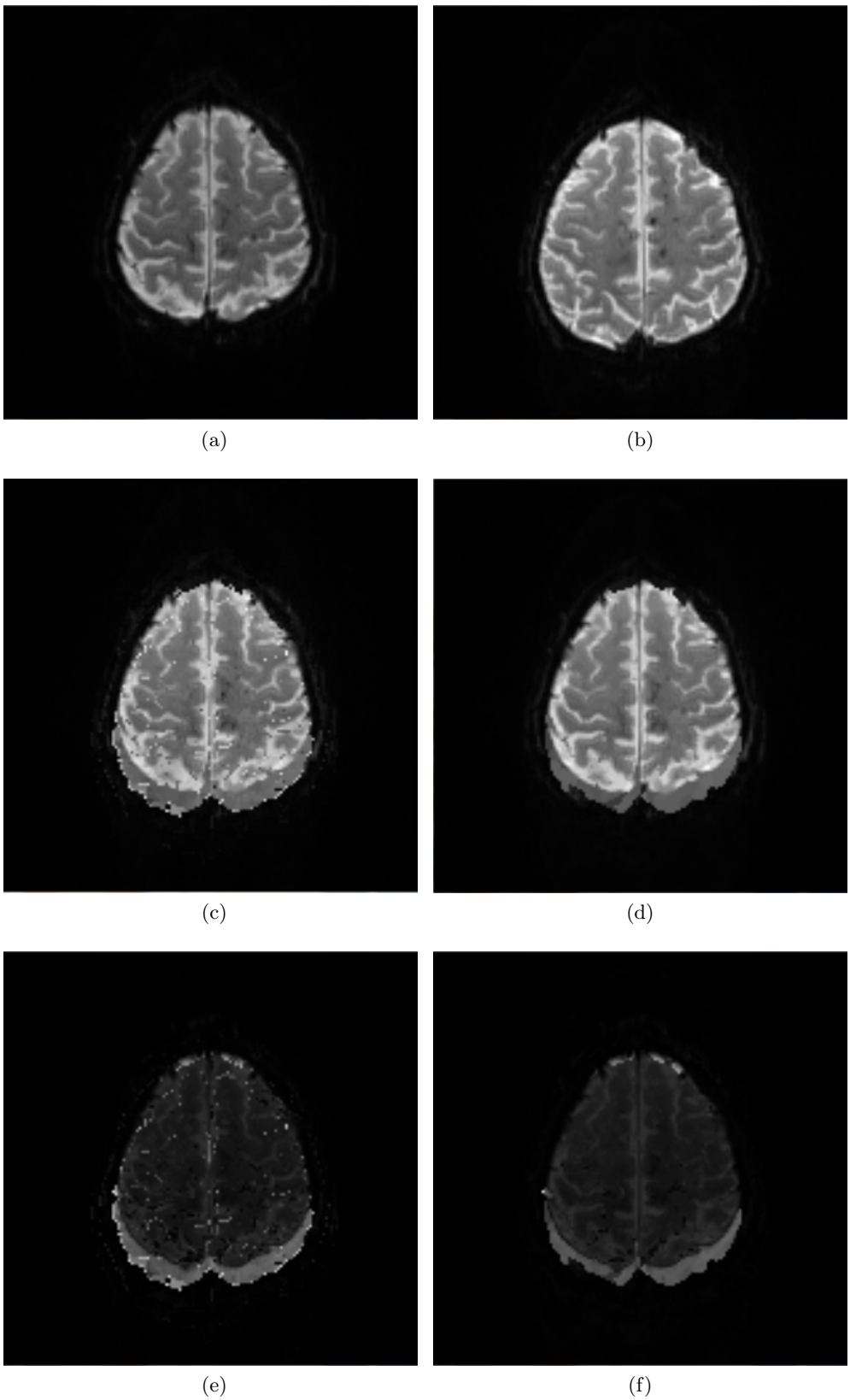
Intéressons-nous à présent à l'évolution de la métrique pour comparer le recalage obtenu en calculant le gradient de manière exacte et celui obtenu avec un gradient approximé par différences finies. L'évolution de la métrique est présentée à la figure 6.16. Nous voyons grâce à la figure zoomée 6.16(b) que la valeur de la métrique, après 300 itérations, est quasiment du simple au double entre les deux méthodes. Le gain obtenu grâce au gradient exact est donc énorme et, de plus, il ne faut qu'une cinquantaine d'itérations à cette méthode pour faire aussi bien que la méthode utilisant des différences finies. C'est cependant probablement dû au fait que le spacing de l'image est assez important : (1.4935, 1.4935, 3.9). Cela engendre donc des différences finies assez loin du gradient exact puisque, pour rappel, le pas utilisé pour leur calcul est le spacing. Dès lors, il pourrait peut-être être intéressant de ne pas utiliser le spacing comme pas pour les différences finies mais bien un pas fixe qui serait un compromis entre précision du gradient obtenu et amplitude du pas temporel permettant de garder la stabilité du schéma de résolution numérique.

Si nous prenons maintenant également les temps d'exécution en compte, nous voyons que la méthode avec gradient exact est ici clairement supérieure à l'autre. En effet, la table 6.2 indique, comme pour les cas 2D étudiés précédemment, un temps plus que doublé pour la méthode calculant le gradient par différences finies. Dans l'absolu, les temps d'exécution ne sont pas très bons puisqu'ils sont respectivement de l'ordre de 16 et 7 minutes.



(a) (b) (c) (d) (e) (f)

FIGURE 6.14 – (a) Image fixe - tranche 22, (b) Image mouvante - tranche 22, (c) Image mouvante déformée - Différences finies, (d) Image mouvante déformée - Gradient exact, (e) Différence absolue entre (a) et (c), (f) Différence absolue entre (a) et (d)



(a) (b)

(c) (d)

(e) (f)

FIGURE 6.15 – (a) Image fixe - tranche 31, (b) Image mouvante - tranche 31, (c) Image mouvante déformée - Différences finies, (d) Image mouvante déformée - Gradient exact, (e) Différence absolue entre (a) et (c), (f) Différence absolue entre (a) et (d)

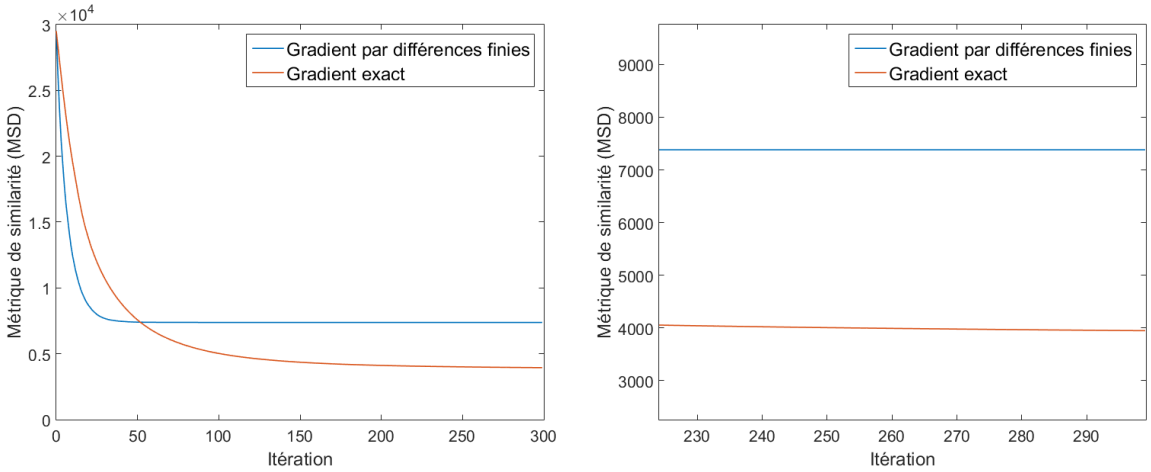


FIGURE 6.16 – (a) Evolution du critère de similarité MSD pour le recalage 3D de deux IRM du cerveau, (b) Zoom sur les dernières itérations

	Différences finies	**Gradient exact**
Paire d'images 3D - IRM du cerveau	1 001 446 ms	469 363 ms

TABLE 6.2 – Temps d'exécution (en millisecondes) du recalage d'images 3D effectué : comparaison des méthodes par gradient approximé et exact

Au vu de l'ensemble des observations faites, nous nous rendons compte que, pour cette image 3D, il est préférable d'utiliser un calcul de gradient exact avec la méthode des ensembles de niveau. Ce n'est cependant qu'une seule image et nous ne pouvons prédire avec certitude comment se comporteraient les deux méthodes sur une autre image 3D.

6.4 Conclusion

Bien que la méthode implémentée avec un gradient exact semble prometteuse, il est difficile de se prononcer sur son utilité réelle. En effet, tout est question de compromis entre temps d'exécution et précision du recalage et les différents tests présentés ici ont été réalisés sur un ordinateur à processeur peu puissant. Il pourrait donc être intéressant de refaire ces tests avec un autre matériel. Dans l'idéal, il pourrait aussi être mieux de tester l'algorithme sur un plus grand nombre d'images avant de tirer des conclusions.

Nous pouvons cependant déjà affirmer pour le cas 2D que, si le processus de recalage d'images doit être rapide, ce qui est souvent le cas, la méthode utilisant un gradient exact est surtout utile pour des images présentant peu de différences avant recalage. Cela est très bien illustré par le premier jeu de données. Par contre, pour une paire d'images très différentes de base, telles que celles représentant une main présentées précédemment, on peut préférer un recalage rapide mais moins précis. Dans ce cas, il faudra plutôt choisir d'approximer le gradient par des différences finies.

En résumé, le choix de la méthode dépend de la situation. Si le médecin est en possession de deux images très différentes, un recalage rapide et manquant un peu de précision pourra sans doute lui suffire. S'il veut par contre recaler deux images déjà très similaires, ce sera alors probablement avec

un but de grande précision et l'utilisation de la méthode avec gradient exact sera alors plus adaptée.

Pour le cas 3D, il serait indispensable de réaliser plus de tests. Nous n'avons ici testé les algorithmes que sur une paire d'images et, même si ici la méthode utilisant un gradient exact semble plus adaptée, les recherches doivent être poussées plus loin avant de tirer des conclusions. Tester les méthodes sur des images provenant d'autres modalités comme par exemple le CT pourrait notamment être utile. Comme déjà mentionné plus haut, nous pourrions aussi éventuellement coupler l'algorithme des ensembles de niveau à un pré-recalage de type affine afin de corriger les différences importantes et puis seulement déformer localement l'image.

Notons finalement que nous avons ici appliqué la méthode utilisant des dérivées exactes à l'algorithme des ensembles de niveau, algorithme qui était de base implémenté dans ITK avec une méthode d'interpolation linéaire afin de limiter le temps de calcul. Il faut néanmoins garder à l'esprit qu'un calcul exact du gradient comme celui-ci serait applicable à n'importe quel algorithme de recalage couplé à une méthode d'interpolation par B-splines. Il pourrait dès lors être intéressant d'étudier cette méthode pour des algorithmes un peu moins coûteux en temps que celui des ensembles de niveau, qui supporteraient peut-être mieux une interpolation un peu coûteuse par splines.

Conclusions et perspectives

Ce mémoire avait pour but l'étude des fonctions B-splines pour l'interpolation dans le recalage d'images médicales afin de fournir une méthode de calcul du gradient de l'image, peu habituelle, employant des dérivées exactes.

Dans cette optique, nous avons commencé par nous immerger dans le monde de l'imagerie médicale. Nous avons ainsi abordé les différentes caractéristiques des images médicales d'un point de vue mathématique et informatique. Nous avons également vu comment ces images sont en pratique acquises dans les hôpitaux, distinguant par la même occasion l'imagerie structurelle de l'imagerie fonctionnelle. Cela nous a mené vers le traitement d'images. En effet, une fois acquises, les images peuvent rarement être utilisées telles quelles. Un traitement préalable est souvent requis. Nous avons ainsi vu qu'il existait plusieurs opérations possibles dans le domaine du traitement d'images et notamment le recalage, qui nous a particulièrement intéressés. Celui-ci consiste en une mise en correspondance de deux images permettant de corriger les différences dues au positionnement, mouvement du patient et autres détails pratiques, tout en gardant l'information des deux images. Cela permet ainsi une interprétation plus précise des images. D'un point de vue mathématique, nous avons appris que le recalage d'images consistait en une optimisation d'un critère de similarité entre images, processus la plupart du temps itératif recherchant une transformation optimale pour recaler l'image dite mouvante sur l'image de référence.

Différentes étapes et caractéristiques du recalage d'images ont ensuite été abordées dont la phase d'interpolation qui nous intéressait particulièrement dans le cadre de ce mémoire. En effet, comme une image est définie de manière discrète, c'est-à-dire comme une grille, il est rare que lors de l'application d'une transformation géométrique sur les pixels, ceux-ci soient envoyés sur des points de la grille. Une interpolation est alors nécessaire pour déterminer l'intensité des points de la grille. Nous avons cependant vu que, par facilité de calcul, l'interpolation est effectuée par transformation inverse afin d'utiliser l'information de points uniformément espacés. Nous avons ensuite étudié différentes méthodes d'interpolation imaginables pour le cas des images. Pour cela, nous nous sommes concentrés sur le cas le plus commun où l'interpolant est exprimé comme une combinaison linéaire de fonctions à choisir. Dans ce mémoire, ces fonctions étaient les B-splines et plus particulièrement les B-splines cubiques. Leur avantage est de respecter des propriétés souhaitables pour l'interpolation d'images telles que le support local, la symétrie, la partition de l'unité ou encore la séparabilité. De plus, l'ordre 3 permet d'obtenir des fonctions deux fois continûment différentiables, ce qui permet de pouvoir calculer le gradient exact de l'image, i.e., de l'interpolant. L'étude de ces fonctions nous a ensuite permis de construire une méthode d'interpolation explicite pour le calcul des coefficients de la combinaisons linéaire de B-splines servant d'interpolant. Cette méthode est basée sur la théorie du signal.

Une autre partie importante de ce mémoire a été le choix et l'étude de l'algorithme de recalage auquel nous allions appliquer l'interpolation par B-splines. La méthode choisie a été celle des ensembles de niveau. Son principe est de faire évoluer les ensembles de niveau de la fonction

image mouvante vers ceux de l'image de référence afin de corriger l'image par des déformations locales. Ce problème peut être exprimé comme une équation aux dérivées partielles (impliquant un calcul du gradient de l'image) de deux manières différentes : soit via une équation d'évolution de l'image, qui s'apparente à l'équation des ensembles de niveau, soit via une équation d'évolution d'une transformation géométrique, qui s'apparente à l'équation d'évolution d'une courbe en termes d'ensembles de niveau. C'est la deuxième possibilité que nous avons ici utilisée. Nous avons vu que cette équation aux dérivées partielles pouvait être résolue de manière discrète via un schéma de résolution numérique par différences finies.

Une fois la théorie nécessaire présentée, nous avons pu nous intéresser à l'implémentation de la méthode étudiée dans une librairie de traitement d'images du C++ nommée ITK. Celle-ci comprenait déjà un code de la méthode des ensembles de niveau, mais avec une interpolation linéaire et un calcul du gradient de l'image via différences finies. Nous avons donc modifié ce code, d'abord pour utiliser l'interpolation par B-splines, ensuite pour utiliser un gradient exact et non approximé par des différences finies. Cette deuxième modification a pu être validée grâce à l'introduction de bornes théoriques sur l'erreur engendrée par l'usage des différences finies.

Le dernier chapitre, enfin, a exposé les différents tests réalisés avec chacune des méthodes de calcul du gradient étudiées. Quatre paires d'images médicales en deux dimensions ont d'abord été utilisées, et il n'a pas été facile de départager les deux méthodes. Comme nous pouvions nous y attendre, nous avons constaté que l'utilisation d'un gradient exact permettait d'obtenir une meilleure optimisation du critère de similarité, mais que le prix à payer était le nombre d'itérations nécessaires pour cela. C'est dû au fait que pour garder un schéma de résolution stable en utilisant des dérivées exactes, nous avons dû diminuer son pas temporel, provoquant ainsi une convergence plus lente en termes du nombre d'itérations. Cependant, nous avons également vu que le temps d'exécution pour un même nombre d'itérations de l'algorithme utilisant des différences finies pour le calcul du gradient était environ le double de celui de la méthode employant un gradient exact. C'est simplement parce que, comme nous l'avons vu, le coût de l'évaluation du gradient exact en un point s'apparente à celui de l'évaluation de l'interpolant en un point. Or les différences finies *minmod* utilisent trois interpolations pour le calcul du gradient en un point. Il faut ainsi être prudent lorsqu'on parle de lenteur de l'algorithme et différencier le temps d'exécution du nombre d'itérations.

Un cas 3D a ensuité été étudié. Nous avons recalé deux IRM du cerveau avec chaque méthode de calcul du gradient. Le résultat obtenu était cette fois assez tranché ; pour ces images, la méthode utilisant un gradient exact était nettement supérieure à l'autre de par la baisse de la valeur de la métrique qu'elle engendrait mais aussi de par la rapidité du processus. Son rapport précision / temps d'exécution était de plus bien meilleur que celui de la méthode approximant le gradient par des différences finies. Les temps d'exécution de la méthode des ensembles de niveau étaient cependant selon nous assez mauvais, mais il faudrait pour bien faire comparer avec d'autres algorithmes. De plus, la méthode des ensembles de niveau donnait tout de même un recalage d'assez basse qualité, sûrement parce qu'il tentait de corriger une importante translation du cerveau par des déformations. Il pourrait donc être intéressant d'essayer de coupler cette méthode à un algorithme de recalage fonctionnant par transformation affine. Cela permettrait probablement de corriger le placement de l'objet de l'image avant de préciser le recalage par des déformation locales de l'image. Par la suite, plus de tests devraient être réalisés. Il est en effet difficile de tirer des conclusions générales sur base d'un seul recalage d'images 3D.

Nous avons donc ici effectué un travail qui devrait pour bien faire être poursuivi. En effet, malgré que la méthode du gradient exact semble posséder des avantages certains, il nous faut

rester prudents en tirant nos conclusions, notamment parce que nous manquons d'informations quant à l'importance exacte du temps d'exécution par rapport à la précision du recalage d'images. Dans un travail futur, fixer un temps maximal d'exécution (réaliste) permettrait par exemple de comparer les deux méthodes plus efficacement. Il faudrait de plus pouvoir tester l'algorithme sur un plus grand nombre d'images et avec un matériel similaire à celui utilisé dans les hôpitaux, afin d'obtenir des informations significatives sur les temps d'exécution de chaque méthode. Ensuite, certaines améliorations du code utilisant un gradient exact pourraient certainement être envisagées pour accélérer le processus. En effet, le code tel qu'il est pour l'instant est prévu pour utiliser des différences finies. L'utilisation d'un gradient exact pourrait changer la donne à certains niveaux de l'algorithme. Cela demanderait cependant une connaissance plus en profondeur des différents composants d'ITK et de la programmation orientée objet. Enfin, il nous faut remarquer que, si nous avons ici uniquement utilisé l'algorithme des ensembles de niveau, une étude comparative des deux méthodes de calcul du gradient pourrait également être réalisée sur d'autres algorithmes, et notamment des algorithmes réputés pour leur rapidité puisque l'utilisation du gradient exact pourrait alors compenser un manque de précision. Ce travail ne se veut donc finalement qu'un premier pas dans la direction de l'étude de la comparaison de ces deux approches pour le calcul du gradient de l'image.

Bibliographie

[1] A. Al Bahou, E. Al Battah et N. Moulin, *L'infographie 3D en médecine*, 2012, https://tpeinfographie3d.wordpress.com/ii-a-acquisition-de-modeles-3d/ii-a-2-passage-de-2d-a-3d/, consulté le 24 avril 2016

[2] I. N. Bankman, *Handbook of Medical Image Processing and Analysis, Second edition*, Elsevier, 2009

[3] C. Barillot, *Fusion de données et imagerie 3D en médecine*, Thèse de doctorat, Université de Rennes 1, 1999

[4] K. Benetos, *La définition/résolution*, 2004, http://tecfaetu.unige.ch/staf/staf-k/benetos/staf13/per1/tache5/resolution.html, consulté le 24 avril 2016

[5] I. Bloch, *Recalage d'images 2d et 3d*, http://perso.telecom-paristech.fr/~bloch/VOIR/recalageVOIR.pdf, consulté le 31 mai 2016

[6] A. Boucher, *Recalage et analyse d'un couple d'images : application aux mammographies*, Thèse de doctorat, Université Paris Descartes, 2013

[7] R. Bourne, *Fundamentals of Digital Imaging in Medicine*, Springer, 2010

[8] D.M. Causon et C.G. Mingham, *Introductory finite difference methods for PDEs*, Ventus Publishing, 2010

[9] F. Cottet, *Traitement des signaux et acquisition des données - Cours et exercices corrigés*, Dunod, 2009

[10] N. Daval, *Statistiques à deux variables*, http://mathematiques.daval.free.fr/IMG/pdf/BTS_Cours_6_Stats2var.pdf, consulté le 20 mai 2016

[11] C. Delahaut, *Médecine nucléaire : optimisation dans le cadre du recalage d'images médicales*, mémoire de fin d'études, Université de Namur, 2014

[12] A. Deniau, C. Dziagwa et C. Neukomm, *Comment l'IRM a-t-elle révolutionné le monde de la médecine ?*, http://tpe-irm.e-monsite.com/pages/interpretation-des-resultats.html, consulté le 27 avril 2016

[13] G. Dougherty, *Digital Image Processing for Medical Applications*, Cambridge University Press, 2009

[14] Site du Zéro, *Les images numériques*, 2013, https://openclassrooms.com/courses/debuter-dans-l-infographie-avec-gimp/les-images-numeriques, consulté le 24 mars 2016

[15] P. Elefante, *4D tomography : walkthrough of my project - part 2*, http://paolaelefante.com/category/medical-imaging/, consulté le 20 juillet 2017

[16] S. Esakkirajan, S. Jayaraman et T. Veerakumar, *Digital Image Processing*, Tata McGraw Hill, 2009

[17] Mevis Fraunhofer et MeVis Medical Solutions AG, *MevisLab*, 2016, http://www.mevislab.de/, consulté le 4 mai 2016

[18] V. Gardeux, *Recalage d'images et méthodes d'optimisation*, 2008, http://www.gardeux-vincent.eu/These/Bibliographie%201.pdf, consulté le 20 mai 2016

[19] J. V. Hajnal, D. L. G. Hill et D. J. Hawkes, *Medical Image Registration*, CRC Press, 2001

[20] Kitware Inc., *CMake*, https://cmake.org/, consulté le 22 juillet 2017

[21] Kitware Inc., *Insight segmentation and registration toolkit*, https://itk.org/, consulté le 22 juillet 2017

[22] Kitware Inc., *ITK registration methods*, https://itk.org/Wiki/images/0/06/Insight-DeformableRegistration-BSplines.ppt, consulté le 26 juillet 2017

[23] H.J. Johnson, M.M. McCormick et L. Ibáñez, *The ITK Software Guide Book 1 : Introduction and Development Guidelines*, Kitware Inc., 2016

[24] M. Kunt, *Traitement numérique des signaux*, Presses polytechniques romandes, 1984

[25] M. Larobina et L. Murino, *Medical image file formats*, 2013, http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3948928/, consulté le 25 avril 2016

[26] T.M. Lehmann, C. Gönner et K. Spitzer, *Survey : Interpolation methods in medical image processing*, *IEEE Transactions in medical imaging*, 1999, 18(11) :p. 1049 – 1075

[27] Commissariat à l'énergie atomique, *L'essentiel sur l'imagerie médicale*, 2016, http://www.cea.fr/comprendre/Pages/sante-sciences-du-vivant/essentiel-sur-imagerie-medicale.aspx, consulté le 2 mai 2016

[28] H. Lombaert, *Level set method : Explanation*, https://profs.etsmtl.ca/hlombaert/levelset/, consulté le 20 juillet 2017

[29] A. J. B. Maintz et M. A. Viergever, *An overview of medical image registration methods*, 1996, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.4417&rep=rep1&type=pdf,consultÃľle15mai2016

[30] A. Manzanera, *Traitement et reconnaissance d'images*, 2016, http://perso.ensta-paristech.fr/~manzaner/Cours/IAD/AM_Couleur.pdf, consulté le 24 avril 2016

[31] A. Marques Ferreira da Silva, *Cours "Processamento de imagem médica"*, 2016, Université d'Aveiro

[32] MathWorks, *Matlab*, https://fr.mathworks.com/products/matlab.html, consulté le 6 août 2017

[33] J. Modersitzki, *Numerical Methods for Image Registration*, Oxford University Press, 2004

[34] J. Modersitzki, *FAIR : Flexible Algorithms for Image Registration*, Siam, 2009

[35] V. Noblet, *Recalage non rigide d'images cérébrales 3D avec contrainte de conservation de la topologie*, Thèse de doctorat, Université Louis Pasteur - Strasbourg I, 2006

[36] J. Nocedal et S.J. Wright, *Numerical Optimization*, Springer-Verlag New York, Inc., 1999

[37] National Institute of Biomedical Imaging et Bioengineering, *Optical imaging*, https://www.nibib.nih.gov/sites/default/files/Optical%20Imaging%20Fact%20Sheet_0.pdf, consulté le 15 mai 2016

[38] National Institutes of Health, *ImageJ*, 2016, https://imagej.nih.gov/ij/, consulté le 24 avril 2016

[39] S. Osher et R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer-Verlag New-York, Inc., 2003

[40] Health Pages, *Anatomical reference planes*, 2016, http://www.healthpages.org/anatomy-function/anatomy-terms/, consulté le 27 avril 2016

[41] W. Puech, *Archivage d'images médicales*, https://www.lirmm.fr/~wpuech/enseignement/Telecom_TIC_SANTE/Compression_DICOM_PUECH.pdf, consulté le 29 avril 2016

[42] M. Rubeaux, *Approximation de l'Information Mutuelle basée sur le développement d'Edgeworth : application au recalage d'images médicales*, Thèse de doctorat, Université de Rennes 1, 2011

[43] L. Schumaker, *Spline functions : basic theory*, John Wiley and Sons, Inc., 1981

[44] J.O. Smith, *Introduction to digital filters with Audio Applications*, W3K Publishing, 2007

[45] M. Unser, *Splines : a perfect fit for signal and image processing*, *IEEE Signal Processing Magazine*, 1999, 16 :p. 22 – 38

[46] M. Unser, A. Aldroubi et M. Eden, *Fast B-spline transforms for continuous image representation and interpolation*, *IEEE Trans. Pattern Analysis and Machine Intelligence*, 1991, 13(3) :p. 277 – 285

[47] M. Unser, A. Aldroubi et M. Eden, *B-spline signal processing : Part 1 - Theory*, *IEEE Trans. Signal Processing*, 1993, 41(2) :p. 821 – 833

[48] M. Unser, A. Aldroubi et M. Eden, *B-spline signal processing : Part 2 - Efficient design and applicaions*, *IEEE Trans. Signal Processing*, 1993, 41(2) :p. 834 – 848

[49] B.C. Vemuri, J. Ye, Y. Chen et C.M. Leonard, *Image registration via level-set motion : Applications to atlas-based segmentation*, *Medical Image Analysis*, 2003, 7 :p. 1 – 20

[50] Wikipédia, *Digital Imaging and Communications in Medicine*, 2015, `https://fr.wikipedia.org/wiki/Digital_imaging_and_communications_in_medicine`, consulté le 29 avril 2016

[51] Wikipédia, *Image matricielle*, 2015, `https://fr.wikipedia.org/wiki/Image\_matricielle`, consulté le 25 mars 2016

[52] Wikipédia, *Imagerie médicale*, 2016, `https://fr.wikipedia.org/wiki/Imagerie_medicale`, consulté le 29 avril 2016

[53] H. Xin, *Modélisation et recalage d'images protéomiques*, Thèse de doctorat, Institut National des Sciences Appliquées de Lyon, 2008

Annexes

Les annexes reprennent les codes principaux pour la méthode de recalage d'images étudiée. Le programme principal est tout d'abord donné. Les classes associées viennent ensuite, mais par souci de concision seuls les fichiers *.hxx* sont donnés, les *.h* ne comprenant quasiment que des déclarations de types, de variables et de fonctions. De plus, nous n'avons repris ici que les fonctions importantes et directement impliquées dans les calculs qui nous intéressent. Le lecteur intéressé peut trouver les fonctions et fichiers manquants dans la librairie ITK.

DeformableRegistration5.cxx

```
/*=========================================================================
*
*  Copyright Insight Software Consortium
*
*  Licensed under the Apache License, Version 2.0 (the "License");
*  you may not use this file except in compliance with the License.
*  You may obtain a copy of the License at
*
*         http://www.apache.org/licenses/LICENSE-2.0.txt
*
*  Unless required by applicable law or agreed to in writing, software
*  distributed under the License is distributed on an "AS IS" BASIS,
*  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
*  See the License for the specific language governing permissions and
*  limitations under the License.
*
*=========================================================================*/

#include <stdio.h>
#include <iostream>
#include <string>
#include <time.h>
#include <fstream>
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkLevelSetMotionRegistrationFilter.h"
#include "itkHistogramMatchingImageFilter.h"
#include "itkChangeInformationImageFilter.h"
#include "itkCastImageFilter.h"
#include "itkWarpImageFilter.h"


// Classe destinée au suivi du processus de recalage
class CommandIterationUpdate : public itk::Command
{
    public:
        typedef  CommandIterationUpdate                    Self;
        typedef  itk::Command                              Superclass;
```

```
        typedef  itk::SmartPointer<CommandIterationUpdate>  Pointer;
41      itkNewMacro(CommandIterationUpdate);

43  protected:
        CommandIterationUpdate() {};
45      typedef itk::Image< float, 3 >            InternalImageType;
        typedef itk::Vector< float, 3 >           VectorPixelType;
47      typedef itk::Image<  VectorPixelType, 3 > DisplacementFieldType;
        typedef itk::LevelSetMotionRegistrationFilter< InternalImageType,
49          InternalImageType, DisplacementFieldType>   RegistrationFilterType;

51  public:
        void Execute(itk::Object *caller, const itk::EventObject & event)
53          ITK_OVERRIDE
        {
55          Execute((const itk::Object *)caller, event);
        }
57      void Execute(const itk::Object * object, const itk::EventObject & event)
            ITK_OVERRIDE
59      {
            const RegistrationFilterType * filter =
61              static_cast< const RegistrationFilterType * >(object);
            if (filter == ITK_NULLPTR)
63          {
                return;
65          }
            if (!(itk::IterationEvent().CheckEvent(&event)))
67          {
                return;
69          }
            // Afficher la valeur de la métrique après chaque itération
71          std::cout << filter->GetMetric() << std::endl;
        }
73 };

75 int main(int argc, char *argv[])
   {
77     /* Vérifier que l'utilisateur a bien rentré assez d'arguments (au moins 3
       arguments : image fixe, image mouvante et nom souhaité de l'image en sortie) */
79     if (argc < 4)
       {
81         std::cerr << "Missing Parameters " << std::endl;
           std::cerr << "Usage: " << argv[0];
83         std::cerr << " fixedImageFile movingImageFile ";
           std::cerr << " outputImageFile " << std::endl;
85         std::cerr << " [outputDisplacementFieldFile] " << std::endl;
           return EXIT_FAILURE;
87     }

89     // Dimension des images à recaler et définition d'un type pour leurs pixels
       const unsigned int Dimension = 2;
91     typedef short int PixelType;

93     // Définition de types pour les images données en entrée
       typedef itk::Image< PixelType, Dimension >  FixedImageType;
95     typedef itk::Image< PixelType, Dimension >  MovingImageType;

97     // Objets permettant de lire l'image fixe et l'image mouvante
       typedef itk::ImageFileReader< FixedImageType  > FixedImageReaderType;
99     typedef itk::ImageFileReader< MovingImageType > MovingImageReaderType;
       FixedImageReaderType::Pointer fixedImageReader = FixedImageReaderType::New();
```

```
101     MovingImageReaderType::Pointer movingImageReader = MovingImageReaderType::New()
        ;
        fixedImageReader->SetFileName(argv[1]);
103     movingImageReader->SetFileName(argv[2]);

105
        /* Modification éventuelle de l'espace physique de l'image mouvante
107     pour qu'il concorde avec celui de l'image fixe */
        typedef itk::ChangeInformationImageFilter<MovingImageType> FilterType;
109     FilterType::Pointer changeFilter = FilterType::New();
        changeFilter->SetInput(movingImageReader->GetOutput());
111     changeFilter->UseReferenceImageOn();
        changeFilter->SetReferenceImage(fixedImageReader->GetOutput());
113     changeFilter->ChangeOriginOn();
        changeFilter->ChangeSpacingOn();
115     changeFilter->ChangeDirectionOn();
        changeFilter->Update();
117
        // Caster les images pour qu'elles aient des pixels de type float (utile pour
        les calculs)
119     typedef float                                          InternalPixelType;
        typedef itk::Image< InternalPixelType, Dimension >  InternalImageType;
121     typedef itk::CastImageFilter< FixedImageType, InternalImageType >
            FixedImageCasterType;
123     typedef itk::CastImageFilter< MovingImageType, InternalImageType >
            MovingImageCasterType;
125     FixedImageCasterType::Pointer fixedImageCaster = FixedImageCasterType::New();
        MovingImageCasterType::Pointer movingImageCaster = MovingImageCasterType::New()
        ;
127     fixedImageCaster->SetInput(fixedImageReader->GetOutput());
        movingImageCaster->SetInput(changeFilter->GetOutput());
129
        // Pré-recalage par matching des histogrammes
131     typedef itk::HistogramMatchingImageFilter< InternalImageType, InternalImageType
        >
            MatchingFilterType;
133     MatchingFilterType::Pointer matcher = MatchingFilterType::New();
        matcher->SetInput(movingImageCaster->GetOutput());
135     matcher->SetReferenceImage(fixedImageCaster->GetOutput());
        matcher->SetNumberOfHistogramLevels(1024);
137     matcher->SetNumberOfMatchPoints(7);
        matcher->ThresholdAtMeanIntensityOn();
139
        // Création d'un champ de déplacement et du filtre de recalage
141     typedef itk::Vector< float, Dimension >                VectorPixelType;
        typedef itk::Image<  VectorPixelType, Dimension >      DisplacementFieldType;
143     typedef itk::LevelSetMotionRegistrationFilter< InternalImageType,
            InternalImageType, DisplacementFieldType> RegistrationFilterType;
145     RegistrationFilterType::Pointer filter = RegistrationFilterType::New();

147     // Lier l'objet suivant le processus de recalage au filtre de recalage d'images
        CommandIterationUpdate::Pointer observer = CommandIterationUpdate::New();
149     filter->AddObserver(itk::IterationEvent(), observer);

151     // Choisir les paramètres du filtre de recalage et enclancher le filtre
        filter->SetFixedImage(fixedImageCaster->GetOutput());
153     filter->SetMovingImage(matcher->GetOutput());
        filter->SetNumberOfIterations(300);
155     filter->SetGradientSmoothingStandardDeviations(1.0);
        filter->Update();
157
```

```
    // Création d'un filtre permettant de déformer l'image mouvante et ajustement
    des paramètres associés
159 typedef itk::WarpImageFilter< MovingImageType, MovingImageType,
        DisplacementFieldType > WarperType;
161 typedef itk::BSplineInterpolateImageFunction< MovingImageType, double >
        InterpolatorType;
163 WarperType::Pointer warper = WarperType::New();
    InterpolatorType::Pointer interpolator = InterpolatorType::New();
165 FixedImageType::Pointer fixedImage = fixedImageReader->GetOutput();
    warper->SetInput(changeFilter->GetOutput());
167 warper->SetInterpolator(interpolator);
    warper->SetOutputSpacing(fixedImage->GetSpacing());
169 warper->SetOutputOrigin(fixedImage->GetOrigin());
    warper->SetOutputDirection(fixedImage->GetDirection());
171 warper->SetDisplacementField(filter->GetOutput());

173 // Cast l'image mouvante en un type de sortie et écrire l'image déformée dans
    un fichier
    typedef short int OutputPixelType;
175 typedef itk::Image< OutputPixelType, Dimension > OutputImageType;
    typedef itk::CastImageFilter< MovingImageType, OutputImageType >
    CastFilterType;
177 typedef itk::ImageFileWriter< OutputImageType > WriterType;
    WriterType::Pointer writer = WriterType::New();
179 CastFilterType::Pointer caster = CastFilterType::New();
    writer->SetFileName(argv[3]);
181 caster->SetInput(warper->GetOutput());
    writer->SetInput(caster->GetOutput());
183 writer->Update();

185 // Sauvegarde éventuelle du champ de déformation final obtenu par le filtre de
    recalage
    // Uniquement si l'utilisateur a fourni un quatrième argument
187 if (argc > 4)
    {
189     typedef itk::ImageFileWriter< DisplacementFieldType > FieldWriterType;
        FieldWriterType::Pointer fieldWriter = FieldWriterType::New();
191     fieldWriter->SetFileName(argv[4]);
        fieldWriter->SetInput(filter->GetOutput();
193     fieldWriter->Update();
    }
195 return EXIT_SUCCESS;
}
```

itkLevelSetMotionRegistrationFunction.hxx

```
/*=========================================================================
2 *
  *  Copyright Insight Software Consortium
4 *
  *  Licensed under the Apache License, Version 2.0 (the "License");
6 *  you may not use this file except in compliance with the License.
  *  You may obtain a copy of the License at
8 *
  *         http://www.apache.org/licenses/LICENSE-2.0.txt
10 *
  *  Unless required by applicable law or agreed to in writing, software
12 *  distributed under the License is distributed on an "AS IS" BASIS,
  *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 *  See the License for the specific language governing permissions and
  *  limitations under the License.
16 *
```

```
*=========================================================================*/
#ifndef itkLevelSetMotionRegistrationFunction_hxx
#define itkLevelSetMotionRegistrationFunction_hxx
#include "itkLevelSetMotionRegistrationFunction.h"
#include "itkMacro.h"
#include "itkMath.h"
#include <iostream>
#include <iomanip>
#include <cmath>
#include <limits>
#include <typeinfo>
#include <thread>

namespace itk
{

    // Constructeur
    template< typename TFixedImage, typename TMovingImage, typename
    TDisplacementField > LevelSetMotionRegistrationFunction< TFixedImage,
    TMovingImage, TDisplacementField >::LevelSetMotionRegistrationFunction()
    {
        RadiusType    r;
        unsigned int j;

        for (j = 0; j < ImageDimension; j++)
        {
            r[j] = 0;
        }
        this->SetRadius(r);

        // Différents paramètres
        m_Alpha = 0.1;
        m_GradientMagnitudeThreshold = 1e-9;
        m_IntensityDifferenceThreshold = 0.001;
        m_GradientSmoothingStandardDeviations = 1.0;
        this->SetMovingImage(ITK_NULLPTR);
        this->SetFixedImage(ITK_NULLPTR);

        // Interpolateur pour l'image mouvante
        typename DefaultInterpolatorType::Pointer interp =
            DefaultInterpolatorType::New();
        m_MovingImageInterpolator = static_cast< InterpolatorType * >
            (interp.GetPointer());

        // Informations liées à la métrique
        m_Metric = NumericTraits< double >::max();
        m_SumOfSquaredDifference = 0.0;
        m_NumberOfPixelsProcessed = 0L;
        m_RMSChange = NumericTraits< double >::max();
        m_SumOfSquaredChange = 0.0;
        m_UseImageSpacing = true;

        // Informations liées au filtre gaussien de lissage
        m_MovingImageSmoothingFilter = MovingImageSmoothingFilterType::New();
        m_MovingImageSmoothingFilter
            ->SetSigma(m_GradientSmoothingStandardDeviations);
        m_MovingImageSmoothingFilter->SetNormalizeAcrossScale(false);

        // Interpolateur pour l'image lissée
        m_SmoothMovingImageInterpolator = static_cast< InterpolatorType * >
            (DefaultInterpolatorType::New().GetPointer());
    }
```

```
80      /* Initialisation d'une itération du schéma de résolution numérique
        de l'EDP d'évolution */
82      template< typename TFixedImage, typename TMovingImage, typename
        TDisplacementField > void LevelSetMotionRegistrationFunction< TFixedImage,
84      TMovingImage, TDisplacementField >::InitializeIteration()
        {
86          /* Vérifier si l'image mouvante, l'image fixe et l'interpolateur de l'image
            mouvante ont bien été définis */
88          if (!this->GetMovingImage() || !this->GetFixedImage() ||
            !m_MovingImageInterpolator)
90          {
                itkExceptionMacro
92                  (<< "MovingImage, FixedImage and/or Interpolator not set");
            }
94
            // Créer une version lissée de l'image mouvante
96          m_MovingImageSmoothingFilter->SetInput(this->GetMovingImage());
            m_MovingImageSmoothingFilter
98              ->SetSigma(m_GradientSmoothingStandardDeviations);
            m_MovingImageSmoothingFilter->Update();
100
            // Interpolateur pour l'image mouvante lissée
102         m_SmoothMovingImageInterpolator
                ->SetInputImage(m_MovingImageSmoothingFilter->GetOutput());
104
            // Interpolateur pour l'image mouvante non lissée
106         m_MovingImageInterpolator->SetInputImage(this->GetMovingImage());

108         // Initialisations liées à la métrique
            m_SumOfSquaredDifference = 0.0;
110         m_NumberOfPixelsProcessed = 0L;
            m_SumOfSquaredChange = 0.0;
112     }

114     // Fonction principale de l'algorithme : mise à jour du champ de déplacement
        template< typename TFixedImage, typename TMovingImage, typename
116     TDisplacementField > typename LevelSetMotionRegistrationFunction< TFixedImage,
        TMovingImage, TDisplacementField >::PixelType
118     LevelSetMotionRegistrationFunction< TFixedImage, TMovingImage,
        TDisplacementField >::ComputeUpdate(const NeighborhoodType & it, void *gd,
120     const FloatOffsetType & itkNotUsed(offset))
        {
122         // Informations sur le voisinage de pixels considéré
            const IndexType index = it.GetIndex();
124         const double fixedValue = (double)this->GetFixedImage()->GetPixel(index);
            PointType mappedPoint;
126         this->GetFixedImage()->TransformIndexToPhysicalPoint(index, mappedPoint);

128         /* Mise à jour du pixel avec le déplacement calculé à l'itération
            précédente */
130         for (unsigned int j = 0; j < ImageDimension; j++)
            {
132             mappedPoint[j] += it.GetCenterPixel()[j];
            }
134
            // Evaluation de l'interpolant au point d'intérêt
136         PixelType update;
            double    movingValue;
138         if (m_MovingImageInterpolator->IsInsideBuffer(mappedPoint))
            {
140             movingValue = m_MovingImageInterpolator->Evaluate(mappedPoint);
```

```
        }
142     else
        {
144         update.Fill(0.0);
            return update;
146     }

148     // Spacing de l'image mouvante
        MovingSpacingType mSpacing = this->GetMovingImage()->GetSpacing();
150     if (!this->m_UseImageSpacing)
        {
152         mSpacing.Fill(1.0);
        }
154
        /* Calcul du gradient (exact si flagGradientExact=1,
156     approximé par différences finies si flagGradientExact=0) */
        int flagGradientExact(0);
158     PointType     mPoint(mappedPoint);
        CovariantVectorType gradient;
160     double gradientMagnitude(0.0);
        if (flagGradientExact == 1)
162     {
            // Calcul du gradient exact
164         if (m_SmoothMovingImageInterpolator->IsInsideBuffer(mPoint))
            {
166             gradient = m_SmoothMovingImageInterpolator
                    ->EvaluateDerivative(mPoint);
168         }
            else
170         {
                for (unsigned int j = 0; j < ImageDimension; j++)
172             {
                    gradient[j] = 0;
174             }
            }
176         // Norme du gradient
            for (unsigned int j = 0; j < ImageDimension; j++)
178         {
                gradientMagnitude += itk::Math::sqr(gradient[j]);
180         }
            gradientMagnitude = std::sqrt(gradientMagnitude);
182     }
        else
184     {
            // Calcul du gradient par différences finies minmod et de sa norme
186         const double centralValue = m_SmoothMovingImageInterpolator
                ->Evaluate(mPoint);
188         double       forwardDifferences[ImageDimension];
            double       backwardDifferences[ImageDimension];
190         for (unsigned int j = 0; j < ImageDimension; j++)
            {
192             mPoint[j] += mSpacing[j];
                if (m_SmoothMovingImageInterpolator->IsInsideBuffer(mPoint))
194             {
                    forwardDifferences[j] = m_SmoothMovingImageInterpolator
196                     ->Evaluate(mPoint) - centralValue;
                    forwardDifferences[j] /= mSpacing[j];
198             }
                else
200             {
                    forwardDifferences[j] = 0.0;
202             }
```

```
204                 mPoint[j] -= (2.0 * mSpacing[j]);
                    if (m_SmoothMovingImageInterpolator->IsInsideBuffer(mPoint))
206                 {
                        backwardDifferences[j] = centralValue
208                         - m_SmoothMovingImageInterpolator->Evaluate(mPoint);
                        backwardDifferences[j] /= mSpacing[j];
210                 }
                    else
212                 {
                        backwardDifferences[j] = 0.0;
214                 }
                    mPoint[j] += mSpacing[j];
216             }

218             for (unsigned int j = 0; j < ImageDimension; j++)
                {
220                 if (forwardDifferences[j] * backwardDifferences[j] > 0.0)
                    {
222                     const double bvalue = itk::Math::abs(backwardDifferences[j]);
                        double       gvalue = itk::Math::abs(forwardDifferences[j]);
224                     if (gvalue > bvalue)
                        {
226                         gvalue = bvalue;
                        }
228                     gradient[j] = gvalue * itk::Math::sgn(forwardDifferences[j]);
                    }
230                 else
                    {
232                     gradient[j] = 0.0;
                    }
234                 gradientMagnitude += itk::Math::sqr(gradient[j]);
                }
236             gradientMagnitude = std::sqrt(gradientMagnitude);

238         }

240         /* Calcul de la mise à jour du déplacement au pixel d'intérêt et des
            informations liées à la métrique */
242         const double speedValue = fixedValue - movingValue;
            GlobalDataStruct *globalData = (GlobalDataStruct *)gd;
244         // Pour la métrique
            if (globalData)
246         {
                globalData->m_SumOfSquaredDifference += itk::Math::sqr(speedValue);
248             globalData->m_NumberOfPixelsProcessed += 1;
            }
250         // Cas d'un gradient presque nul
            if (itk::Math::abs(speedValue) < m_IntensityDifferenceThreshold
252             || gradientMagnitude < m_GradientMagnitudeThreshold)
            {
254             update.Fill(0.0);
                return update;
256         }
            /* Calculs liés à la métrique et au pas temporel Delta t du schéma de
258         résolution de l'EDP */
            double L1norm = 0.0;
260         for (unsigned int j = 0; j < ImageDimension; j++)
            {
262             update[j] = speedValue * gradient[j] / (gradientMagnitude + m_Alpha);
                if (globalData)
264             {
```

```
                globalData->m_SumOfSquaredChange += itk::Math::sqr(update[j]);
266             L1norm += (itk::Math::abs(update[j]) / mSpacing[j]);
            }
268     }

270     /* Garder la norme L1 calculée en ce pixel si elle est plus grande que
        celles des pixels précédemment traités (pour le calcul du pas temporel) */
272     if (globalData && (L1norm > globalData->m_MaxL1Norm))
        {
274         globalData->m_MaxL1Norm = L1norm;
        }
276
        // La fonction retourne la mise à jour du déplacement au pixel d'intérêt
278     return update;
    }
280
    // Calcul du pas temporel Delta t pour la résolution de l'EDP
282 template< typename TFixedImage, typename TMovingImage, typename
    TDisplacementField > typename LevelSetMotionRegistrationFunction<
284 TFixedImage, TMovingImage, TDisplacementField >::TimeStepType
    LevelSetMotionRegistrationFunction< TFixedImage, TMovingImage,
286 TDisplacementField >::ComputeGlobalTimeStep(void *GlobalData) const
    {
288     TimeStepType dt = 1.0;
        /* Delta = 1 si on utilise des différences finies pour le gradient,
290     0.3 sinon */
        double delta = 1.0;
292
        GlobalDataStruct *d = (GlobalDataStruct *)GlobalData;
294
        // Pas temporel Delta t
296     if (d->m_MaxL1Norm > 0.0)
        {
298         dt = 1.0 / (d->m_MaxL1Norm / delta);
        }
300
        return dt;
302 }

304 /* Mise à jour de la métrique et variables liées, et libération de la mémoire
    pour la structure de données liée à la métrique */
306 template< typename TFixedImage, typename TMovingImage, typename
    TDisplacementField > void LevelSetMotionRegistrationFunction< TFixedImage,
308 TMovingImage, TDisplacementField >::ReleaseGlobalDataPointer(void *gd) const
    {
310     GlobalDataStruct *globalData = (GlobalDataStruct *)gd;

312     m_MetricCalculationLock.Lock();
        m_SumOfSquaredDifference += globalData->m_SumOfSquaredDifference;
314     m_NumberOfPixelsProcessed += globalData->m_NumberOfPixelsProcessed;
        m_SumOfSquaredChange += globalData->m_SumOfSquaredChange;
316     if (m_NumberOfPixelsProcessed)
        {
318         m_Metric = m_SumOfSquaredDifference
                / static_cast< double >(m_NumberOfPixelsProcessed);
320         m_RMSChange = std::sqrt(m_SumOfSquaredChange
                / static_cast< double >(m_NumberOfPixelsProcessed));
322     }
        m_MetricCalculationLock.Unlock();
324     delete globalData;
    }
326 }
```

```
#endif
```

itkBSplineDecompositionImageFilter.hxx

```
/*=========================================================================
 *
 *  Copyright Insight Software Consortium
 *
 *  Licensed under the Apache License, Version 2.0 (the "License");
 *  you may not use this file except in compliance with the License.
 *  You may obtain a copy of the License at
 *
 *         http://www.apache.org/licenses/LICENSE-2.0.txt
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS,
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 *  See the License for the specific language governing permissions and
 *  limitations under the License.
 *
 *=========================================================================*/
/*=========================================================================
 *
 *  Portions of this file are subject to the VTK Toolkit Version 3 copyright.
 *
 *  Copyright (c) Ken Martin, Will Schroeder, Bill Lorensen
 *
 *  For complete copyright, license and disclaimer of warranty information
 *  please refer to the NOTICE file at the top of the ITK source tree.
 *
 *=========================================================================*/
#ifndef itkBSplineDecompositionImageFilter_hxx
#define itkBSplineDecompositionImageFilter_hxx
#include "itkBSplineDecompositionImageFilter.h"
#include "itkImageRegionConstIteratorWithIndex.h"
#include "itkImageRegionIterator.h"
#include "itkProgressReporter.h"
#include "itkVector.h"

namespace itk
{

    // Constructeur
    template< typename TInputImage, typename TOutputImage >
    BSplineDecompositionImageFilter< TInputImage, TOutputImage >
    ::BSplineDecompositionImageFilter()
    {
      m_SplineOrder = 0;
      int SplineOrder = 3;
      m_Tolerance = 1e-10;
      m_IteratorDirection = 0;
      this->SetSplineOrder(SplineOrder);
    }

    // Calcul des coefficients d'une interpolation 1D
    template< typename TInputImage, typename TOutputImage >
    bool BSplineDecompositionImageFilter< TInputImage,
    TOutputImage >::DataToCoefficients1D()
    {

        /* Calcul de la constante multiplicatrice permettant d'obtenir c(k)
        dans le théorème 3.11 (6 pour l'ordre 3) et application */
```

```
59          double c0 = 1.0;
            if ( m_DataLength[m_IteratorDirection] == 1 )
61          {
                return false;
63          }
            for ( int k = 0; k < m_NumberOfPoles; k++ )
65          {
                c0 = c0 * ( 1.0 - m_SplinePoles[k] ) * ( 1.0 - 1.0 / m_SplinePoles[k] )
    ;
67          }
            for ( unsigned int n = 0; n < m_DataLength[m_IteratorDirection]; n++ )
69          {
                m_Scratch[n] *= c0;
71          }

73          // Calcul des coefficients
            for ( int k = 0; k < m_NumberOfPoles; k++ )
75          {
                // Initialiasation du filtre causal (conditions frontières)
77              this->SetInitialCausalCoefficient(m_SplinePoles[k]);
                // Récursion causale
79              for ( unsigned int n = 1; n < m_DataLength[m_IteratorDirection]; n++ )
                {
81                  m_Scratch[n] += m_SplinePoles[k] * m_Scratch[n - 1];
                }
83              // Initialisation du filtre anticausal (conditions frontières)
                this->SetInitialAntiCausalCoefficient(m_SplinePoles[k]);
85              // Récursion anticausale
                for ( int n = m_DataLength[m_IteratorDirection] - 2; 0 <= n; n-- )
87              {
                    m_Scratch[n] =
89                      m_SplinePoles[k] * ( m_Scratch[n + 1] - m_Scratch[n] );
                }
91          }
            return true;
93      }

95      // Fixer l'ordre des B-splines à utiliser
        template< typename TInputImage, typename TOutputImage >
97      void BSplineDecompositionImageFilter< TInputImage, TOutputImage >
        ::SetSplineOrder(unsigned int SplineOrder)
99      {
          if ( SplineOrder == m_SplineOrder )
101         {
            return;
103         }
          m_SplineOrder = SplineOrder;
105       this->SetPoles();
          this->Modified();
107     }

109     // Fixer les poles en fonction de l'ordre des splines choisi
        template< typename TInputImage, typename TOutputImage >
111     void BSplineDecompositionImageFilter< TInputImage,
        TOutputImage >::SetPoles()
113     {
            switch ( m_SplineOrder )
115         {
                case 3:
117                 m_NumberOfPoles = 1;
                    m_SplinePoles[0] = std::sqrt(3.0) - 2.0;
119                 break;
```

```
            case 0:
121             m_NumberOfPoles = 0;
                break;
123         case 1:
                m_NumberOfPoles = 0;
125             break;
            case 2:
127             m_NumberOfPoles = 1;
                m_SplinePoles[0] = std::sqrt(8.0) - 3.0;
129             break;
            case 4:
131             m_NumberOfPoles = 2;
                m_SplinePoles[0] = std::sqrt( 664.0 - std::sqrt(438976.0) )
133                 + std::sqrt(304.0) - 19.0;
                m_SplinePoles[1] = std::sqrt( 664.0 + std::sqrt(438976.0) )
135                 - std::sqrt(304.0) - 19.0;
                break;
137         case 5:
                m_NumberOfPoles = 2;
139             m_SplinePoles[0] = std::sqrt(135.0 / 2.0 - std::sqrt(17745.0 / 4.0)
    )
                    + std::sqrt(105.0 / 4.0) - 13.0 / 2.0;
141             m_SplinePoles[1] = std::sqrt(135.0 / 2.0 + std::sqrt(17745.0 / 4.0)
    )
                    - std::sqrt(105.0 / 4.0) - 13.0 / 2.0;
143             break;
            default:
145             ExceptionObject err(__FILE__, __LINE__);
                err.SetLocation(ITK_LOCATION);
147             err.SetDescription("SplineOrder must be between 0 and 5. \
                    Requested spline order has not been implemented yet.");
149             throw err;
                break;
151     }
    }
153
    // Conditions initiales du filtre causal données par le théorème 3.11
155 template< typename TInputImage, typename TOutputImage >
    void BSplineDecompositionImageFilter< TInputImage, TOutputImage >
157 ::SetInitialCausalCoefficient(double z)
    {
159     CoeffType      sum;
        double         zn, z2n, iz;
161     typename TInputImage::SizeValueType horizon;

163     // horizon correspond à k_0 dans le théorème 3.11
        horizon = m_DataLength[m_IteratorDirection];
165     zn = z;
        if ( m_Tolerance > 0.0 )
167     {
            horizon = (typename TInputImage::SizeValueType)
169             std::ceil( std::log(m_Tolerance) / std::log( std::fabs(z) ) );
        }
171
        /* Calcul des conditions initiales en fonction de la valeur de horizon
173     (voir théorème 3.11) */
        if ( horizon < m_DataLength[m_IteratorDirection] )
175     {
            sum = m_Scratch[0];
177         for ( unsigned int n = 1; n < horizon; n++ )
            {
179             sum += zn * m_Scratch[n];
```

```
                zn *= z;
            }
            m_Scratch[0] = sum;
        }
        else
        {
            iz = 1.0 / z;
            z2n = std::pow( z, (double)( m_DataLength[m_IteratorDirection] - 1L ) )
    ;
            sum = m_Scratch[0]
                + z2n * m_Scratch[m_DataLength[m_IteratorDirection] - 1L];
            z2n *= z2n * iz;
            for (unsigned int n = 1; n<=(m_DataLength[m_IteratorDirection]-2); n++)
              {
              sum += ( zn + z2n ) * m_Scratch[n];
              zn *= z;
              z2n *= iz;
              }
            m_Scratch[0] = sum / ( 1.0 - zn * zn );
        }
    }

    // Conditions initiales du filtre anticausal données par le théorème 3.11
    template< typename TInputImage, typename TOutputImage >
    void BSplineDecompositionImageFilter< TInputImage, TOutputImage >
    ::SetInitialAntiCausalCoefficient(double z)
    {
      m_Scratch[m_DataLength[m_IteratorDirection] - 1] = ( z / ( z * z - 1.0 ) )
        * ( z * m_Scratch[m_DataLength[m_IteratorDirection] - 2]
        + m_Scratch[m_DataLength[m_IteratorDirection] - 1] );
    }

    // Calcul des coefficients pour une interpolation ND
    template< typename TInputImage, typename TOutputImage >
    void BSplineDecompositionImageFilter< TInputImage, TOutputImage >
    ::DataToCoefficientsND()
    {
        OutputImagePointer output = this->GetOutput();
        Size< ImageDimension > size = output->GetBufferedRegion().GetSize();
        unsigned int count = output->GetBufferedRegion().GetNumberOfPixels()
            / size[0] * ImageDimension;
        ProgressReporter progress(this, 0, count, 10);

        // Initialisation de l'image de coefficients avec l'image à interpoler
        this->CopyImageToImage();

        // Boucle pour interpoler en 1D selon chaque direction (x, y et
    éventuellement z)
        for ( unsigned int n = 0; n < ImageDimension; n++ )
        {
            m_IteratorDirection = n;

            // Itérateur pour parcourir l'image ligne par ligne
            OutputLinearIterator CIterator( output, output->GetBufferedRegion() );
            CIterator.SetDirection(m_IteratorDirection);

            // Pour chaque ligne dans la direction m_IteratorDirection :
            while ( !CIterator.IsAtEnd() )
            {
                /* Copie de la ligne de coefficients dans la variable m_Scratch
                de l'interpolation 1D ; au début coefficients = input image */
                this->CopyCoefficientsToScratch(CIterator);
```

```
                // Calcul des coefficients pour interpolation 1D de la ligne courante
241             this->DataToCoefficients1D();
                CIterator.GoToBeginOfLine();
243             // Remplacer les coefficients utilisés par ceux trouvés
                this->CopyScratchToCoefficients(CIterator);    // m_Scratch = m_Image;
245             // L'itérateur avance d'une ligne
                CIterator.NextLine();
247             progress.CompletedPixel();
            }
249         }
        }
251
        // Allocation de mémoire et enclanchement du calcul des coefficients
253     template< typename TInputImage, typename TOutputImage >
        void BSplineDecompositionImageFilter< TInputImage, TOutputImage >
255     ::GenerateData()
        {
257         InputImageConstPointer inputPtr = this->GetInput();
            m_DataLength = inputPtr->GetBufferedRegion().GetSize();
259         typename TOutputImage::SizeValueType maxLength = 0;
            for ( unsigned int n = 0; n < ImageDimension; n++ )
261         {
                if ( m_DataLength[n] > maxLength )
263             {
                    maxLength = m_DataLength[n];
265             }
            }
267         m_Scratch.resize(maxLength);
            OutputImagePointer outputPtr = this->GetOutput();
269         outputPtr->SetBufferedRegion( outputPtr->GetRequestedRegion() );
            outputPtr->Allocate();
271
            this->DataToCoefficientsND();
273
            m_Scratch.clear();
275     }
}
277 #endif
```

itkBSplineInterpolateImageFunction.hxx

```
1  /*=========================================================================
   *
3  *  Copyright Insight Software Consortium
   *
5  *  Licensed under the Apache License, Version 2.0 (the "License");
   *  you may not use this file except in compliance with the License.
7  *  You may obtain a copy of the License at
   *
9  *         http://www.apache.org/licenses/LICENSE-2.0.txt
   *
11 *  Unless required by applicable law or agreed to in writing, software
   *  distributed under the License is distributed on an "AS IS" BASIS,
13 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   *  See the License for the specific language governing permissions and
15 *  limitations under the License.
   *
17 *=========================================================================*/
   /*=========================================================================
19 *
```

```
 *  Portions of this file are subject to the VTK Toolkit Version 3 copyright.
 *
 *  Copyright (c) Ken Martin, Will Schroeder, Bill Lorensen
 *
 *  For complete copyright, license and disclaimer of warranty information
 *  please refer to the NOTICE file at the top of the ITK source tree.
 *
 *=========================================================================*/

#ifndef itkBSplineInterpolateImageFunction_hxx
#define itkBSplineInterpolateImageFunction_hxx
#include "itkBSplineInterpolateImageFunction.h"
#include "itkImageLinearIteratorWithIndex.h"
#include "itkImageRegionConstIteratorWithIndex.h"
#include "itkImageRegionIterator.h"
#include "itkVector.h"
#include "itkMatrix.h"

namespace itk
{
    // Constructeur
    template< typename TImageType, typename TCoordRep, typename TCoefficientType >
    BSplineInterpolateImageFunction< TImageType, TCoordRep, TCoefficientType >
    ::BSplineInterpolateImageFunction()
    {
        // Pour la parallélisation
        m_NumberOfThreads = 1;
        m_ThreadedEvaluateIndex = ITK_NULLPTR;
        m_ThreadedWeights = ITK_NULLPTR;
        m_ThreadedWeightsDerivative = ITK_NULLPTR;

        // Création de filtres pour le calcul des coefficients d'interpolation
        m_CoefficientFilter = CoefficientFilter::New();
        m_Coefficients = CoefficientImageType::New();

        // Paramètres sur les B-splines
        m_SplineOrder = 0;
        unsigned int SplineOrder = 3;
        this->SetSplineOrder(SplineOrder);
        this->m_UseImageDirection = true;
    }

    // Fixer l'image à interpoler et calcul des coefficients d'interpolation
    template< typename TImageType, typename TCoordRep, typename TCoefficientType >
    void BSplineInterpolateImageFunction< TImageType, TCoordRep, TCoefficientType >
    ::SetInputImage(const TImageType *inputData)
    {
        if ( inputData )
        {
            m_CoefficientFilter->SetInput(inputData);
            m_CoefficientFilter->Update();
            m_Coefficients = m_CoefficientFilter->GetOutput();
            Superclass::SetInputImage(inputData);
            m_DataLength = inputData->GetBufferedRegion().GetSize();
        }
        else
        {
            m_Coefficients = ITK_NULLPTR;
        }
    }

    // Fixer l'ordre des splines et calcul de variables de travail
```

```
template< typename TImageType, typename TCoordRep, typename TCoefficientType >
void BSplineInterpolateImageFunction< TImageType, TCoordRep, TCoefficientType >
::SetSplineOrder(unsigned int SplineOrder)
{
    if ( SplineOrder == m_SplineOrder )
    {
        return;
    }
    m_SplineOrder = SplineOrder;
    m_CoefficientFilter->SetSplineOrder(SplineOrder);
    m_MaxNumberInterpolationPoints = 1;
    for ( unsigned int n = 0; n < ImageDimension; n++ )
    {
        m_MaxNumberInterpolationPoints *= ( m_SplineOrder + 1 );
    }
    this->GeneratePointsToIndex();
}

// Evaluation des B-splines utiles à l'évaluation de l'interpolant en un point
template< typename TImageType, typename TCoordRep, typename TCoefficientType >
void BSplineInterpolateImageFunction< TImageType, TCoordRep, TCoefficientType >
::SetInterpolationWeights(const ContinuousIndexType & x, const vnl_matrix<long>
& EvaluateIndex, vnl_matrix< double > & weights, unsigned int splineOrder)
const
{
    double w, w2, w4, t, t0, t1;
    switch ( splineOrder )
    {
        case 3:
        {
            for ( unsigned int n = 0; n < ImageDimension; n++ )
            {
                // w=x-k où k est le plus petit index servant à l'interpolation
                w = x[n] - (double)EvaluateIndex[n][1];
                weights[n][3] = ( 1.0 / 6.0 ) * w * w * w;
                weights[n][0] = ( 1.0 / 6.0 ) + 0.5 * w * (w - 1.0)
                    - weights[n][3];
                weights[n][2] = w + weights[n][0] - 2.0 * weights[n][3];
                weights[n][1] = 1.0 - weights[n][0] - weights[n][2]
                    - weights[n][3];
            }
            break;
        }
        case 0:
        {
            for ( unsigned int n = 0; n < ImageDimension; n++ )
            {
                weights[n][0] = 1;
            }
          break;
        }
        case 1:
        {
            for ( unsigned int n = 0; n < ImageDimension; n++ )
            {
                w = x[n] - (double)EvaluateIndex[n][0];
                weights[n][1] = w;
                weights[n][0] = 1.0 - w;
            }
            break;
        }
        case 2:
```

```
143             {
                    for ( unsigned int n = 0; n < ImageDimension; n++ )
145                 {
                        w = x[n] - (double)EvaluateIndex[n][1];
147                     weights[n][1] = 0.75 - w * w;
                        weights[n][2] = 0.5 * ( w - weights[n][1] + 1.0 );
149                     weights[n][0] = 1.0 - weights[n][1] - weights[n][2];
                    }
151                 break;
                }
153             case 4:
                {
155                 for ( unsigned int n = 0; n < ImageDimension; n++ )
                    {
157                     w = x[n] - (double)EvaluateIndex[n][2];
                        w2 = w * w;
159                     t = ( 1.0 / 6.0 ) * w2;
                        weights[n][0] = 0.5 - w;
161                     weights[n][0] *= weights[n][0];
                        weights[n][0] *= ( 1.0 / 24.0 ) * weights[n][0];
163                     t0 = w * ( t - 11.0 / 24.0 );
                        t1 = 19.0 / 96.0 + w2 * ( 0.25 - t );
165                     weights[n][1] = t1 + t0;
                        weights[n][3] = t1 - t0;
167                     weights[n][4] = weights[n][0] + t0 + 0.5 * w;
                        weights[n][2] = 1.0 - weights[n][0] - weights[n][1]
169                         - weights[n][3] - weights[n][4];
                    }
171                 break;
                }
173             case 5:
                {
175                 for ( unsigned int n = 0; n < ImageDimension; n++ )
                    {
177                     w = x[n] - (double)EvaluateIndex[n][2];
                        w2 = w * w;
179                     weights[n][5] = ( 1.0 / 120.0 ) * w * w2 * w2;
                        w2 -= w;
181                     w4 = w2 * w2;
                        w -= 0.5;
183                     t = w2 * ( w2 - 3.0 );
                        weights[n][0] = ( 1.0 / 24.0 ) * ( 1.0 / 5.0 + w2 + w4 )
185                         - weights[n][5];
                        t0 = ( 1.0 / 24.0 ) * ( w2 * ( w2 - 5.0 ) + 46.0 / 5.0 );
187                     t1 = ( -1.0 / 12.0 ) * w * ( t + 4.0 );
                        weights[n][2] = t0 + t1;
189                     weights[n][3] = t0 - t1;
                        t0 = ( 1.0 / 16.0 ) * ( 9.0 / 5.0 - t );
191                     t1 = ( 1.0 / 24.0 ) * w * ( w4 - w2 - 5.0 );
                        weights[n][1] = t0 + t1;
193                     weights[n][4] = t0 - t1;
                    }
195                 break;
                }
197             default:
                {
199                 ExceptionObject err(__FILE__, __LINE__);
                    err.SetLocation(ITK_LOCATION);
201                 err.SetDescription("SplineOrder must be between 0 and 5. \
                        Requested spline order has not been implemented yet.");
203                 throw err;
                    break;
```

```
205             }
            }
207     }

209     /* Evaluation des B-splines pour l'évaluation en un point de la dérivée
        exacte de l'interpolant */
211     template< typename TImageType, typename TCoordRep, typename TCoefficientType >
        void BSplineInterpolateImageFunction< TImageType, TCoordRep, TCoefficientType >
213     ::SetDerivativeWeights(const ContinuousIndexType & x, const vnl_matrix<long>
        & EvaluateIndex, vnl_matrix<double> & weights, unsigned int splineOrder) const
215     {
            double w, w1, w2, w3, w4, w5, t, t0, t1, t2;
217         int    derivativeSplineOrder = (int)splineOrder - 1;
            switch ( derivativeSplineOrder )
219         {
                case -1:
221             {
                    for ( unsigned int n = 0; n < ImageDimension; n++ )
223                 {
                        weights[n][0] = 0.0;
225                 }
                    break;
227             }
                case 0:
229             {
                    for ( unsigned int n = 0; n < ImageDimension; n++ )
231                 {
                        weights[n][0] = -1.0;
233                     weights[n][1] =  1.0;
                    }
235                 break;
                }
237             case 1:
                {
239             for ( unsigned int n = 0; n < ImageDimension; n++ )
                {
241                 w = x[n] + 0.5 - (double)EvaluateIndex[n][1];
                    w1 = 1.0 - w;
243                 weights[n][0] = 0.0 - w1;
                    weights[n][1] = w1 - w;
245                 weights[n][2] = w;
                }
247             break;
                }
249             case 2:
                {
251                 for ( unsigned int n = 0; n < ImageDimension; n++ )
                    {
253                     w = x[n] + .5 - (double)EvaluateIndex[n][2];
                        w2 = 0.75 - w * w;
255                     w3 = 0.5 * ( w - w2 + 1.0 );
                        w1 = 1.0 - w2 - w3;
257                     weights[n][0] = 0.0 - w1;
                        weights[n][1] = w1 - w2;
259                     weights[n][2] = w2 - w3;
                        weights[n][3] = w3;
261                 }
                    break;
263             }
                case 3:
265             {
                    for ( unsigned int n = 0; n < ImageDimension; n++ )
```

```
267                 {
                        w = x[n] + 0.5 - (double)EvaluateIndex[n][2];
269                     w4 = ( 1.0 / 6.0 ) * w * w * w;
                        w1 = ( 1.0 / 6.0 ) + 0.5 * w * ( w - 1.0 ) - w4;
271                     w3 = w + w1 - 2.0 * w4;
                        w2 = 1.0 - w1 - w3 - w4;
273                     weights[n][0] = 0.0 - w1;
                        weights[n][1] = w1 - w2;
275                     weights[n][2] = w2 - w3;
                        weights[n][3] = w3 - w4;
277                     weights[n][4] = w4;
                    }
279                 break;
                }
281             case 4:
                {
283                 for ( unsigned int n = 0; n < ImageDimension; n++ )
                    {
285                     w = x[n] + .5 - (double)EvaluateIndex[n][3];
                        t2 = w * w;
287                     t = ( 1.0 / 6.0 ) * t2;
                        w1 = 0.5 - w;
289                     w1 *= w1;
                        w1 *= ( 1.0 / 24.0 ) * w1;
291                     t0 = w * ( t - 11.0 / 24.0 );
                        t1 = 19.0 / 96.0 + t2 * ( 0.25 - t );
293                     w2 = t1 + t0;
                        w4 = t1 - t0;
295                     w5 = w1 + t0 + 0.5 * w;
                        w3 = 1.0 - w1 - w2 - w4 - w5;
297                     weights[n][0] = 0.0 - w1;
                        weights[n][1] = w1 - w2;
299                     weights[n][2] = w2 - w3;
                        weights[n][3] = w3 - w4;
301                     weights[n][4] = w4 - w5;
                        weights[n][5] = w5;
303                 }
                    break;
305             }
                default:
307             {
                    ExceptionObject err(__FILE__, __LINE__);
309                 err.SetLocation(ITK_LOCATION);
                    err.SetDescription("SplineOrder (for derivatives) must be between \
311                     1 and 5. Requested spline order has not been implemented yet.")
    ;
                    throw err;
313                 break;
                }
315         }
    }
317
    /* Génération de la variable de travail m_PointsToIndex et calculs
319 liés à la parallélisation */
    template< typename TImageType, typename TCoordRep, typename TCoefficientType >
321 void BSplineInterpolateImageFunction< TImageType, TCoordRep, TCoefficientType >
    ::GeneratePointsToIndex()
323 {
        delete[] m_ThreadedEvaluateIndex;
325     m_ThreadedEvaluateIndex = new vnl_matrix< long >[m_NumberOfThreads];
        delete[] m_ThreadedWeights;
327     m_ThreadedWeights = new vnl_matrix< double >[m_NumberOfThreads];
```

```
        delete[] m_ThreadedWeightsDerivative;
329     m_ThreadedWeightsDerivative = new vnl_matrix< double >[m_NumberOfThreads];
        for ( unsigned int i = 0; i < m_NumberOfThreads; i++ )
331     {
            m_ThreadedEvaluateIndex[i].set_size(ImageDimension, m_SplineOrder + 1);
333         m_ThreadedWeights[i].set_size(ImageDimension, m_SplineOrder + 1);
            m_ThreadedWeightsDerivative[i].set_size(ImageDimension,
335             m_SplineOrder + 1);
        }
337
        m_PointsToIndex.resize(m_MaxNumberInterpolationPoints);
339     for ( unsigned int p = 0; p < m_MaxNumberInterpolationPoints; p++ )
        {
341         int pp = p;
            unsigned long indexFactor[ImageDimension];
343         indexFactor[0] = 1;
            for ( int j = 1; j < static_cast< int >( ImageDimension ); j++ )
345         {
                indexFactor[j] = indexFactor[j - 1] * ( m_SplineOrder + 1 );
347         }
            for (int j = (static_cast<int>(ImageDimension)-1); j >= 0; j--)
349         {
                m_PointsToIndex[p][j] = pp / indexFactor[j];
351             pp = pp % indexFactor[j];
            }
353     }
    }
355
    // Calcul de la région de support des B-splines
357 template< typename TImageType, typename TCoordRep, typename TCoefficientType >
    void BSplineInterpolateImageFunction< TImageType, TCoordRep, TCoefficientType >
359 ::DetermineRegionOfSupport(vnl_matrix< long > & evaluateIndex,
    const ContinuousIndexType & x, unsigned int splineOrder) const
361 {
        // Décalage de 1/2 pour les ordres de splines pairs
363     const float halfOffset = splineOrder & 1 ? 0.0 : 0.5;
        for (unsigned int n = 0; n < ImageDimension; n++)
365     {
            // Index le plus bas servant à l'interpolation
367         long indx = (long)std::floor((float)x[n] + halfOffset) - splineOrder/2;
            // Calcul des index suivants
369         for (unsigned int k = 0; k <= splineOrder; k++)
            {
371             evaluateIndex[n][k] = indx++;
            }
373     }
    }
375
    /* Calcul des conditions miroirs lorsque les indexs calculés par
377 DetermineRegionOfSupport sont hors de l'image */
    template< typename TImageType, typename TCoordRep, typename TCoefficientType >
379 void BSplineInterpolateImageFunction< TImageType, TCoordRep, TCoefficientType >
    ::ApplyMirrorBoundaryConditions(vnl_matrix< long > & evaluateIndex,
381 unsigned int splineOrder) const
    {
383     // Index du premier pixel
        const IndexType startIndex = this->GetStartIndex();
385     // Index du dernier pixel
        const IndexType endIndex = this->GetEndIndex();
387     for ( unsigned int n = 0; n < ImageDimension; n++ )
        {
389         // Si l'image est de longueur 1 selon une des directions
```

```
            if ( m_DataLength[n] == 1 )
391         {
                for ( unsigned int k = 0; k <= splineOrder; k++ )
393             {
                    // Seul le pixel 0 est utilisé pour cette direction
395                 evaluateIndex[n][k] = 0;
                }
397         }
            else
399         {
                for ( unsigned int k = 0; k <= splineOrder; k++ )
401             {
                    // Si l'index est plus petit que le premier index
403                 if ( evaluateIndex[n][k] < startIndex[n] )
                    {
405                     // Conditions miroir
                        evaluateIndex[n][k] = startIndex[n] +
407                         ( startIndex[n] - evaluateIndex[n][k] );
                    }
409                 // Si l'index est plus grand que le dernier index
                    if ( evaluateIndex[n][k] >= endIndex[n] )
411                 {
                        // Conditions miroir
413                     evaluateIndex[n][k] = endIndex[n] -
                            ( evaluateIndex[n][k] - endIndex[n] );
415                 }
                }
417         }
        }
419 }

421 // Evaluation de l'interpolant en un point
    template<typename TImageType, typename TCoordRep, typename TCoefficientType>
423 typename BSplineInterpolateImageFunction<TImageType, TCoordRep,
    TCoefficientType>::OutputType BSplineInterpolateImageFunction<TImageType,
425 TCoordRep, TCoefficientType>::EvaluateAtContinuousIndexInternal(const
    ContinuousIndexType & x, vnl_matrix< long > & evaluateIndex,
427 vnl_matrix< double > & weights) const
    {
429     // Calcul de la région de support
        this->DetermineRegionOfSupport( ( evaluateIndex ), x, m_SplineOrder );
431
        // Evaluation des B-splines
433     SetInterpolationWeights(x, ( evaluateIndex ), ( weights ), m_SplineOrder);

435     // Calcul des conditions miroir si evaluateIndex sort des limites de l'
    image
        this->ApplyMirrorBoundaryConditions( ( evaluateIndex ), m_SplineOrder );
437
        // Calcul de l'interpolant en x
439     double    interpolated = 0.0;
        IndexType coefficientIndex;
441     // On parcourt chaque point du cube d'interpolation n-dimensionnel
        for ( unsigned int p = 0; p < m_MaxNumberInterpolationPoints; p++ )
443     {
            double w = 1.0;
445         for ( unsigned int n = 0; n < ImageDimension; n++ )
            {
447             unsigned int indx = m_PointsToIndex[p][n];
                w *= ( weights )[n][indx];
449             coefficientIndex[n] = ( evaluateIndex )[n][indx];
            }
```

```
451             // Combinaison linéaire des coefficients et des B-splines évaluées
                interpolated += w * m_Coefficients->GetPixel(coefficientIndex);
453         }
            return ( interpolated );
455     }

457     // Evaluation de la dérivée exacte de l'interpolant en un point x
        template<typename TImageType, typename TCoordRep, typename TCoefficientType>
459     typename BSplineInterpolateImageFunction<TImageType, TCoordRep,
        TCoefficientType>::CovariantVectorType BSplineInterpolateImageFunction
461     < TImageType, TCoordRep, TCoefficientType >
        ::EvaluateDerivativeAtContinuousIndexInternal(const ContinuousIndexType & x,
463     vnl_matrix<long> & evaluateIndex, vnl_matrix<double> & weights,
        vnl_matrix< double > & weightsDerivative) const
465     {
            // Calcul de la région de support
467         this->DetermineRegionOfSupport( (evaluateIndex), x, m_SplineOrder );

469         // Evaluation des B-splines nécessaires au calcul de l'interpolant
            SetInterpolationWeights(x, (evaluateIndex), (weights), m_SplineOrder);
471
            /* Evaluation des B-splines nécessaires au calcul de la dérivée de
473         l'interpolant */
            SetDerivativeWeights(x, (evaluateIndex), (weightsDerivative),
475             m_SplineOrder);

477         /* Calcul des conditions miroir au cas où evaluateIndex sort des
            index de l'image*/
479         this->ApplyMirrorBoundaryConditions( ( evaluateIndex ), m_SplineOrder );

481         // Image en entrée et son spacing
            const InputImageType *inputImage = this->GetInputImage();
483         const typename InputImageType::SpacingType & spacing =
                inputImage->GetSpacing();
485
            // Calcul de la dérivée
487         CovariantVectorType derivativeValue;
            double              tempValue;
489         IndexType           coefficientIndex;
            for ( unsigned int n = 0; n < ImageDimension; n++ )
491         {
                derivativeValue[n] = 0.0;
493             for ( unsigned int p = 0; p < m_MaxNumberInterpolationPoints; p++ )
                {
495                 tempValue = 1.0;
                    // Selon chaque direction:
497                 for ( unsigned int n1 = 0; n1 < ImageDimension; n1++ )
                    {
499                     unsigned int indx;
                        indx = m_PointsToIndex[p][n1];
501                     coefficientIndex[n1] = ( evaluateIndex )[n1][indx];
                        /* Utiliser soit les B-splines pour l'interpolant si on n'
503                     est pas dans la même direction que la boucle sur n, soit les
                        B-splines pour la dérivée si on est dans la même direction*/
505                     if ( n1 == n )
                        {
507                         tempValue *= ( weightsDerivative )[n1][indx];
                        }
509                     else
                        {
511                         tempValue *= ( weights )[n1][indx];
                        }
```

```
513                 }
                    /* Combinaison linéaire avec les coefficients d'interpolation
515                 et les B-splines évaluées */
                    derivativeValue[n] += m_Coefficients->GetPixel(coefficientIndex)
517                     * tempValue;
                }
519             // Tenir compte de l'espace physique
                derivativeValue[n] /= spacing[n];
521         }

523         // Tenir compte de l'espace physique
            if ( this->m_UseImageDirection )
525         {
                CovariantVectorType orientedDerivative;
527             inputImage->TransformLocalVectorToPhysicalVector(derivativeValue,
                    orientedDerivative);
529             return orientedDerivative;
            }
531
            return ( derivativeValue );
533     }
    }
535 #endif
```

Fonctions importantes supplémentaires, extraites de *itkBSplineInterpolateImageFunction.h* :

```
1  // Fonction appelée par ComputeUpdate de itkLevelSetMotionRegistrationFunction
   // Fonction chargée d'interpoler en un point de l'espace physique
3  virtual OutputType Evaluate(const PointType & point) const ITK_OVERRIDE
   {
5      ContinuousIndexType index;

7      // De l'espace physique vers un index continu
       this->GetInputImage()->TransformPhysicalPointToContinuousIndex(point, index);
9
       return ( this->EvaluateAtContinuousIndex(index) );
11 }

13 // Fonction chargée d'interpoler en un index continu
   virtual OutputType EvaluateAtContinuousIndex(const ContinuousIndexType & index)
15 const ITK_OVERRIDE
   {
17     vnl_matrix< long >   evaluateIndex( ImageDimension, ( m_SplineOrder + 1 ) );
       vnl_matrix< double > weights( ImageDimension, ( m_SplineOrder + 1 ) );
19
       return this->EvaluateAtContinuousIndexInternal(index, evaluateIndex, weights);
21 }

23 /* Fonction chargée d'évaluer la dérivée exacte de l'interpolant en un point
   de l'espace physique
25 Fonction appelée par ComputeUpdate de itkLevelSetMotionRegistrationFunction */
   CovariantVectorType EvaluateDerivative(const PointType & point) const
27 {
       ContinuousIndexType index;
29
       this->GetInputImage()->TransformPhysicalPointToContinuousIndex(point, index);
31     return ( this->EvaluateDerivativeAtContinuousIndex(index) );
   }
33
```

```
// Fonction chargée d'évaluer la dérivée exacte de l'interpolant en un index
    continu
CovariantVectorType EvaluateDerivativeAtContinuousIndex(const
ContinuousIndexType & x) const
{
    vnl_matrix< long >   evaluateIndex( ImageDimension, ( m_SplineOrder + 1 ) );
    vnl_matrix< double > weights( ImageDimension, ( m_SplineOrder + 1 ) );
    vnl_matrix< double > weightsDerivative( ImageDimension, ( m_SplineOrder + 1 ) )
    ;

    return this->EvaluateDerivativeAtContinuousIndexInternal(x, evaluateIndex,
        weights, weightsDerivative);
}
```