HAL
open science

A Fast and Log-Euclidean Polyaffine Framework for Locally Affine Registration

Vincent Arsigny, Olivier Commowick, Xavier Pennec, Nicholas Ayache

To cite this version:

Vincent Arsigny, Olivier Commowick, Xavier Pennec, Nicholas Ayache. A Fast and Log-Euclidean Polyaffine Framework for Locally Affine Registration. [Research Report] RR-5865, INRIA. 2006, pp.46. inria-00070161

HAL Id: inria-00070161

https://inria.hal.science/inria-00070161

Submitted on 19 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Fast and Log-Euclidean Polyaffine Framework for Locally Affine Registration

Vincent Arsigny — Olivier Commowick — Xavier Pennec — Nicholas Ayache

N° 5865

March, 2006

Thème BIO

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--5865--FR+ENG

A Fast and Log-Euclidean Polyaffine Framework for Locally Affine Registration

Vincent Arsigny, Olivier Commowick , Xavier Pennec , Nicholas Ayache

Thème BIO — Systèmes biologiques
Projet Asclepios

Rapport de recherche n° 5865 — March, 2006 — 46 pages

Abstract: In this article, we focus on the parameterization of non-rigid geometrical deformations with a small number of flexible degrees of freedom . In previous work, we proposed a general framework called *polyaffine* to parameterize deformations with a finite number of rigid or affine components, while guaranteeing the invertibility of global deformations. However, this framework lacks some important properties: the inverse of a polyaffine transformation is not polyaffine in general, and the polyaffine fusion of affine components is not invariant with respect to a change of coordinate system. We present here a novel general framework, called *Log-Euclidean polyaffine*, which overcomes these defects.

We also detail a simple algorithm, the *Fast Polyaffine Transform*, which allows to compute very efficiently Log-Euclidean polyaffine transformations and their inverses on regular grids. The results presented here on real 3D locally affine registration suggest that our novel framework provides a general and efficient way of fusing local rigid or affine deformations into a global invertible transformation without introducing artifacts, independently of the way local deformations are first estimated.

Last but not least, we show in this article that the Log-Euclidean polyaffine framework is implicitely based on a Log-Euclidean framework for rigid and affine transformations, which generalizes to linear transformations the Log-Euclidean framework recently proposed for tensors. We detail in the Appendix of this article the properties of this novel framework, which allows a straightforward and efficient generalization to linear transformations of classical vectorial tools, with excellent theoretical properties. In particular, we propose here a simple generalization to locally rigid or affine deformations of a visco-elastic regularization energy used for dense transformations.

Key-words: Locally affine transformations, medical imaging, ODE, diffeomorphisms, polyaffine transformations, Log-Euclidean, non-rigid registration

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 —Télécopie : +33 4 92 38 77 65

Un cadre polyaffine rapide et log-euclidien pour le recalage localement affine.

Résumé : Dans cet article, nous nous concentrons sur la paramétrisation des déformations géométriques régulières par un nombre restreint de degrés de liberté flexibles. Précédemment, nous avions proposé un cadre général appelé *polyaffine* pour paramètrer des déformations avec un nombre fini de composantes rigides ou pour affines, tout en garantissant l'inversibilité des déformations globales. Cependant, plusieurs propriétés importantes font défaut à ce cadre : l'inverse d'une transformation polyaffine n'est pas polyaffine en général, et la fusion polyaffine n'est pas invariante par changement de système de coordonnées. Nous présentons ici un nouveau cadre général, appelé *polyaffine log-euclidien*, qui corrige ces défauts.

Nous détaillons également un algorithme simple, la *Transformée Polyaffine rapide*, qui permet de calculer très efficacement les transformations log-euclidiennes polyaffines et leurs inverses sur une grille régulière. Les résultats de recalage localement affines 3D présentés ici suggèrent que notre nouveau cadre fournit une manière générale et efficace de fusionner des déformations localement rigides ou affines en une transformation inversible globale sans introduire d'artefact, indépendamment de la manière dont les déformations locales sont préalablement estimées.

Enfin, nous montrons dans cet article que le cadre log-euclidien polyaffine est implicitement basé sur un cadre log-euclidien pour les transformations rigides et affines, qui généralise aux transformations linéaires le cadre log-euclidien récemment proposé pour les tenseurs. Nous détaillons dans l'appendice de cet article les propriétés de ce nouveau cadre, qui permet une généralisation directe et efficace des outils vectoriels classiques aux transformations linéaires, avec d'excellentes propriétés théoriques. En particulier, nous proposons ici une généralisation simple aux déformations localement rigides ou affines d'une énergie de régularité visco-élastique utilisée pour des transformations denses.

Mots-clés : Transformations localement affines, imagerie médicale, EDO, difféomorphismes, transformations polyaffines, log-euclidien, recalage non-rigide.

Contents

1 Introduction **4**

2 A Log-Euclidean Polyaffine Framework **4**
2.1 Previous Polyaffine Framework 4
2.2 Simpler Speed Vectors for Affine Transformations 10
2.3 Log-Euclidean Polyaffine Transformations 13

3 Fast Polyaffine Transform **15**
3.1 Matrix Exponential and the 'Scaling and Squaring' Method 20
3.2 A 'Scaling and Squaring' Method for LEPTs 20
3.3 2D Synthetic Experiments 23
3.4 3D MRI Example 34

4 Conclusion and Perspectives **37**

A A Log-Euclidean Framework for Rigid and Affine Transformations **38**
A.1 Log-Euclidean Metrics 38
A.2 Invariance Properties 40
A.3 Log-Euclidean Regularization for Locally Rigid or Affine Registration 41
A.4 Numerical Implementation of Matrix Logarithm. 44

1 Introduction

The registration of medical images is in general a difficult problem, and numerous methods and tools have been already devised to address this task [1]. Still currently, much effort continues to be devoted to finding adequate measures of similarity, relevant parameterizations of geometrical deformations, efficient optimization methods, or realistic mechanical models of deformations, depending on the precise type of registration considered.

In this article, we focus on the parameterization of non-rigid geometrical deformations with a small number of flexible degrees of freedom . This type of parameterization is particularly well-adapted for example to the registration of articulated structures [2] and to the registration of histological slices [3, 4]. After a *global* affine (or rigid) alignment, this sort of parameterization also allows a finer *local* registration with *very smooth* transformations [5, 6, 7, 8].

In [4], we parameterized deformations with a small number of *rigid or affine components*, which can model smoothly a large variety of local deformations. We provided a general framework to fuse these components into a global transformation, called *polyrigid* or *polyaffine*, whose *invertibility* is guaranteed. However, this framework lacks some important properties: the inverse of a polyaffine transformation is not polyaffine in general, and the polyaffine fusion of affine components is not invariant with respect to a change of coordinate system (i.e. is not *affine-invariant*). Here, we present a novel general framework to fuse rigid or affine components, called *Log-Euclidean polyaffine*, which overcomes these defects and yields transformations which can be very efficiently computed.

The sequel of this article is organized as follows. In Section 2, we present the Log-Euclidean polyaffine framework and its intuitive properties. Then, we present the *Fast Polyaffine Transform* (FPT), which allows to compute very efficiently Log-Euclidean polyaffine transformations (LEPTs) and their inverses on a regular grid. Afterward, we apply the FPT to a real 3D example, where affine components are estimated with the algorithm of [5]. *Without introducing artifacts*, our novel fusion ensures the invertibility of the global transformation. Last but not least, we present in an appendix the properties of the Log-Euclidean framework for rigid and affine transformations on which our polyaffine Log-Euclidean framework is implicitely based. This Log-Euclidean framework is the analoguous of the framework we presented in [9] for tensors.

2 A Log-Euclidean Polyaffine Framework

2.1 Previous Polyaffine Framework

Before presenting our novel polyaffine framework let us briefly recall the original polyaffine framework, described in [4].

The idea is to define transformations that exhibit a locally affine behavior, with nice invertibility properties. Following the seminal work of [10], we model here such transformations by a finite number N of affine *components*. Precisely, each component i consists of

an affine transformation T_i and of a non-negative *weight function* $w_i(x)$ which models its spatial extension: the influence of the ith component at point x is proportional to $w_i(x)$. Furthermore, we assume that for all x, $\sum_{i=1}^{N} w_i(x) = 1$, i.e. the weights are normalized.

Fusion of Displacements. In order to obtain a global transformation from several weighted components, the classical approach to fuse the N components, given in [11], simply consists in averaging the associated displacements according to the weights:

$$T(x) = \sum_{i=1}^{N} w_i(x) T_i(x). \tag{1}$$

The transformation obtained using (1) is smooth, but this approach has one major drawback: although each component is invertible, the resulting global transformation is *not invertible* in general. To remedy this, it was proposed in [4] to rely on the averaging of some *infinitesimal* displacements associated to each affine component instead. The resulting global transformation is obtained by integrating an Ordinary Differential Equation (ODE), which is computationally more expensive but guarantees its invertibility and also yields a simple form for its inverse. The nice invertibility properties of this approach are illustrated by Fig. 1.

Polyaffine Framework. The polyaffine approach can be decomposed into three steps. They are given below:

- **Step 1: Associating Speed Vectors to Affine Transformations.** The idea behind the polyaffine framework is essentially the following: for each component i, one can define a family of *speed vector fields* $V_i(., s)$ parameterized by s, which is a time parameter varying continuously between 0 and 1. $V_i(., s)$ satisfy a consistency property with T_i: when integrated between time 0 and 1, they should give back the transformation T. Hence the following definition:

  Definition 1. *The family of vector fields $V(., s)$, where s belongs to $[0, 1]$, is consistent with the transformation T if and only if its integration between time 0 and 1 gives back the transformation T:*

  1. *for any initial condition x_0 one can integrate between 0 and 1 the differential equation $\dot{x} = V(x, s)$ so that $x(1)$ exists.*
  2. *$x(1)$ is equal to $T(x_0)$.*

  Several possible choices exist to associate speed vector to affine transformations. One of the main contributions of this work is precisely to propose a novel choice for such speed vectors. Interestingly, we do not know at present how many other choices exist and whether they might have even better properties than the ones we have found so far.

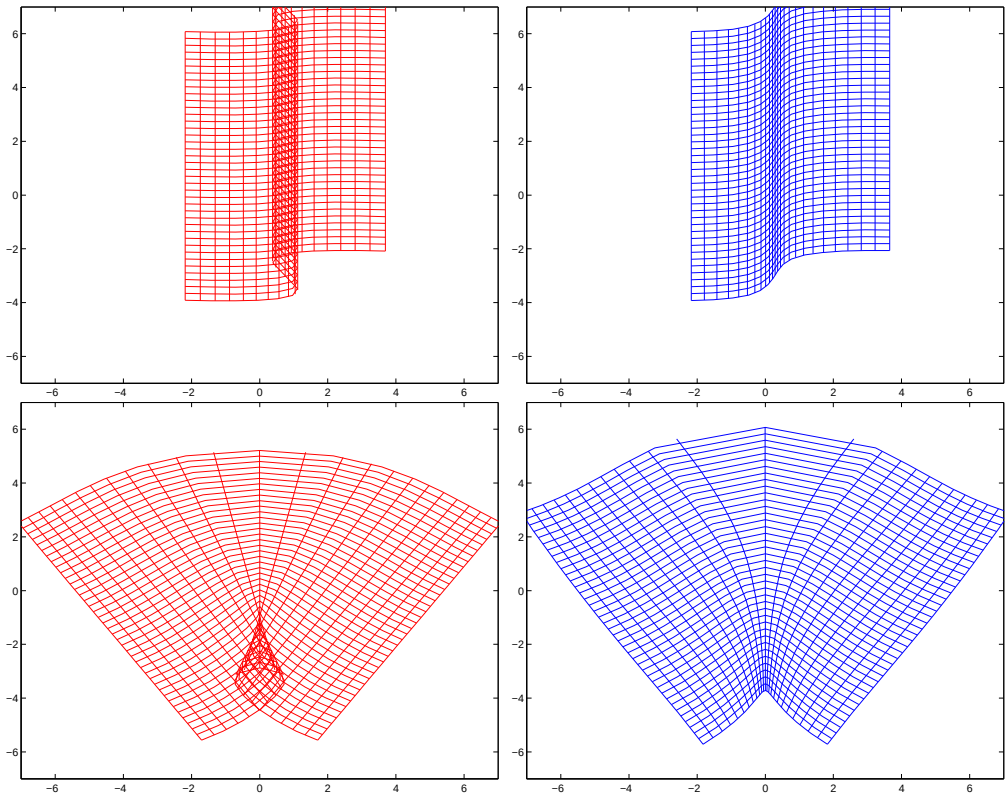

Figure 1: **Guaranteeing invertibility with infinitesimal fusion. In red:** regular grid deformed by the fusion of two affine transformations, using the direct averaging of displacements. **In blue:** regular grid deformed by the infinitesimal fusion of the transformations in the polyaffine framework. **On top:** two translations are fused. **Bottom:** two rotations of opposite angles are fused. Note how the regions of overlap disappear when infinitesimal fusion is used. The translations used were the following: $t_1 = (3,1)^T$ and $t_2 = (-1.5,3)^T$, and the two rotations of opposite angles of magnitude 0.63 radians where centered on $(-2,0)$ and $(+2,0)$. The fusion was carried out with the following weights (given here in unnormalized form): $w_i(x) = 1/(1+((x_1-c_i)/\sigma)^2)$, where $c_1 = -2$, $c_2 = +2$ and $\sigma = 5$ (smooth transition between the two components).

- **Step 2: Fusing Speed Vectors instead of Displacements.** The idea is then to average the vector fields $V_i(., s)$ according to the weight functions $w_i(x)$ to define an ODE fusing the N components. Weight functions are very important and model the influence in space of each component. They controls in particular the sharpness of transitions between the fused affine transformations. Also, they can take into account the geometry of anatomical regions of interest, as will be the case in the experimental results on 3D MRI data given in the sequel.

  The Polyaffine ODE fusing speed vectors according to weights functions is the infinitesimal analogous of (1) and writes:

$$\dot{x} = V(x, s) \stackrel{def}{=} \sum_i w_i(x) V_i(x, s). \tag{2}$$

- **Step3: Integration of the Polyaffine ODE.** In this infinitesimal framework, the value at point x_0 of the global transformation T fusing the N components is obtained via the integration of Eq. (2) between 0 and 1, with the initial condition $x(0) = x_0$. This principle is illustrated by Fig. 2.

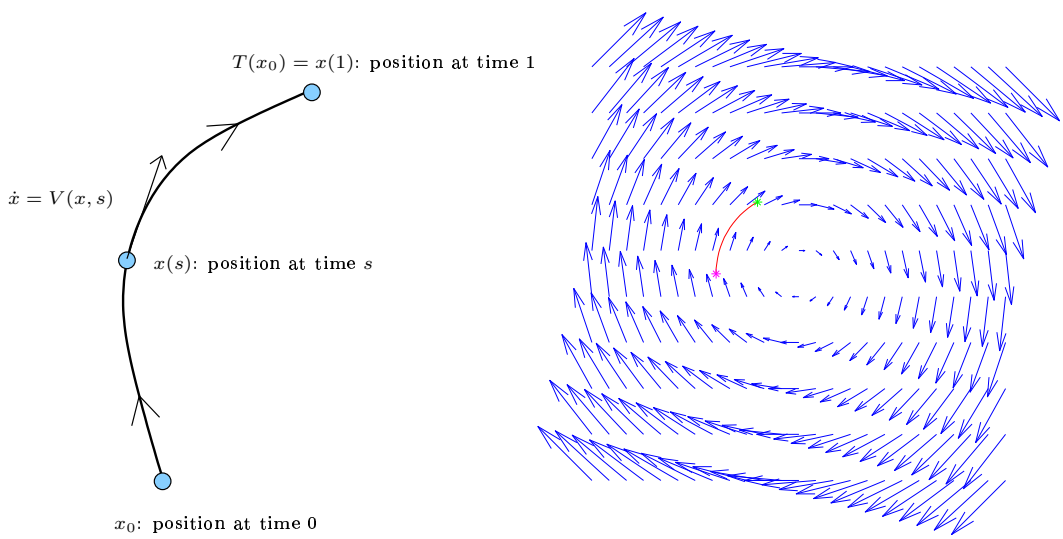

Figure 2: **Integration of speed vector fields. Left:** integration of a vector field between time 0 and time 1. The value at a point x_0 of the global transformation T is given by $x(1)$. **Right:** example of a single rotation. Speed vectors are displayed in blue. The magenta point corresponds to the initial condition and the green point is the position reached at time 3 (not time 1 so that the trajectory be longer and thus more visible).

What Speed Vectors for Affine Transformations at Step 1? Let us take an affine transformations $T = (M, t)$, where M is the linear part and t the translation. To define a family of speed vector fields consistent with T, it was proposed in [4] to rely on the *matrix logarithm* of the linear part M of T. More precisely, let L be the principal matrix logarithm of M. The family of speed vector fields $V(., s)$ we associated to T writes:

$$V(x, s) = t + L(x - s\,t) \text{ for } s \in [0, 1]. \tag{3}$$

Well-Definedness of the Principal Logarithm. One should note that using principal logarithms of the linear part of affine transformations at the first step of the polyaffine framework is not always possible.

The theoretical limitation implied by this particular choice of speed vectors is the following: principal logarithms are not always well-defined. More precisely, the principal logarithm of an invertible matrix M is well-defined if and only if the (complex) eigenvalues of M do not lie on the (closed) half-line of negative real numbers [12].

For rotations, this means quite intuitively that the amount of (local) rotation present in each of the components should be strictly below π radians in magnitude. This can be clearly seen in the domain of matrix logarithms, where this constraint corresponds to imposing that the imaginary part of eigenvalues be less then π in magnitude. Fig. 3 illustrates this general situation, which is not specific to rotations.

For general invertible linear transformations with positive determinant, the interpretation of this constraint on eigenvalues is not so clear, since rotational and non-rotational deformations are intertwined. However, one should note the closed half-line of negative number is a set of null (Lebesgue) measure in the complex plane, which indicates that very few linear transformations with positive determinant (corresponding to extremely large deformations) will not have a principal matrix logarithm. From a practical point of view, one can anyway just check whether the constraint is satisfied by computing numerically the eigenvalues of M, which only amounts to solving a third degree polynomial equation for 3D affine transformations.

In the context of medical image registration, we do not believe this restriction to be problematic, since a global affine alignment of the images to be registered is always performed first. This factors out the largest rotations and it would be very surprising from an anatomical point of view to observe very large deformations (e.g., local rotations close to 180 degrees) of an anatomical structure from one individual to another after the anatomies of these individuals have already been affinely aligned.

Heavy Computational Burden at Step 3. Now, from a practical point of view, integrating the ODE given by Eq. (2) with the speed vectors of Eq. (3) is quite computationally expensive, especially when one wishes to do this for all the points of a 3D regular grid, for example a $256 \times 256 \times 100$ grid, which is commonly in the case for T_1-weighted MR images. We will see in the rest of this section how one can drastically reduce this complexity by slightly modifying the speed vectors of Eq. (3).

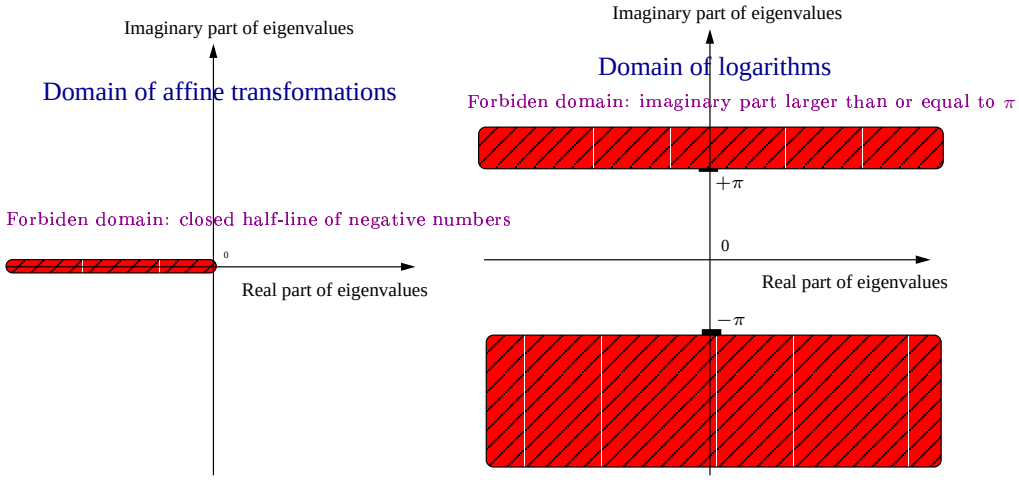

Figure 3: **Constraint imposed on affine transformations by the use of the principal matrix logarithm. Left:** only affine transformations whose (complex) eigenvalues do not lie on the (closed) half-line of negative real numbers have a principal logarithm and can be handled by our framework. Simplifying things a bit, this corresponds intuitively to imposing that (local) rotations be smaller in magnitude than π radians. This can be seen more clearly on the principal logarithms of these *admissible* affine transformations: the imaginary part of their eigenvalues must be smaller than π in magnitude. This is illustrated on the **right** on this figure. A more detailed discussion of this constraint is given in Subsection 2.1.

2.2 Simpler Speed Vectors for Affine Transformations

We will see now how one can define much simpler speed vectors for affine transformations than the ones given in Eq. (3). The basic idea is to rely on the logarithms of the transformations *themselves*, and not only on the logarithms of their *linear parts*. These logarithms can be defined in an abstract way in the context of the theory of Lie groups [13], but for more simplicity we will define them here via the representation with *homogeneous coordinates* of affine transformations. Homogeneous coordinates also provide a very convenient way to compute these logarithms in practice.

Details about our numerical implementation of the matrix logarithm are given in the Subsection A.4 of the Appendix.

Homogeneous Coordinates. Homogeneous coordinates are a classical tool in Computer Vision. They are widely used to represent any n-dimensional affine transformation T by $(n+1) \times (n+1)$ matrix, written here $\tilde{T}$. Such a representation is called by mathematicians 'faithful' (in the sense of representation theory), which means that there is no loss of information in this representation. $\tilde{T}$ takes the following form:

$$T \sim \tilde{T} \stackrel{def}{=} \begin{pmatrix} M & t \\ 0 & 1 \end{pmatrix}, \tag{4}$$

where M is the linear part of T ($n \times n$ matrix) and t its translation. In this setting, points x of the ambient space are represented by $n+1$-dimensional vectors $\tilde{x}$, adding an extra '1' after their coordinates:

$$x \sim \tilde{x} \stackrel{def}{=} \begin{pmatrix} x \\ 1 \end{pmatrix}.$$

This way, the action of the affine transformation on a point x can be obtained simply in terms of matrix multiplication and is given by $\tilde{T}.\tilde{x}$.

Principal Logarithms of Affine Transformations. Using homogeneous coordinates, the principal logarithm of the affine transformations *themselves* can be defined and computed in a simple way (i.e. without using abstract Lie groups theory).

The main point here is that the principal logarithm of an affine transformation T is represented in homogeneous coordinates by the matrix logarithm of its representation $\tilde{T}$. This matrix logarithm takes the following form:

$$\log\left(\tilde{T}\right) = \begin{pmatrix} L & v \\ 0 & 0 \end{pmatrix},$$

where log stands for the principal matrix logarithm. L is an $n \times n$ matrix and v an n-dimensional vector. Exactly as in the former subsection, L is the principal matrix logarithm of M. But v is *not* equal in general to the translation t. Actually, the difference between our novel approach and the previous one resides essentially in this v.

Interestingly, the well-definedness of the principal logarithm of an affine transformation T is equivalent to the well-definedness of the principal logarithm of its linear part M. The reason for this is that the principal matrix logarithm of an invertible matrix is well-defined if and only if the imaginary parts of its (complex) eigenvalues of M do not lie on the (closed) half-line of negative real numbers [12], as mentioned before. Because of the form taken by $\tilde{T}$ (see Eq. (4)), the spectrum of $\tilde{T}$ is exactly that of its linear part M plus an extra eigenvalue equal to 1. Hence the equivalence of the existence of both principal logarithms.

Simpler Speed Vectors for Affine Transformations at Step 1 of the Polyaffine Framework. Using now principal logarithms of affine transformations instead of the principal logarithms of their linear parts, one can now associate to an affine transformation T a simpler family of speed vector fields than in Eq. (3) in the following way:

$$V(x,s) = V(x) = v + L.x \text{ for } s \in [0,1]. \tag{5}$$

What is remarkable here is that the speed vector field at time s associated to T *does not depend on* s! To prove the consistence of this speed vector with T, let us write the associated ODE:

$$\dot{x} = v + L.x. \tag{6}$$

While the mathematical form taken by (6) might seem unfamiliar, it is much simpler (and more familiar) when expressed in homogeneous coordinates. It simply writes:

$$\dot{\tilde{x}} = \log(\tilde{T}).\tilde{x}, \tag{7}$$

which is this time a *linear* ODE. It is well-known from the theory of linear ODEs [14] that Eq. (7) can be solved analytically and that its solutions are well-defined for all time. With an initial condition x_0 at time 0, the value $x(s)$ of the unique mapping $x(.)$ satisfying Eq. (6) is given in terms of matrix exponential by:

$$\tilde{x}(s) = \exp\left(s.\log\left(\tilde{T}\right)\right).\tilde{x}_0. \tag{8}$$

By letting s be equal to 1, we thus see that our new speed vector is truly consistent with the transformation T.

The ODE of Eq. (6) is called *autonomous* (or equivalently *stationary*). Such ODEs have some very nice mathematical properties, which can be expressed in terms of *one-parameter subgroups* of transformations. The next paragraph presents these general properties, which will be detailed afterward in our particular case.

Autonomous ODEs and One-Parameter Subgroups. Let us write our ODE in the following way: $\dot{x} = V(x)$. To simplify the discussion, we will assume in the sequel that the stationary speed vector $V(x)$ is differentiable ($\mathcal{C}^1$) and that the solutions of this ODE are well-defined for all time, regardless of initial conditions.

First of all, a very important notion associated to ODE is that of its *flow*. We have the following definition:

Definition 2. *The flow associated to an autonomous ODE is the family of mappings* $\Phi(.,s) : \mathbb{R}^n \to \mathbb{R}^n$ *parameterized by a time parameter* $s \in \mathbb{R}$*, such that for a fixed* x_0*,* $s \mapsto \Phi(x_0, s)$ *is the unique solution of* $\dot{x} = V(x)$ *with initial condition* x_0 *at time* 0*.*

Intuitively, for a fixed s, the mapping $x \mapsto \Phi(x,s)$ gives the way the ambient space is deformed by the integration of the ODE during s units of time. It is always a *diffeomorphism*, i.e. a differentiable one-to-one mapping between the ambient space and itself, whose inverse is also differentiable. This invertibility property simply comes from the fact that all deformations induced by the ODE are reversible since one can go back in time by simply multiplying the speed vector $V(x)$ by -1! The smoothness of the flow comes from the smoothness of $V(x)$.

Now, let us define one-parameter subgroups:

Definition 3. *Let* $(G,.)$ *be a group (i.e., the multiplication '.' is associative and there exists a neutral element* e *and each element of* G *has a unique inverse). Then a family of elements* $g(s)$ *of* G *parameterized by* $s \in \mathbb{R}$ *is called a one-parameter subgroup of* G *if and only if:*

1. $g(0) = e$*, i.e. the neutral element of* G
2. *for all* s, t *in* $\mathbb{R}$*:* $g(s).g(t) = g(s+t)$*.*

Furthermore, if one knows how to differentiate functions valued in G *(e.g.,* G *is a Lie group), and if* $g(s)$ *is differentiable at* 0*, then* $\frac{dG}{dt}(0)$ *is called the infinitesimal generator of the subgroup.*

Interestingly, the infinitesimal generator of a one-parameter subgroup contains all the information about the subgroup, and can generate it entirely, which explains its name. See for example [15] for more details on this topic in the case of Lie groups.

We have the following result: the flow $\Phi(.,s)$ is a *one-parameter subgroup of the group of diffeomorphisms*! In other words: $\Phi(.,s) \circ \Phi(.,t) = \Phi(.,s+t)$. This implies in particular that the deformations of space given at time 1 by $\Phi(.,1)$ are twice that observed at time 0.5 via $\Phi(.,0.5)$. Last but not least, the infinitesimal generator of the flow is simply $V(x)$. This is not surprising since it is quite clear how $V(x)$ infinitesimally generates the flow: this is done precisely by integrating the associated ODE!

One-Parameter Subgroups of Affine Transformations. Now that we have recalled the general properties of autonomous ODEs, let us go back to Eq. (6). From the explicit form taken by the solutions of this ODE (see Eq. (8)), we can see that the associated flow is simply the family of affine transformations $(T^s(.))$, where T^s is the affine transformation represented by $\exp(s.\log(\tilde{T}))$, i.e. the sth power of T.

From the general properties of flows associated to autonomous ODEs, we know that the family of transformations $(T^s(.))$ is a *one-parameter subgroup* of diffeomorphisms. From this point of view, its infinitesimal generator is the vector field $V(x) = v + L.x$. From the viewpoint of the *affine group* (in contrast to diffeomorphisms), (T^s) is also a one-parameter subgroups of affine transformations, whose infinitesimal generator is this time the principal

logarithm of T. Interestingly, it can be shown with the classical tools of Lie groups theory that all continuous one-parameter subgroups of affine transformations are of this form [13].

2.3 Log-Euclidean Polyaffine Transformations

An Autonomous ODE for Polyaffine Transformations. With the speed vectors defined by Eq. (5), one can define a novel type of polyaffine transformations using the steps 2 and 3 of the Polyaffine framework. In the sequel, we will refer to these new polyaffine transformations as *Log-Euclidean polyaffine transformations* (or LEPTs). This name comes from our work on diffusion tensors [9, 16], where we have already used principal logarithms to process this other type of data.

More precisely, let (M_i, t_i) be N affine transformations, and let (L_i, v_i) be their respective principal logarithms. Then one can fuse them according to the weights $w_i(x)$ with the following ODE, which is this time *autonomous*, i.e. without any influence of the time parameter s in the second member of the equation:

$$\dot{x} = \sum_i w_i(x)\,(v_i + L_i.x)\,. \tag{9}$$

Exactly as in the case of the non-autonomous polyaffine ODE based on Eq. (3), solutions to this novel ODE are well-defined for all time s (i.e. never go infinitely far in a finite time, do not 'blow up'), regardless of the initial condition. The proof is extremely similar (although simpler, in fact) to that given in [4] for the previous polyaffine framework.

Now, we know from the general properties of stationary ODEs (which where presented above) that the flow $T(s,.)$ of this ODE forms a *one-parameter subgroup* of diffeomorphisms: $T(0,.)$ is the identity and $T(r,.) \circ T(s,.) = T(r+s,.)$.

One-Parameter Subgroups of LEPTs. Exactly like in the affine case, the ODE given by (9) defines not only a one-parameter subgroup of *diffeomorphisms*, it also yields a one-parameter subgroup of *Log-Euclidean polyaffine transformations*. More precisely, a simple change of variable $(s \mapsto \frac{s}{2})$ shows that the flow at time $\frac{1}{2}$, written here $T(\frac{1}{2},.)$, corresponds to a polyaffine transformation whose parameters are the same weights as the original ones, but where the affine transformations have been transformed into their square roots (i.e. their logarithms have been multiplied by $\frac{1}{2}$). Similarly, the flow at time s, $T(s,.)$ corresponds to a polyaffine transformations with identical weights but with the sth power of the original affine transformations.

As a consequence, $T(s,.)$ can be interpreted as the sth power of the Log-Euclidean polyaffine transformation defined by $T(1,.)$. In particular, the inverse of $T(1,.)$ (resp. its square root) is given simply by $T(-1,.)$ (resp. $T(1/2,.)$), which is the polyaffine transformation with identical weights but whose affine transformations have been inverted (resp. have been transformed into their square roots).

One should note that our previous polyaffine transformations did *not* have the same remarkable algebraic properties as Log-Euclidean polyaffine transformations. In our previous

framework, the inverse of a polyaffine transformation was not even in general a polyaffine transformation. LEPTs have very intuitive and satisfactory properties, because they are based on a fusion of speed vectors much better adapted to the algebraic properties of affine transformations than the speed vectors we previously used.

In the next Section, we will see how this specific *algebraic* property of our novel framework can be used to alleviate drastically the computational cost of Step 3 of the polyaffine framework (i.e. the cost of the integration of the polyaffine ODE).

Affine-Invariance of LEPTs. Contrary to the previous polyaffine framework, our novel Log-Euclidean framework has another sound mathematical property: *affine-invariance*. This means the Log-Euclidean polyaffine fusion of affine transformations is invariant with respect to any affine change of coordinate system. This type of fusion is therefore *intrinsic*: it truly is a fusion between *geometric transformations* since it does not depend at all on the arbitrary choice of coordinate system chosen to represent them.

To see why this is so, let us see how the various ingredients of our framework are affected by a change of coordinate system induced by an affine transformation A. In homogeneous coordinates, these changes are the following:

- a point $\tilde{x}$ becomes $\tilde{A}.\tilde{x}$
- a weight function $\tilde{x} \mapsto w_i(\tilde{x})$ becomes $\tilde{y} \mapsto w_i(\tilde{A}^{-1}.\tilde{y})$
- an affine transformation $\tilde{T}_i$ becomes $\tilde{A}.\tilde{T}_i.\tilde{A}^{-1}$.

In our new coordinate system, the Log-Euclidean polyaffine ODE writes in homogeneous coordinates:

$$\dot{\tilde{y}} = \sum_i w_i(\tilde{A}^{-1}.\tilde{y}) \log\left(\tilde{A}.\tilde{T}_i.\tilde{A}^{-1}\right).\tilde{y}. \tag{10}$$

Then, using the property $\log\left(\tilde{A}.\tilde{T}_i.\tilde{A}^{-1}\right) = \tilde{A}.\log\left(\tilde{T}_i\right).\tilde{A}^{-1}$, the simple change of variable $\tilde{y} \mapsto \tilde{A}.\tilde{x}$ shows that a mapping $s \mapsto \tilde{x}(s)$ is a solution of the Log-Euclidean polyaffine ODE (9) if and only if $s \mapsto \tilde{A}.\tilde{x}(s)$ is a solution of (10). This means that the solutions of the Log-Euclidean polyaffine ODE in the new coordinate system are exactly the same as in the original coordinate system: our novel polyaffine framework is therefore not influenced by the choice of a coordinate system. Our previous polyaffine framework does not have this property, because it does not take sufficiently into account the algebraic properties of affine transformations.

Another Reason Why our Novel Polyaffine Framework Is Called Log-Euclidean. In the special case where all the weight functions $w_i(x)$ do not depend on x, the Log-Euclidean polyaffine fusion of the affine transformations T_i simply yields an affine transformations T, which is given by the following *Log-Euclidean mean*:

$$T = \exp(\sum_i w_i \log(T_i)).$$

This is another reason why we refer to our novel polyaffine framework as Log-Euclidean. Indeed, the use of a generalization to affine transformations of our Log-Euclidean framework for tensors is implicit in this novel framework. More details on the affine Log-Euclidean framework are presented in the Appendix of this article.

Synthetic Examples. Examples of 2D LEPTs are shown in Figs. 4, 5 and 6. In these examples, one can see how antagonistic affine transformations (i.e. transformations whose direct fusion results in local singularities) can be globally fused into a regular and invertible polyaffine transformation.

Extreme Closeness to Previous Polyaffine Framework. Interestingly, we have observed in our experiments that the Log-Euclidean and the previous polyaffine frameworks provide extremely similar results. Fig. 7 illustrates the striking closeness between both frameworks. Notable differences only appear when very large deformations are fused.

Therefore, the advantage of our Log-Euclidean polyaffine framework over the previous one does not reside in the quality of its results, which are very close to those of the previous one. Rather, it resides in its much better and more intuitive mathematical properties, which allow for much faster computations, as will be shown in the next Section. This situation is somehow comparable to the closeness between the affine-invariant and Log-Euclidean Riemannian frameworks used to process diffusion tensors [9]. They also yield very similar results, but in a simpler and faster way in the Log-Euclidean case.

Limitation of this Approach. This novel approach is based on the notion of principal logarithm of an affine transformation. As pointed out before, this logarithm is well-defined if and only if the principal logarithm of the linear part of this affine transformation is well-defined. As a consequence, our new approach has exactly the same limitation as the previous one: to simplify, the amount of rotation in the affine transformation considered should not be too close to 180 degrees. For more precisions on this topic, please refer to Subsection 2.1.

3 Fast Polyaffine Transform

As will be shown in this section, the specific *algebraic* properties of the Log-Euclidean polyaffine framework allow for fast computations of LEPTs. In particular, we propose an efficient algorithm to evaluate a Log-Euclidean polyaffine transformations on a regular grid. If N is the number of intermediate points chosen to discretize the continuous trajectory of each point, we present here an algorithm only requiring $\log_2(N)$ steps to integrate our autonomous polyaffine ODE, provided that the trajectories of all the points of the regular grid are computed simultaneously. This drastic drop in complexity is somehow comparable to that achieved by the 'Fast Fourier Transform' in its domain.

Surprisingly, the key to this approach lies in the generalization to the non-linear case of a classical method used in the linear case, one that is widely used to compute numerically the exponential of a square matrix.

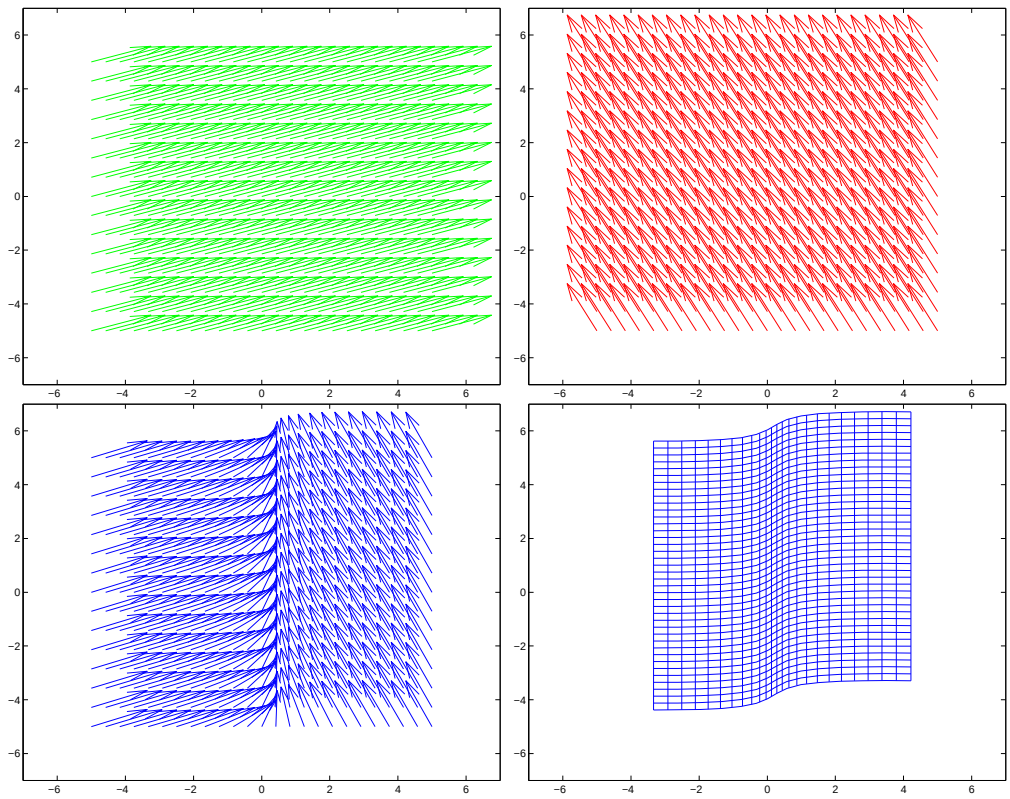

Figure 4: **Fusing speed vectors of two translations. Red and Green:** Log-Euclidean polyaffine speed vectors (with the novel framework) of two affine transformations to be fused. **In blue:** on the left, fused speed vectors, and on the right, regular grid deformed after integration of the autonomous ODE. Note how the antagonism between the two translations results in a progressive compression along the boundary between the two components. The fusion was carried out with two functions of the first coordinate as weights, as in Fig. 1.

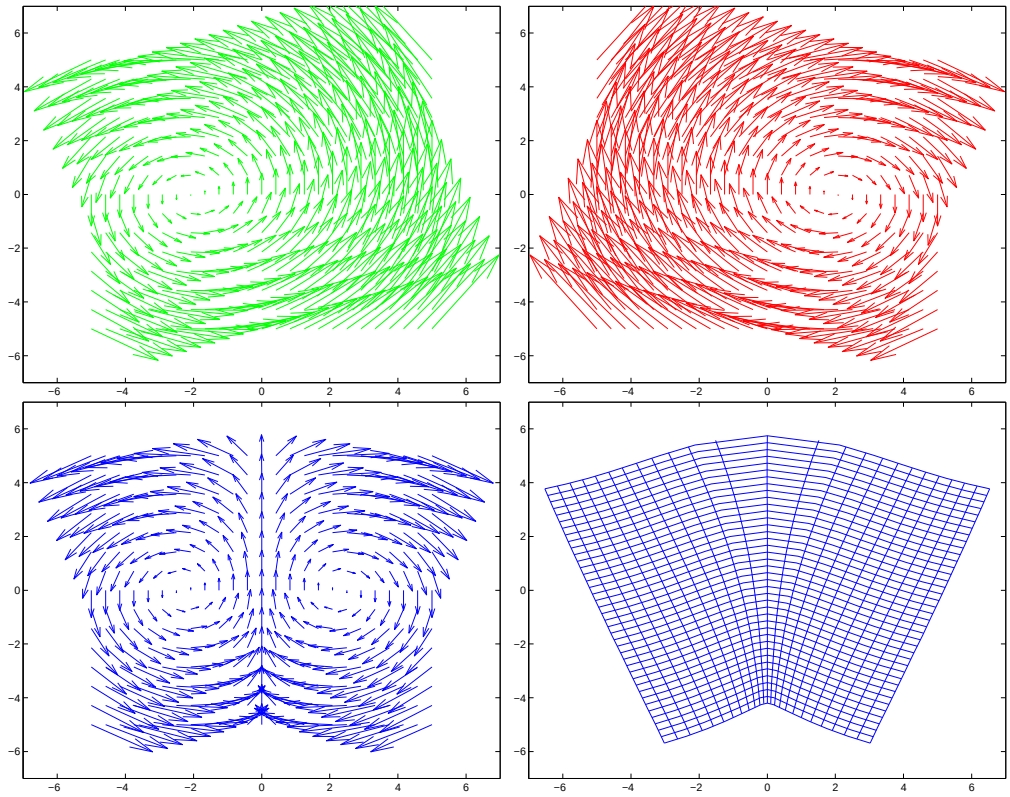

Figure 5: **Fusing speed vectors of two rotations. Red and Green:** Log-Euclidean polyaffine speed vectors (with the novel framework) of two affine transformations to be fused. **In blue:** on the left, fused speed vectors, and on the right, regular grid deformed after integration of the autonomous ODE. Note how regular and invertible the fused polyaffine transformation, however antagonistic the two fused rotations are locally. The fusion was carried out with two radial functions of the first coordinate as weights, as in Fig. 1.

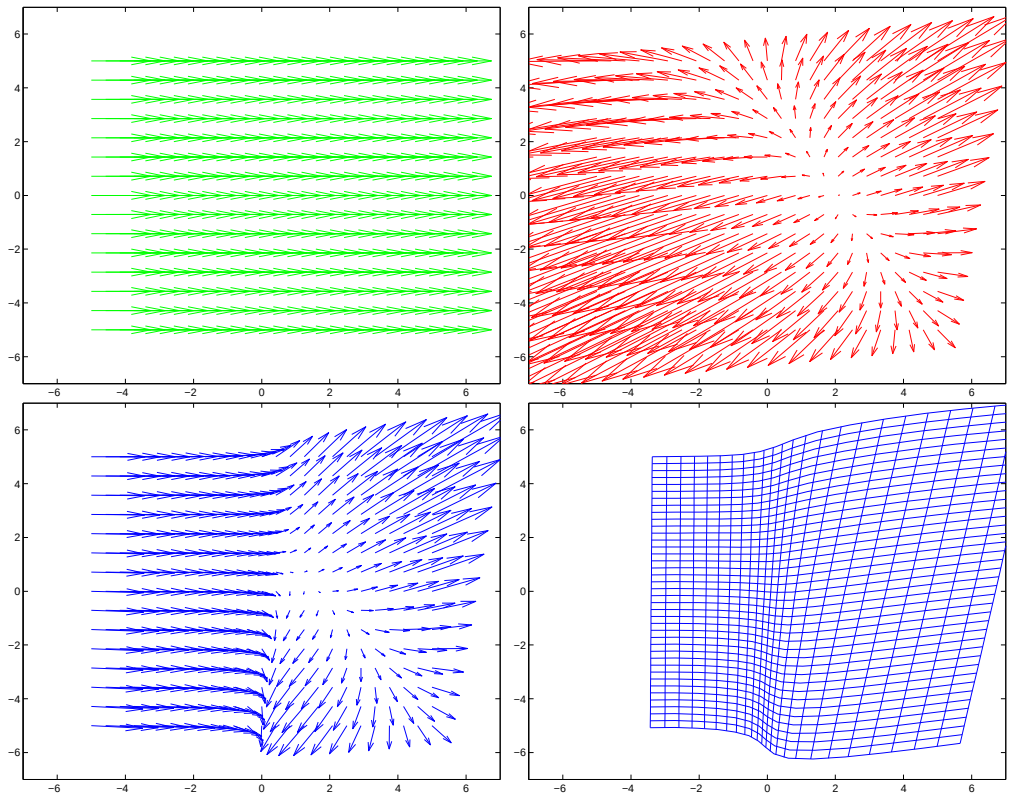

Figure 6: **Fusing speed vectors of a translation and an anisotropic swelling. Red and Green:** Log-Euclidean polyaffine speed vectors (with the novel framework) of two affine transformations to be fused. **In blue:** on the left, fused speed vectors, and on the right, regular grid deformed after integration of the autonomous ODE. Again, note how locally antagonistic displacements are invertibly fused, resulting in compressions or swelling at the boundary between the two components. The fusion was carried out with two radial functions of the first coordinate as weights, as in Fig. 1.

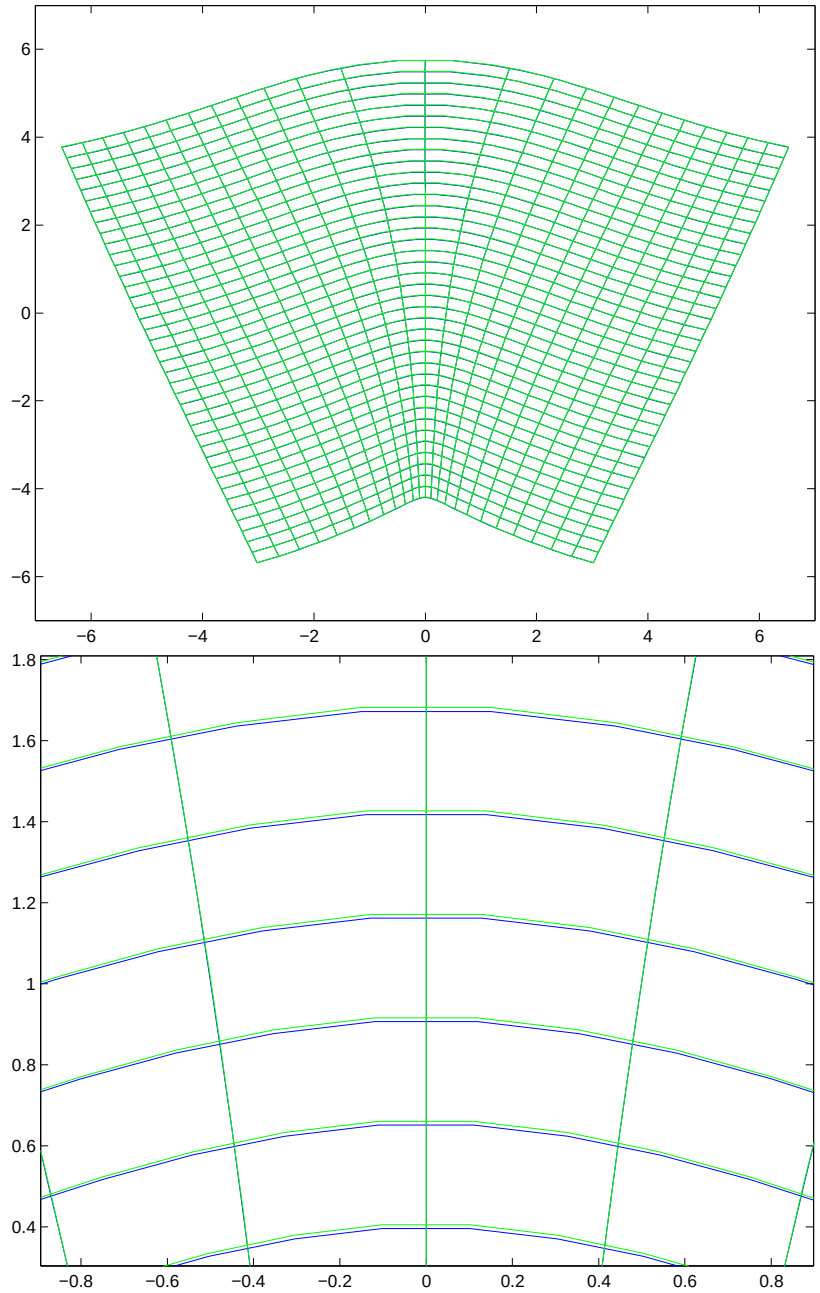

Figure 7: **Extreme closeness between the Log-Euclidean polyaffine framework and the previous polyaffine framework.** Superimposed deformed grids in both cases. **Top:** whole grid and **bottom:** close-up. The blue grid corresponds to Log-Euclidean results and the green one to the previous framework (on top of the blue grid), in the case of the fusion of two rotations presented in Fig. 5.

3.1 Matrix Exponential and the 'Scaling and Squaring' Method

The matrix exponential of a square matrix can be computed numerically in a large number of ways, with more or less efficiency [17]. One of the most popular of these numerical recipes is called the 'Scaling and Squaring' method, which is for example used by Matlab ™ to compute matrix exponentials [18]. Fundamentally, this method is very efficient because it takes advantage of the very specific algebraic properties of matrix exponential, which are in fact quite simple, as we shall see now. For any square matrix M, we have:

$$\exp(M) = \exp\left(\frac{M}{2}\right).\exp\left(\frac{M}{2}\right) = \exp\left(\frac{M}{2}\right)^2. \tag{11}$$

This comes from the fact that M commutes with itself in the sense of matrix multiplication. Iterating this equality, we get for any positive integer N:

$$\exp(M) = \exp\left(\frac{M}{2^N}\right)^{2^N}, \tag{12}$$

Then, the key idea is to realize that the matrix exponential is much simpler to compute for matrices *close to zero*. In this situation, one can for example use just a few terms of the infinite series of exponential, since high-order terms will be completely negligible. An even better idea is to use Padé approximants, which provide excellent approximations by rational fractions of the exponential around zero with very few terms. For more (and recent) details on this topic, see [18].

The 'Scaling and Squaring' Method for computing the matrix exponential of a square matrix M can be sketched as follows:

1. **Scaling step**: divide M by a factor 2^N, so that $\frac{M}{2^N}$ is close enough to zero (according to some criterion based on the level of accuracy desired: see [18] for more details).

2. **Exponentiation step**: $\exp\left(\frac{M}{2^N}\right)$ is computed with a high precision using for example a Padé approximant.

3. **Squaring step**: using Eq. (12), $\exp\left(\frac{M}{2^N}\right)$ is squared N times (only N matrix multiplications are required.) to obtain a very accurate estimation of $\exp(M)$.

In the rest of this Section, we will see how one can generalize this method to compute with an excellent accuracy polyaffine transformations based on autonomous ODEs.

3.2 A 'Scaling and Squaring' Method for LEPTs

Goal of the Method. We would like to compute efficiently and with a good accuracy the values of a Log-Euclidean polyaffine transformation at the vertices of a regular n-dimensional (well, 2D or 3D in practice) grid. The method described below will be referred to as the 'Fast Polyaffine Transform' (or FPT) in the rest of this paper.

Algebraic Properties of Log-Euclidean Polyaffine Transformations Revisited. Let $T(s,.)$ be the flow associated to the autonomous polyaffine ODE (9), as in Subsection 2.3. As mentioned before, this flow is a one-parameter subgroup of LEPTs:

$$T(0,.) = Id \quad \text{and for all } r,s\text{: } T(r,.) \circ T(s,.) = T(r+s,.).$$

As a consequence, Exactly as Eq. (11) for the matrix exponential, we obtain for $r = s = \frac{1}{2}$:

$$T(1,.) = T\left(\frac{1}{2},.\right) \circ T\left(\frac{1}{2},.\right) = T\left(\frac{1}{2},.\right)^2.$$

Iterating this equality, we get for any positive integer N:

$$T(1,.) = T\left(\frac{1}{2^N},.\right)^{2^N}. \tag{13}$$

Intuitively, Eq. (13) means that what the deformation observed at time 1 results of 2^N times the repetition of the small deformations observed at time $\frac{1}{2^N}$. The total deformation is entirely determined by the initial (and small) deformations occurring just at the beginning of the integration of our ODE (which is a well-known and general phenomenon with autonomous ODEs).

Fast Polyaffine Transform. We can now generalize the 'Scaling and Squaring' Method to the Log-Euclidean polyaffine case. This method, called the 'Fast Polyaffine Transform', follows the usual three steps:

1. **Scaling step**: divide $V(x)$ (the field of speed vectors) by a factor 2^N, so that $\frac{V(x)}{2^N}$ is close enough to zero (according to the level of accuracy desired).
2. **Exponentiation step**: $T\left(\frac{1}{2^N},.\right)$ is computed using an adequate numerical scheme.
3. **Squaring step**: using Eq. (13), $T\left(\frac{1}{2^N},.\right)$ is squared N times (in the sense of the composition of transformations; only N compositions are required.) to obtain an accurate estimation of $T(1,.)$, i.e. of the polyaffine transformation to be computed.

From a practical (or numerical) point of view, two points remain to be clarified. First what numerical scheme can be used to compute $T\left(\frac{1}{2^N},.\right)$ with a good precision during the 'exponentiation step'? Second, how should the composition (which is the multiplication operator for transformations) be performed during the 'squaring step'?

Exponentiation Step. Exactly as in the matrix exponential case, integrating an ODE during a very short interval of time (short with respect to the smoothness of the solution) is quite easy. We can use any of the methods classically used to integrate ODEs during short periods of times, like explicit schemes or Runge-Kutta methods, which are based on various

uses of the Taylor development to compute solutions of ODEs (see [19] for more details on these methods).

The simplest of these schemes is undoubtedly the first-order explicit scheme. In our case, it simply consists in computing the following value:

First Order Explicit Exponentiation Scheme (E.S):

$$T\left(\frac{1}{2^N}, x\right)_{\text{E.S.}} \stackrel{def}{=} x + \frac{1}{2^N}.V(x).$$

Generalizing the ideas already developed in [4] for the previous polyaffine framework, we can also use a second-order scheme which takes into account the affine nature of all components, and which is *exact* in the case of a single component. We will refer to this scheme as the *affine exponentiation scheme* in the following. It writes:

Second Order Affine Exponentiation Scheme (A.S):

$$T\left(\frac{1}{2^N}, x\right)_{\text{A.S.}} \stackrel{def}{=} \sum_{i=1}^{N} w_i(x).T_i^{\frac{1}{2^N}}(x),$$

where $T_i^{\frac{1}{2^N}}$ is the $2^{N\text{th}}$ root of the affine transformation T_i. We will see later in this Section that the accuracy of this numerical scheme adapted to the affine nature of the components is slightly better than that of the explicit scheme.

Composing Discrete Transformations. In this work, we are evaluating our transformation at a *finite* number of vertices of a regular grid. Practically, one has to resort to some kind of interpolation/extrapolation technique to calculate the value of such a transformation at *any* spatial position. Numerous possibilities exist in this domain, such a nearest-neighbor interpolation, bi- or tri-linear interpolation, continuous representations via the use of a basis of smooth functions like wavelets, radial basis functions... In the following, we use bi- and tri-linear interpolations, which are simple tools guaranteeing a continuous interpolation of our transformation. The best type of interpolation technique for the purposes of our Fast Polyaffine Transform remains to be determined and will be the subject of future work.

Algorithmic Complexity. Note that to compute polyaffine transformations using the FPT, the weight functions need only be evaluated *once* per voxel, instead of evaluating them at *every step* of the integration of the ODE, as was done in [4]. In the case where weight functions are stored in the computer memory as 3D scalar images, this offers the opportunity of removing them from the computer RAM after the exponentiation step. This could be particularly useful when a large number of affine components are used on high-resolution images.

Furthermore, the equivalent of 2^N intermediary points is achieved in only N steps, in contrast with the 2^N steps required by a traditional method. Last but not least, after the

$2^{N\text{th}}$ root has been computed, only N compositions between transformations need to be computed, which is an operation based on interpolation techniques and therefore not very computationally expensive. Let N_{vox} be the number of voxels and let N_{pts} be the number of intermediary points chosen to integrate the polyaffine ODE. The complexity of our new algorithm is thus $O(N_{\text{vox}}.\log_2(N_{\text{pts}}))$, whereas the complexity of traditional methods of integration of this ODE is $O(N_{\text{vox}}.N_{\text{pts}})$.

Computing the Inverse of a Polyaffine Transformation. As pointed out in Subsection 2.3, in our new framework the inverse of a polyaffine transformation is simply the polyaffine transformation associated with the opposite vector field (i.e. the polyaffine transformation with the same weights but inverted affine components). As a consequence, the inverse of a polyaffine transformation can be also computed using the Fast Polyaffine Transform. Actually, any power (square root, etc.) a polyaffine transformation can be computed this way.

3.3 2D Synthetic Experiments

Throughout this results Section, we measure the accuracy of our results by computing the relative difference of the results with respect to accurate estimations of the real (continuous) transformations. These reference transformations are obtained by a classical integration (i.e., a fixed time step was used) of the Log-Euclidean polyaffine ODE for each of the pixels of the grid, using a small time step: 2^{-8}.

One should note that several parameters influence the accuracy of the results:

- the scaling 2^N
- the geometry of the regular grid
- the interpolation method
- the extrapolation method.

Thus, compared to the classical estimation method with a fixed time step, our fast transform possesses three new sources of numerical errors: the geometry of the regular grid (the transformation is evaluated only at a finite number of points, the more points the more precise the result will be), the interpolation method and the fact that regardless of the extrapolation method, some part of the information about what happens outside of the regular grid is lost. It is therefore important to check that the accuracy of the results obtained with the FPT are not marred by these new sources of error.

A Typical FPT. Figs. 8 and 9 display the results of a typical Fast Polyaffine Transform, using two rotations of opposite angles, and a scaling of 2^6 (and therefore 6 squarings). The regular grid chosen to sample the transformation is of 50×40 pixels. The affine exponentiation scheme is used.

On average, the results are quite good: the average relative error is approximately equal to 0.6%. However, much higher errors (around 11%) are obtained at the boundary, which comes from the fact that the bi-linear interpolation we use here does not take into account the rotational behavior of the transformation outside of the grid.

Using Bounding Boxes to Correct Boundary Effects. The numerical errors stemming from the loss of information at the boundary of the regular grid can be drastically reduced for example by enlarging the regular grid used. A simple idea consists in adding to the regular grid some extra points so that it contains the points of boundary deformed by Euclidean fusion of the affine components. This is illustrated by Fig. 10.

Fig. 11 presents the accuracy of the results given by the FPT, this time using a regular grid extended in the way described just above. This time, errors are much lower: the relative accuracy of the resulting estimation of the polyaffine transformation is on average of 0.21% (instead of approximately 0.6% without an enlarged grid), and the maximal relative error is below 3.2% (instead of 11% without an enlarged grid). This simple and efficient technique, which drastically reduces the effect of boundary effects on the FPT, is used systematically in the rest of this article.

Influence of Scaling. What scaling should be chosen when the FPT is used? Of course, this depends on the quantity of high frequencies present in the polyaffine transformations. The more sharp changes, the smaller the scaling should be and the finer the sampling grid should also be.

Fig. 12 displays the performance in accuracy of the FPT when the number of iterations N varies. In this experiment, we use the same fusion of rotations as in the previous experiment. In this case, the optimal scaling is 2^5. Larger scalings do not result in better accuracy, essentially because of the missing information at the boundary.

We observed in the experiments on real 3D medical images described in the sequel of this article that even much smaller scalings (typically 2^3 or 2^2) could be used without sacrificing the accuracy of the result. In short, introducing even a small number of intermediary points substantially regularizes the fused transformation with respect to the direct fusion, since this suffices to remove singularities in practice. Using more intermediary points, i.e. 5 or more squarings, offers the possibilities to be very close to the ideal polyaffine transformation, which provides a simple way to compute the inverse of the fused transformation with an excellent accuracy, as will be shown in this subsection.

Last but not least, one should also note from Fig. 12 that our Fast Polyaffine Transform is very *stable*: using unnecessary iterations (or equivalently a very large scaling) does not result in numerical instabilities. The result is mostly independent of N for $N > 6$.

Comparison between Numerical Schemes. Here, we compare the explicit affine exponentiation schemes. We perform this comparison on our three favorite examples: the fusion of two rotations, the fusion of two antagonistic translations as in Fig. 4, and the fusion

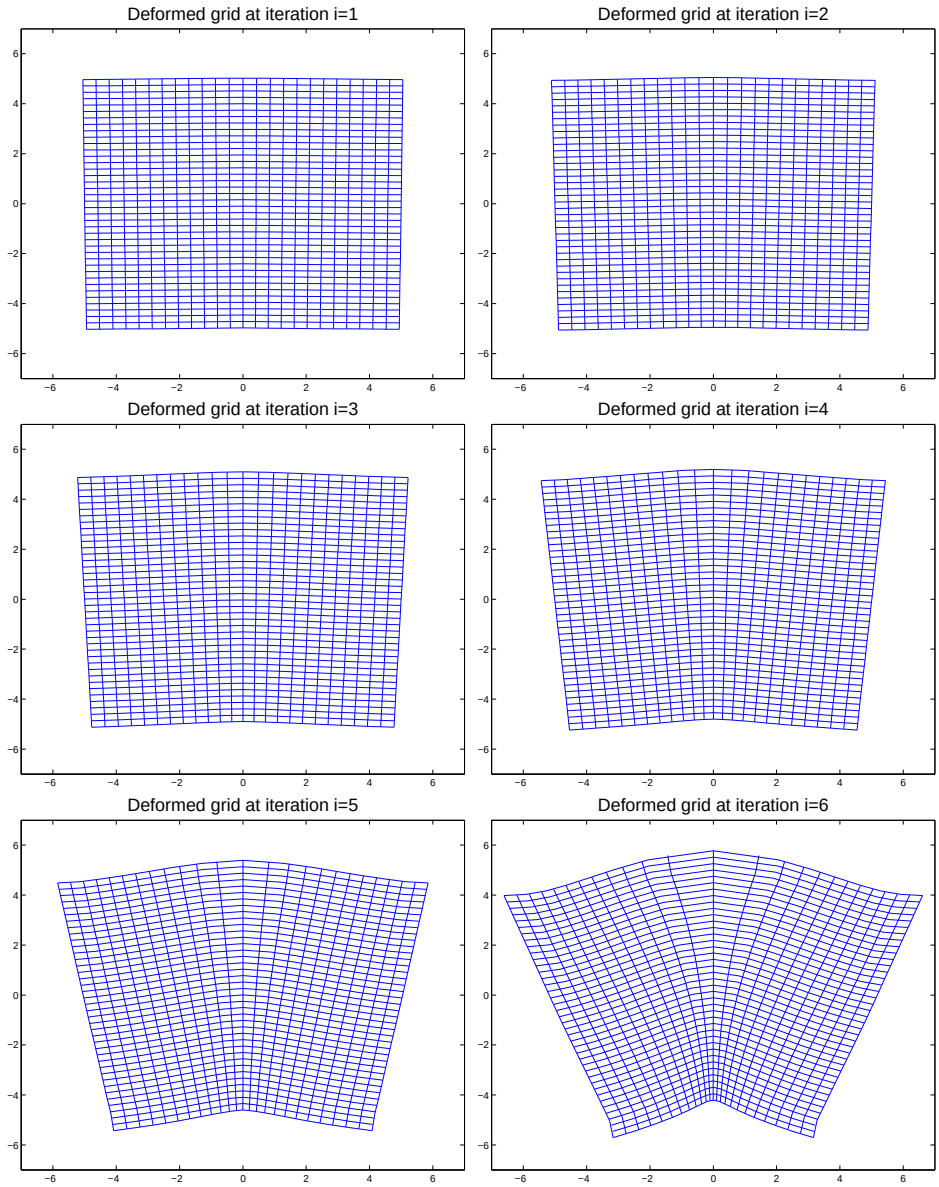

Figure 8: **Fast polyaffine transform for two rotations.** A scaling factor of 2^6 was used in this experiment, and there are therefore 6 squaring steps. Note how the deformation is initially very small, and increases exponentially. The accuracy of the FPT results was measured with respect to the results given by a classical integration (voxel by voxel) of the polyaffine ODE with 2^8 intermediary points. The relative error of the resulting estimation of the polyaffine transformation is below 0.6% on average and the maximal relative error, as expected, is made at the boundary and is below 11%.

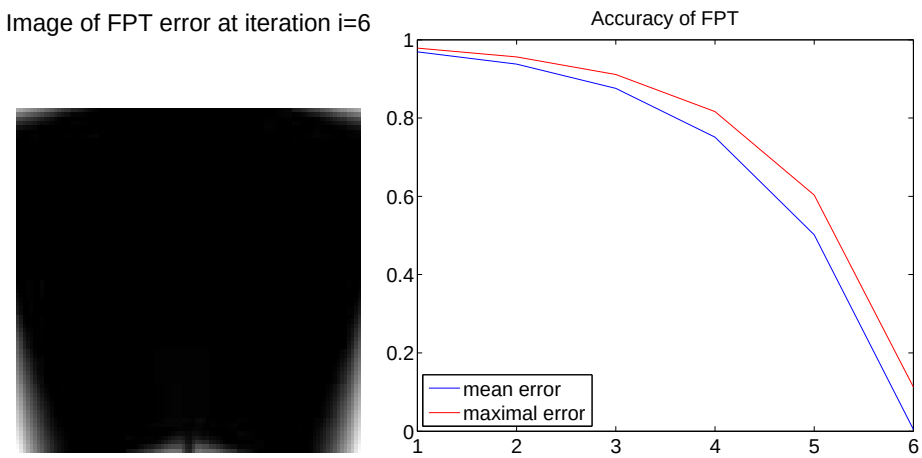

Figure 9: **Fast polyaffine transform for two rotations: errors localization ans evolution. Left:** the errors at the vertices of our 50×40 regular grid are displayed as an image, after a FPT with 6 squarings. Note how the maximal relative errors are concentrated on the boundary of our grid. This is due to the inaccuracy of our extrapolation technique, which is only bi-linear and does not deal very precisely with the affine nature of the polyaffine transformation. **Right:** the evolution of errors along squarings is displayed. The relative error of the resulting estimation of the polyaffine transformation is below 0.6% on average and the maximal relative error is below 11%.

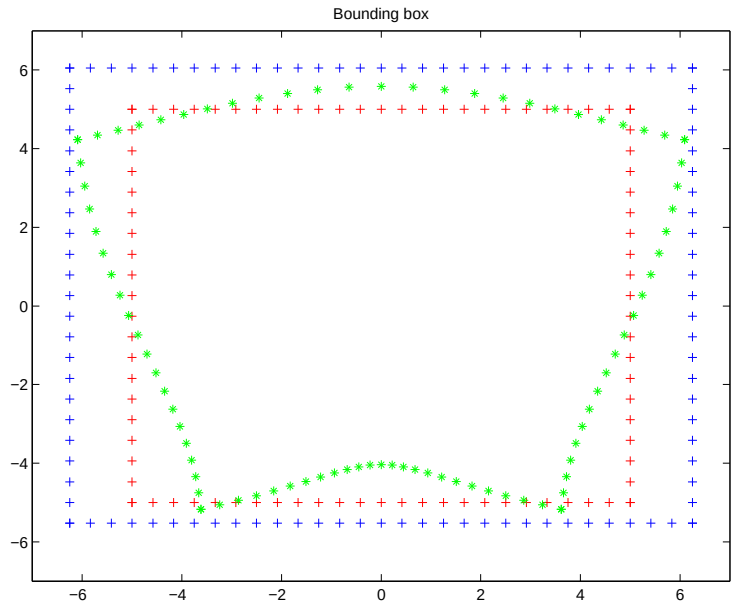

Figure 10: **Enlarging the original regular grid with a bounding box. In red:** the original regular grid used to sample the Log-Euclidean polyaffine fusion between two rotations. **In green:** grid deformed by direct fusion of the two rotations, which can be computed at a very low computational cost. **In blue:** regular grid extended so that it now contains the green points. This enlarging procedure considerably reduces the impact on the Fast Polyaffine Transform of the loss of information beyond the boundaries of the regular grid , as shown in this Section.

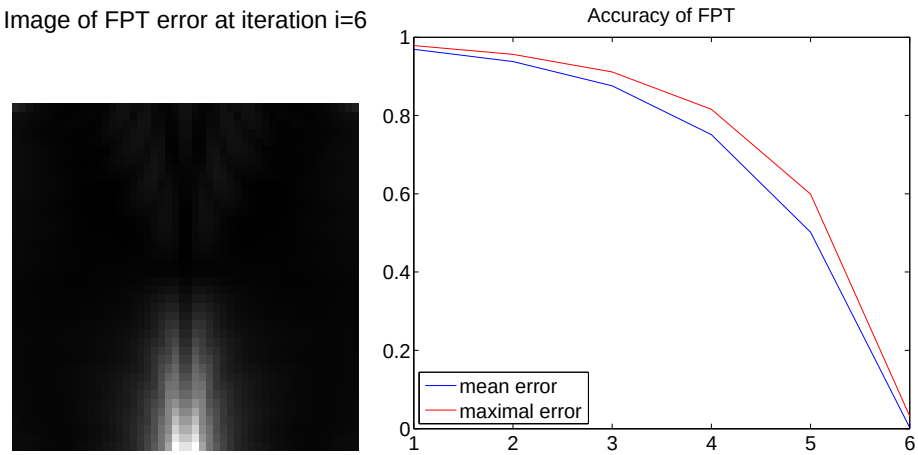

Figure 11: **Using an enlarged sampling grid: impact on errors localization and evolution for the fast polyaffine transform for two rotations. Left:** the errors at the vertices of our 50×40 regular grid are displayed as an image, after a FPT with 6 squarings. Note how the maximal errors are concentrated this time on the region of highest compression. **Right:** the evolution of errors along squarings is displayed. This time, errors are much lower: the relative error of the resulting estimation of the polyaffine transformation is on average below 0.21% (instead of below 0.6% without an enlarged grid), and the maximal relative error is below 3.2% (instead of 11% wihtout an enlarged grid).

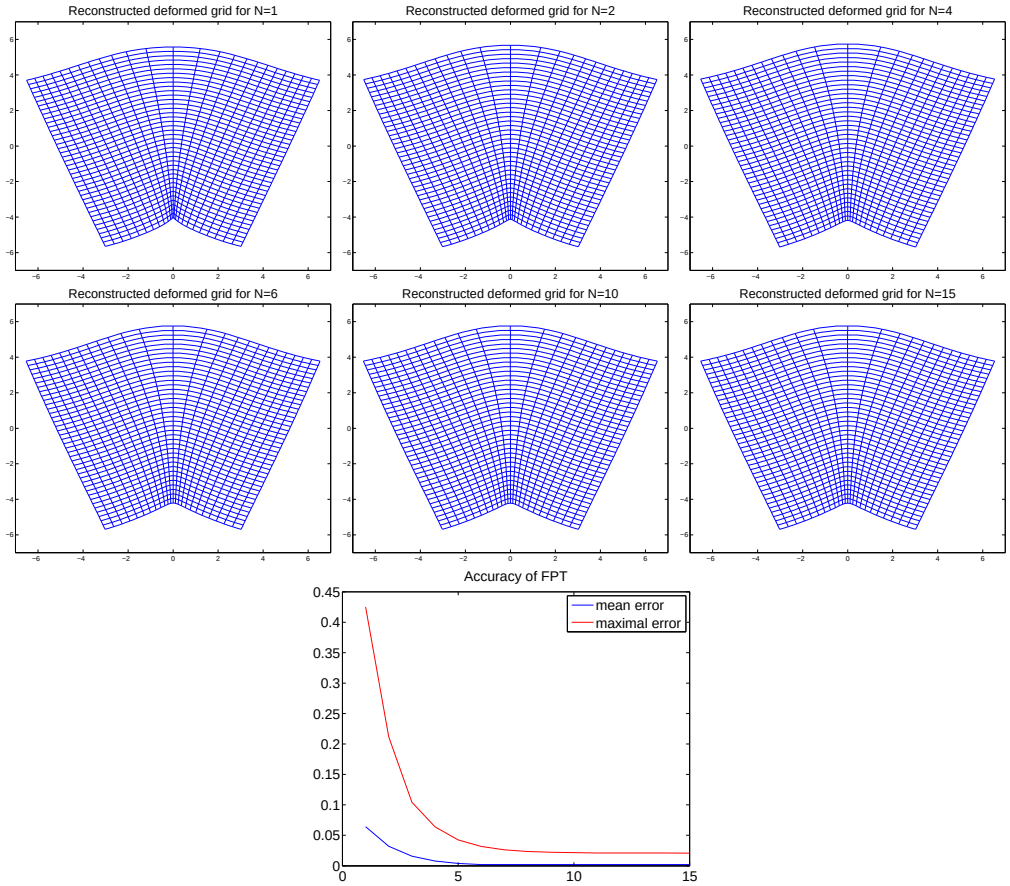

Figure 12: **Fast polyaffine transform for two rotations: influence of scaling. Above:** regular grids deformed by polyaffine transformations obtained with the FPT using different values of the scaling factor. The scaling factors are the following: 2^1, 2^2, 2^4, 2^6, 2^{10} and 2^{15}. Note how close the results are when the number of squarings N is larger or equal to 7. **Below:** accuracy of the estimations when N varies. The results are extremely *stable* for $N > 6$: it is unnecessary to use larger scalings in the example considered here. However, remarkably, using larger scalings *does not change* the results: our FPT is very stable. The relative error of the resulting estimation of the polyaffine transformation converges toward 0.2% on average and the maximal relative error converges toward 2% for large Ns. The residual maximal error is essentially due to the sampling of the transformation on a grid and the use of an interpolation method between the points of the grid, since an extended grid is used to drastically reduce errors at the boundary of the grid.

between a translation and an anisotropic swelling as in Fig. 6. The accuracy of the FPT using both numerical schemes is compared in all three cases. Fig. 13 shows the results.

Both numerical schemes make the FPT converge toward the same accuracy as the number of squarings increases, but the convergence is slightly faster in the affine exponentiation case: the average error is 40% smaller in the affine case for scalings smaller than 2^6. Interestingly, the two numerical schemes are identical for the fusion of the two translations, because the linear parts of these two affine translations are equal to the identity.

Inverting Polyaffine Transformations with the FPT. As pointed out previously, in our novel framework, the inverse of a polyaffine transformation is simply (and quite intuitively) the polyaffine transformation with the same weights and with inverted affine components. This inverse can also be computed using the Fast Polyaffine Transform, and in this experiment we tested the accuracy of the inversion obtained this way. The affine exponentiation scheme was used for exponentiation along with a 50×40 grid.

Fig. 14 presents with deformed grids the evolution of the accuracy of inversion when the number of squarings varies, in our example of fusion between two rotations. Fig. 15 presents the quantitative results in the three cases of fusion used in the previous experiment. We thus see that an excellent quality of inversion can be achieved using a small number of squarings, typically 6.

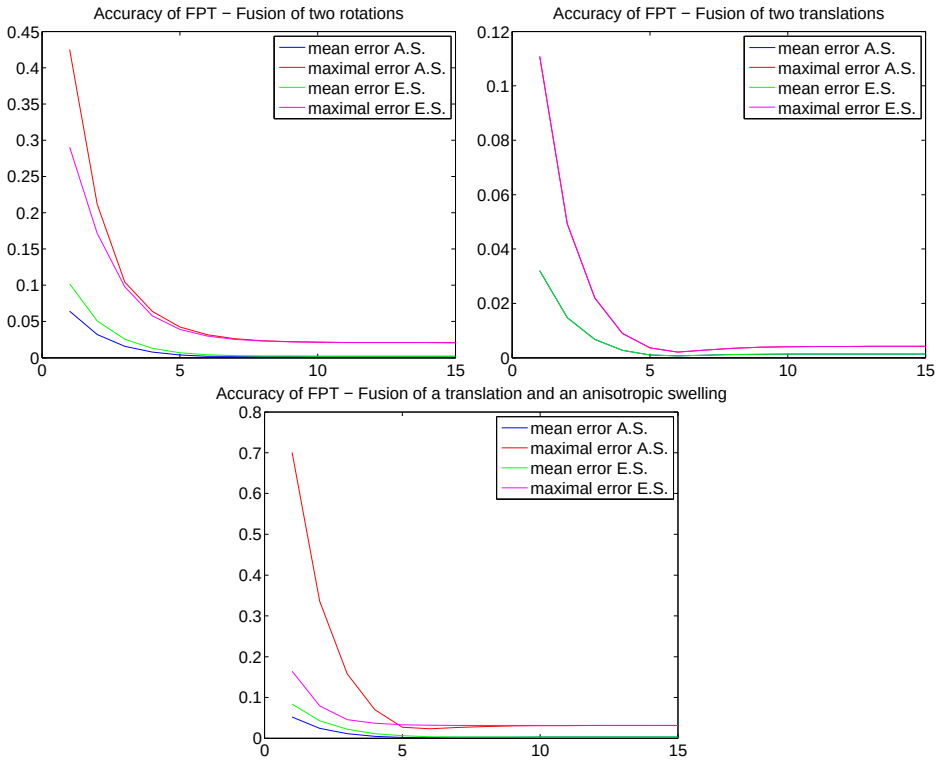

Figure 13: **Comparison between numerical schemes. From left to right and then from top to bottom:** fusion between two rotations, two translation and finally a translation and an anisotropic swelling. A.S. stands for 'affine exponentiation scheme' and E.S. for 'explicit exponentiation scheme'. Interestingly, the two numerical schemes are identical for the fusion of the two translations, because the linear parts of these two affine translations are equal to the identity. Both numerical schemes make the FPT converge toward the same accuracy as the number of squarings increases, but the convergence is substantially faster in the case of the affine exponentiation scheme: in the two cases where the schemes yield different results, the average relative error is 40% smaller in the affine case for scalings smaller than 2^6.

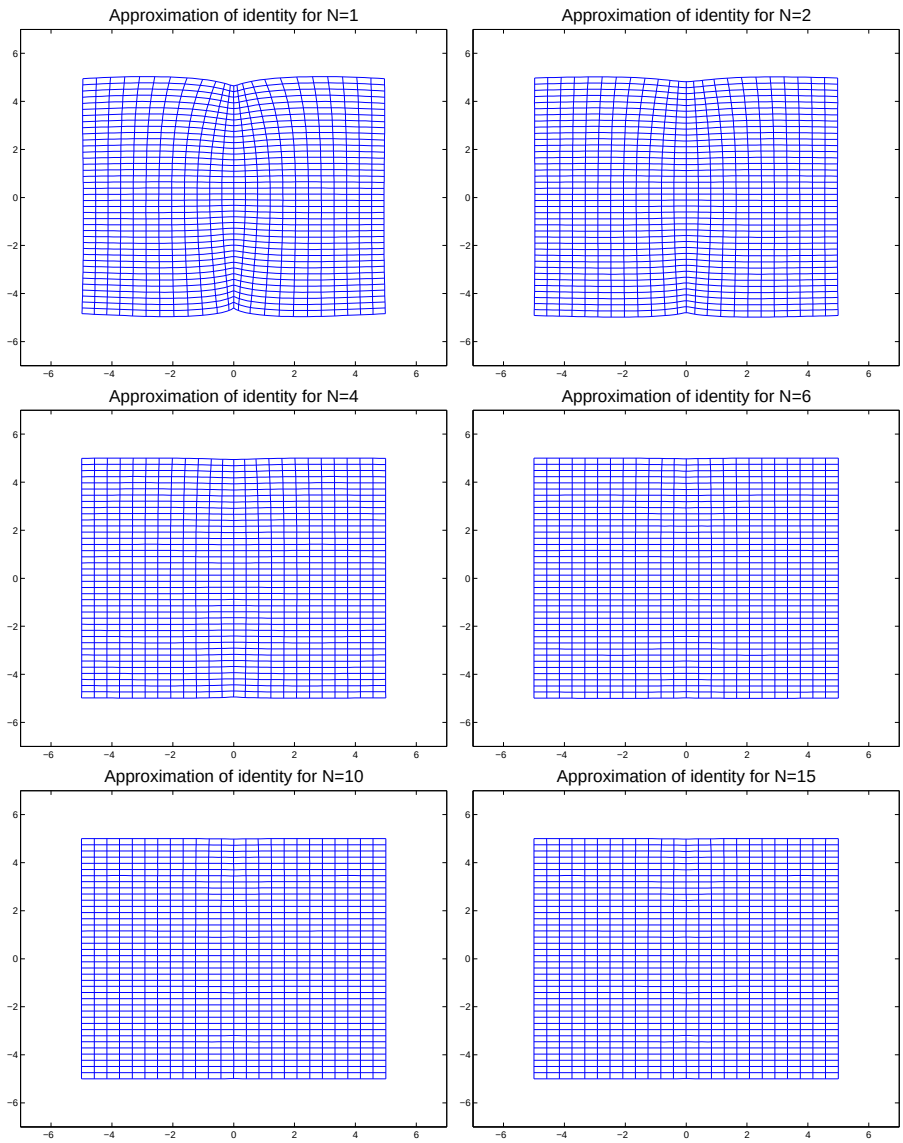

Figure 14: **Inverting a polyaffine transformation with the FPT. From left to right and then from top to bottom:** our regular grid is deformed by the composition between the FPT of the fusion between two rotations and the FPT of its inverse, for different numbers of squarings N. One can see that an excellent accuracy of inversion is already achieved with 6 squarings.

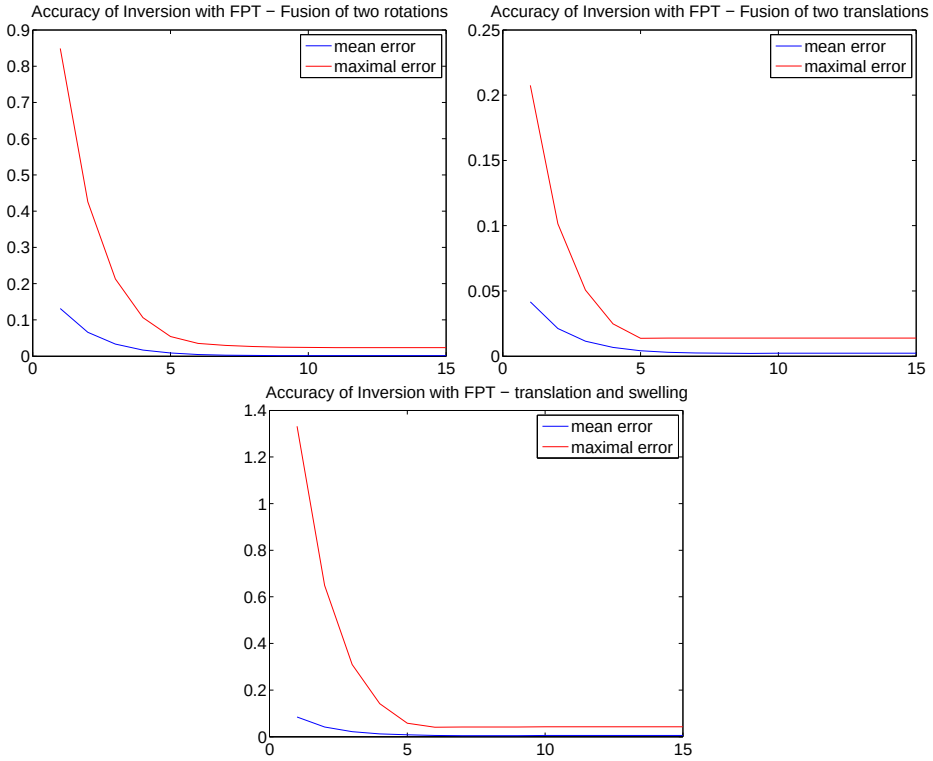

Figure 15: **Inverting a polyaffine transformation with the FPT: quantitative results. From left to right and then from top to bottom:** fusion between two rotations, two translation and finally a translation and an anisotropic swelling. The composition between the FPT of the transformation of the FPT of its inverse is carried out, for different numbers of squarings. The errors displayed are relative with respect the polyaffine transformation considered: the displacements are expected to be close to zero (i.e. the resulting transformation is expected to be close to the identity), and the errors are measured with respect to the displacements observed originally. One can see that an excellent accuracy of inversion is already achieved with 6 squarings. As expected, the maximal errors are observed at the boundary of the grid, which can be fixed for example by using a larger grid to compute the FPT.

3.4 3D MRI Example

Let us now consider a real 3D example of locally affine registration, between an atlas of $216 \times 180 \times 180$ voxels and a T_1-weighted MR image, with the multi-resolution and robust block-matching algorithm described in [5], without regularization. 7 structures of interest are considered: eyes (1 affine component each), cerebellum (2 components), brain stem (2 components), optic chiasm (1 component), 1 supplementary component (set to the identity) elsewhere. Weight functions are defined in the atlas geometry using mathematical morphology and a smoothing kernel in a preliminary step [5].

Philosophy of our Locally Affine Algorithm. Here, the idea is to use a registration procedure capable of registering finely a number of fixed structures of interest, with *very smooth* transformations. In contrast, many registration algorithms are able to register finely *the intensities* of the images of two anatomies, but this is done in most cases at the cost of the regularity of the resulting spatial transformation. This lack of smoothness leads to serious doubts regarding the anatomical likelihood of such transformations.

Fig. 16 provides a comparison between the typical smoothness of dense transformation and locally affine registration results. Interestingly, much smoother deformations are obtained in the locally affine case with an accuracy in the structures of interest which is comparable to the dense transformation case of [20].

LEPTs as a Post-Processing Tool. To obtain short computation times (typically 10 minutes), our locally affine registration algorithm estimates affine components using the *direct fusion*. The FPT is used in a *final step* to ensure the invertibility of the final transformation, as well as to compute its inverse.

Here, the scaling used in 2^8 and the FPT is computed in $40s$ on a Pentium4 Xeon™2.8 GHz on a $216 \times 180 \times 180$ regular grid. As shown by Fig. 17, the direct fusion of components estimated by [5] can lead to singularities, which is not the case when the FPT is used. Remarkably, both fusions are very close *outside* of regions with singularities. This means that no artifacts are introduced by the FPT, which justifies *a posteriori* the estimation of affine components with the (faster) direct fusion.

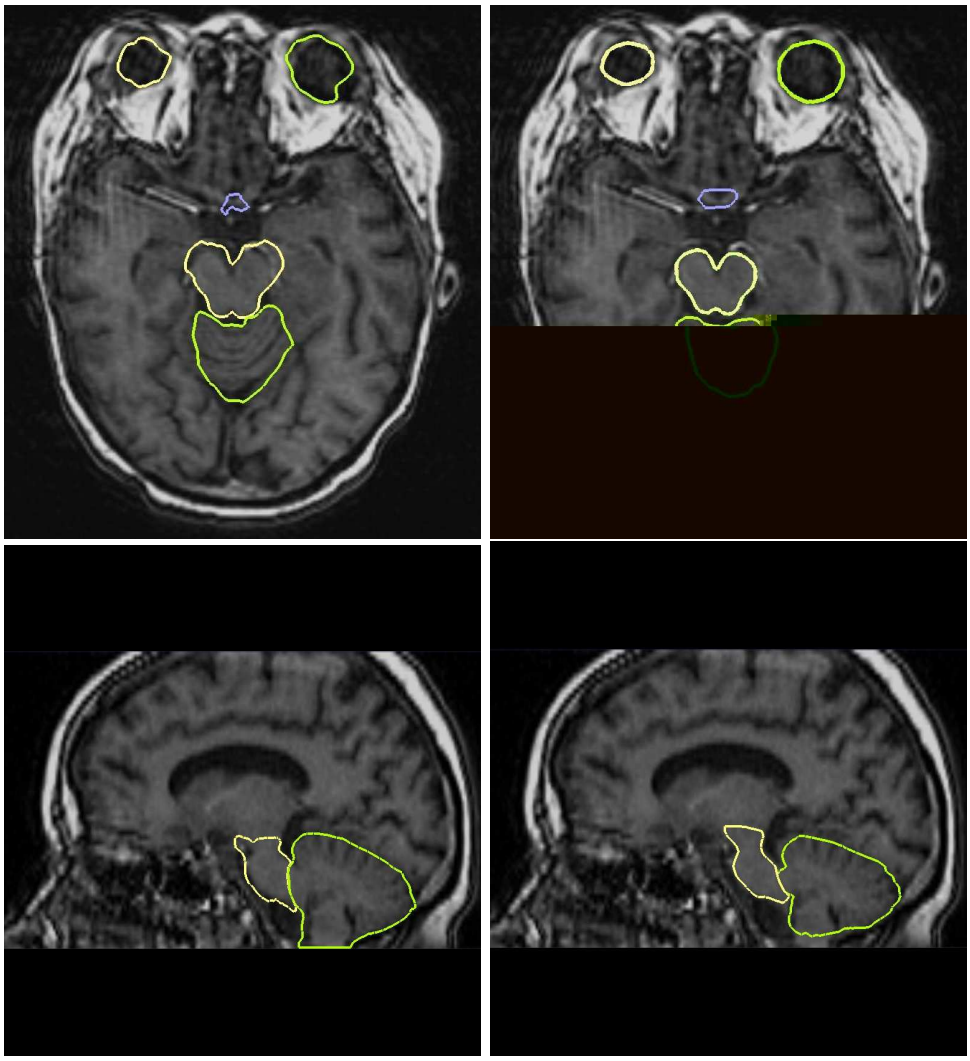

Figure 16: **Locally affine vs. dense-transformation: smoothness of deformations.** The contours of our structures of interest (eyes, brain stem, cerebellum, optic chiasm) are displayed on the subject and are obtained by deforming those of the atlas using the dense transformation of [20] and using our locally affine framework. **From left to right**: dense and then locally affine deformations (top: axial slice, bottom: sagittal slice). Note how smoother contours are in the locally affine case, although both accuracies are comparable.

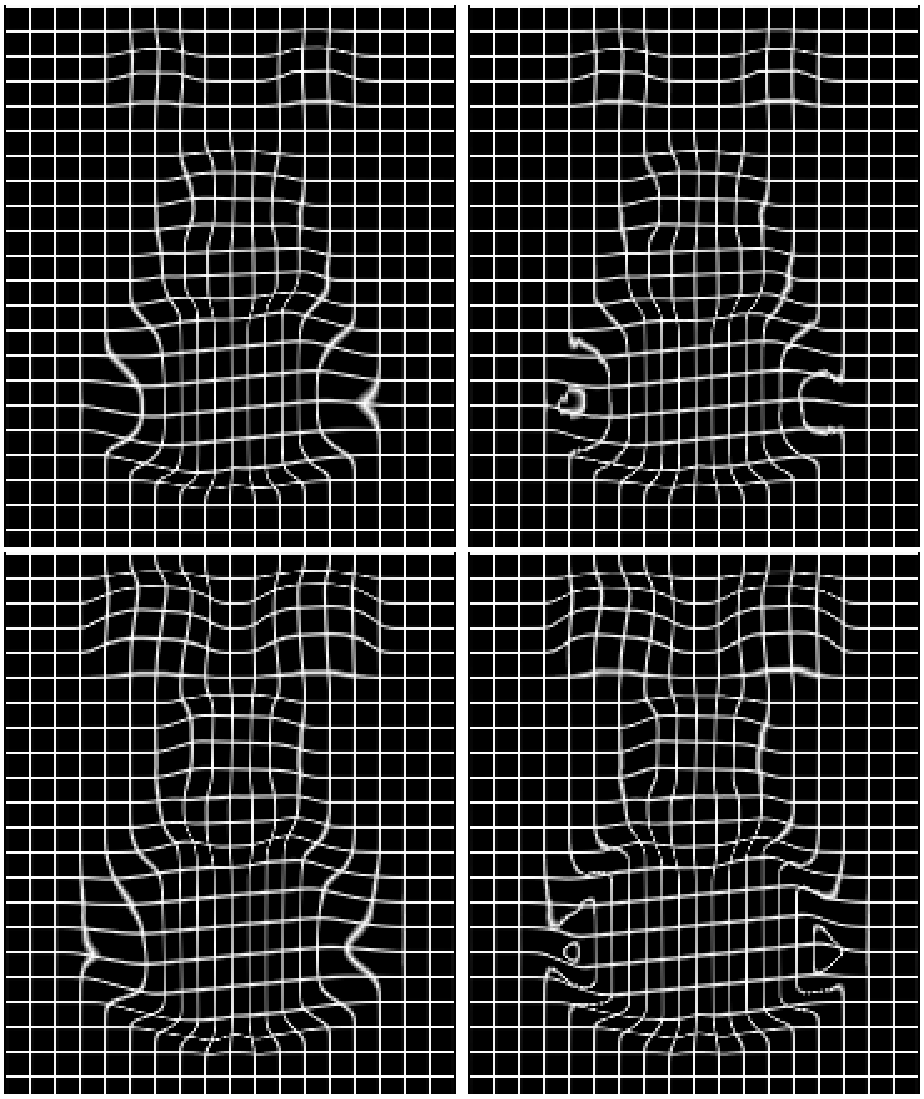

Figure 17: **Singularity removal with LEPTs.** A 3D regular grid is deformed with the locally affine transformation obtained with the algorithm of [5], two slices are displayed. **From left to right:** polyaffine fusion and direct fusion (two axial slices are displayed: one on top, one at the bottom). Note how the singularities of the direct fusion disappear with LEPTs. Remarkably, this is obtained without introducing any artifacts: outside singularities, both fusions yield very close results.

4 Conclusion and Perspectives

In this work, we have presented a novel framework to fuse rigid or affine components into a global transformation, called *Log-Euclidean polyaffine*. Similarly to the previous polyaffine framework of [4], it guarantees the *invertibility* of the result. However, contrary to the previous framework, this is achieved with very intuitive properties: for example the inverse of a LEPT is a LEPT with identical weights and inverted affine components. Moreover, this novel fusion is *affine-invariant*, i.e. does not depend on the choice of coordinate system. We have also shown that remarkably, and contrary to previous polyaffine transformations, the specific properties of LEPTs allow their fast computations on regular grids, with an algorithm called the 'Fast Polyaffine Transform', whose efficiency is somehow comparable to that of the Fast Fourier Transform.

In the example of locally affine 3D registration presented here, we use LEPTs in a final step to fuse the affine components estimated during the algorithm of [5]. With the FPT, this is done very efficiently. Remarkably, the novel fusion is *very close* to the direct fusion in regions without singularities. This suggests that our novel framework provides a general and efficient way of fusing local rigid or affine deformations into a global invertible transformation without introducing artifacts, *independently* of the way local affine deformations are first estimated.

We have also presented in this article a Log-Euclidean framework for rigid and affine transformations, which generalizes to linear transformations the Log-Euclidean framework we described in [9] for tensors. It allows a straightforward and efficient generalization to linear transformations of classical vectorial tools, with excellent theoretical properties. In particular, we have already used this framework in [5] to define a simple visco-elastic regularization energy for locally rigid or affine deformations. In future work, we are planning to use this simple framework to compute statistics on rigid and affine local components of deformations, which could help to better constraint non-rigid registration algorithms, as some of us begun to do in [21] and [22] with (local) statistics on strain tensors.

A A Log-Euclidean Framework for Rigid and Affine Transformations

Similarly to the Log-Euclidean framework for tensors [9], the rigid and affine Log-Euclidean frameworks are simple *and* have excellent theoretical properties. First, they is invariant with respect to inversion (of affine transformations).

Second, the Log-Euclidean mean of rigid or affine transformations (which is always a rigid or affine transformation) has very nice properties: it is affine-invariant, i.e. does not depend on the system of coordinates chosen. Furthermore, the determinant of the Log-Euclidean mean is equal to the geometric mean of the determinants of the data. This implies for instance that the Log-Euclidean mean of several contractions (resp. dilations) is still a contraction (resp. dilation). This is a great advantage compared to the Euclidean computations on affine transformations, which are also fast and simple , but do not take into account the group structure of the affine group. In particular, the arithmetic mean (in the sense of homogeneous coordinates) of several affine transformation is also affine-invariant but can perfectly be non-invertible, and the mean of several contractions can be a dilation (very much like in the tensor case).

A.1 Log-Euclidean Metrics

As we have seen in the Subsection 2.2, homogeneous coordinates provide a simple way to compute (and define!) the principal logarithm of affine transformations. We recall that this logarithm is well-defined if and only if the eigenvalues of the homogeneous (matrix) representation of the affine transformation do not lie on the (closed) half-line of negative real numbers [12]. We will here call such transformations *admissible*. In the particular case of rigid transformations this means that the amount of local rotation should not reach π radians [23]. A more detailed discussion of this restriction is given in Subsection 2.1.

Details about our numerical implementation of the matrix logarithm are given in the subsection A.4 of this Appendix.

Log-Euclidean Distances. Then, let T_1 and T_2 be two admissible affine transformations. A *Log-Euclidean distance* (or metric) between the two transformations will be of the following form:

$$\mathrm{dist}(T_1, T_2) = \| \log(T_1) - \log(T_2) \|, \tag{14}$$

where $\|.\|$ is a Euclidean norm, for example the Frobenius norm, which is defined by $\|M\|_{\mathrm{Frob}} = \left(\mathrm{Trace}(M^T.M)\right)^{\frac{1}{2}}$. This is the norm we will be using in the rest of this article.

Log-Euclidean Mean. As in the tensor case, one can associate to Log-Euclidean distances a generalization of the *arithmetic mean*, in the classical way, called the *Fréchet* mean. The

Log-Euclidean mean $\mathbb{E}(T_1, ..., T_2)_{\mathrm{LE}}$ of N admissible transformations $T_1, ...T_N$ with non-negative weights $w_1, ..., w_N$ (such that $\sum_i w_i = 1$) is defined as the point minimizing the following *metric dispersion*:

$$\mathbb{E}(T_1, ..., T_N)_{\mathrm{LE}} = \arg \min_{\text{admissible } T} \sum_{i=1}^{N} w_i \operatorname{dist}^2(T, T_i).$$

As in the tensor case, classical Euclidean geometry on the principal logarithms of the transformations show that the Log-Euclidean mean is indeed well-defined for transformations close enough to a scaled version of the identity. In this case, it is simply given by exponential of the arithmetic mean of data:

$$\mathbb{E}(T_1, ..., T_N)_{\mathrm{LE}} = \exp\left(\sum_{i=1}^{N} w_i . \log(T_i)\right).$$

For admissible transformations very far away from the identity, it could be possible that the arithmetic mean of their logarithms lies *outside* the logarithmic domain of admissible transformations! To our knowledge, although the set of complex numbers whose imaginary part lies in $]-\pi, \pi[$ is obviously convex, the set of real matrices whose eigenvalues are all in this domain is only *open*, and *not convex*. This set of matrices is only *locally* convex.

In practice, to check the well-definedness of the Log-Euclidean mean, it suffices to check that the Euclidean mean of the logarithms is still admissible. However, having a simple and general criterion on the T_i ensuring that all convex combinations of their logarithms are admissible would be very desirable, and this will be the subject of future work.

The Log-Euclidean Mean as a Geometric Mean. Exactly as in the tensor case, the determinant of the homogeneous representation of the Log-Euclidean mean is equal to the scalar *geometric mean* of the determinants of the data. In other words, we have:

$$\det(\mathbb{E}(T_1, ..., T_N)) = \exp\left(\sum_i w_i \ln\left(\det(T_i)\right)\right),$$

where ln is the (scalar) natural logarithm.

This can be shown with the same techniques as for tensors (well, actually nothing more than Jordan or Schur decompositions of matrices), see [16] for more details. The Log-Euclidean mean is therefore a generalization of the scalar geometric mean to affine transformations. The determinant of affine transformations has a very simple physical interpretation: it describes how volumes are changed by the affine transformation. Above 1, the transformation dilates volumes, and below 1, there is a contraction. The geometric interpolation of the determinants of the data performed by Log-Euclidean averaging thus guarantees that determinants are monotonically interpolated. This is not true for the arithmetic mean of affine-transformation, since Euclidean operations do not take into account the group structure of affine transformations. The arithmetic mean of several affine transformations can for

example introduce more dilation than there originally was in the data, which is the 'swelling effect' well-known in the tensor case [9].

A.2 Invariance Properties

Invariance by Inversion. It can be easily seen that any Log-Euclidean distance on affine transformations is invariant by *inversion*: the principal logarithm of an inverted transformation is simply the logarithm of the original transformation multiplied by -1, which does not change the value the distance (see Eq. (14)).

Rotational Invariance. A number of Log-Euclidean distances on affine transformations are *rotation-invariant*. This means that when the coordinate system is changed by a rotation, the Log-Euclidean distance between two affine transformations is unchanged. To see this, let us recall how an affine change of coordinates affects affine transformations. Let T be an affine transformation used to deformed locally the ambient space and let A be another affine transformation, used this time to change the coordinate system. When the point x is changed into $A.x$, then the affine transformation T is classically changed into $A.T.A^{-1}$. The principal logarithm of $A.T.A^{-1}$ is simply $A.\log(T).A^{-1}$. Using this equation, we get in homogeneous coordinates:

$$\log(A.T.A^{-1}) \sim \begin{pmatrix} M_2.L.M_2^{-1} & -M_2.L.M_2^{-1}.t_2 + M_2.v \\ 0 & 0 \end{pmatrix}, \tag{15}$$

where $A = (M_2, t_2)$ and where L and v are respectively the linear part and the translation part of $\log(T)$. As consequence, when M_2 is a rotation matrix R and when t_2 is equal to zero, we get:

$$\|\log(A.T.A^{-1})\|^2_{\text{Frob.}} = \|R.L.R^T\|^2_{\text{Frob.}} + \|R.v\|^2_{\text{Eucl.}} = \|L\|^2_{\text{Frob.}} + \|v\|^2_{\text{Eucl.}} = \|\log(T)\|^2_{\text{Frob.}},$$

which precisely means that this Log-Euclidean distance is rotation-invariant.

And Translation-Invariance? We have just shown that some Log-Euclidean distances on affine transformations are rotation-invariant, which means that the orientation of the current coordinate system does not influence computations with this type of metrics. What about translation, i.e. how does the choice of the *origin* of the coordinate system have an impact on Log-Euclidean metrics? To see this, let us re-write Eq. (15) in the case where the affine change of coordinate system is a pure translation and writes $A = (Id, t_2)$. Then Eq. (15) becomes:

$$\log(A.T.A^{-1}) \sim \begin{pmatrix} L & -L.t_2 + v \\ 0 & 0 \end{pmatrix},$$

which implies in turn that we have:

$$\|\log(A.T.A^{-1})\|_{\text{Frob.}} = \|L\|^2_{\text{Frob.}} + \|v - L.t_2\|^2_{\text{Eucl.}},$$

which depends on t_2 and is not equal in general to $\|\log(T)\|^2_{\text{Frob.}}$. Changing the origin of the coordinate system by a translation t_2 results in shifting by $-L.t_2$ the translation part v of the logarithm of the current affine transformation T, which changes the norm of $\log(T)$ in general.

The Frobenius Log-Euclidean distance is therefore not *translation-invariant*, and one could wonder whether this biases the computations made with such a distance in the case of multi-affine registration. Actually, this is not the case when one restricts oneself to using only *Log-Euclidean means* of affine transformations. This comes from the fact that this type of Fréchet mean is completely *affine-invariant*, i.e. does not depend at all on the current system of coordinates. The regularization strategy we propose in Subsection A.3 entirely relies entirely on such means, which ensures its affine-invariance.

Affine-Invariance of the Log-Euclidean Mean. As announced in the previous paragraph, the Log-Euclidean mean is *affine-invariant*: it is not biased by the current coordinate system. To see this, let $T_1, ..., T_N$ be N affine transformations with logarithms $(L_1, v_1), ..., (L_N, v_N)$, and let $w_1, ..., W_N$ N be N non-negative weights. Using Eq. (15), we have:

$$\begin{array}{rcl}
\log(\mathbb{E}(A.T_1.A^{-1}, ..., A.T_N.A^{-1})_{\text{LE}}) & = & \sum_i w_i \log(A.T_i.A^{-1}) \\
& = & \sum_i w_i A.\log(T_i).A^{-1} \\
& = & A.\left(\sum_i w_i \log(T_i)\right).A^{-1} \\
& = & A.\log\left(\mathbb{E}(T_1, ..., T_N)\right).A^{-1} \\
& = & \log\left(A.\mathbb{E}(T_1, ..., T_N).A^{-1}\right)
\end{array}$$

which implies the affine-invariance of the Log-Euclidean mean:

$$\mathbb{E}(A.T_1.A^{-1}, ..., A.T_N.A^{-1})_{\text{LE}} = A.\log(\mathbb{E}(T_1, ..., T_N)_{\text{LE}}).A^{-1}.$$

A.3 Log-Euclidean Regularization for Locally Rigid or Affine Registration

One of the great advantages of the Log-Euclidean framework is that classical vector tools can be readily recycled in this framework. Once affine-transformations have been transformed into their principal logarithm, one can simply perform Euclidean operations on them. This allows the direct generalization of classical *vectorial* regularization tools to locally affine deformations, which we have already begun to use in [5].

Elastic Regularization. In the context of locally rigid or affine registration, we propose to use the following *elastic regularity energy*:

$$Reg(T_i, w_i) = \frac{1}{2}.\sum_{i=1}^{N}\sum_{j\neq i} p_{i,j}\|\log(T_i) - \log(T_j)\|^2, \tag{16}$$

where the *correlation weights* $p_{i,j}$ between the components are defined in the following way:

$$p_{i,j} = \left(\int_\Omega w_i(x).w_j(x)dx \right) / \left(\int_\Omega w_i(x)dx \right),$$

where Ω is the (bounded) image domain chosen for the registration experiment. Thus, this type of energy takes into account the spatial extensions of the components, and the various correlations existing between them.

Elastic Regularization and Log-Euclidean Means. From a practical point of view, to regularize the current locally affine transformation, we perform a gradient descent of the elastic energy given in (16). The partial derivative of this energy writes:

$$\frac{\partial}{\partial \log(T_k)} Reg(T_i, w_i) = \sum_{j \neq k} p_{j,k} \left(\log(T_k) - \log(T_j) \right).$$

As a consequence, performing a gradient descent of our elastic energy simply results in replacing affine transformations T_i by Log-Euclidean means between all affine transformations, the weights depending on the correlation weights $p_{i,j}$ and on the time step used (which should be small enough to ensure that the weights used are all non-negative). This guarantees the *affine-invariance* of our regularization approach, since performing Log-Euclidean means of affine transformations is an affine-invariant operation. This was not obvious at all, since the Log-Euclidean metric we use is only *rotation-invariant*, and *not* affine-invariant.

Extensions of this Approach. The principle behind the Log-Euclidean framework is the following: the principal logarithm provides a simple way of mapping affine transformations into a vector space, while conserving excellent theoretical properties which are consistent with the group structure of these transformations. Once affine-transformations have been transformed into vectors with the matrix logarithm, it is quite straightforward to generalize to these transformations all classical variational approaches usually reserved to vectors.

Therefore, other types of regularization techniques, such as *fluid regularization* (i.e. elastic regularization on the small modifications made to the components between the iterations of the registration algorithm), can be generalized to locally affine transformations in the same straightforward way as elastic regularization. In [5] for example, we use *visco-elastic* regularization (i.e. elastic *and* fluid regularization), exactly as was done in [24] in the (vector) dense-transformation case.

Synthetic 2D Experiment. Here, we regularize a polyaffine transformation using the Log-Euclidean elastic regularization energy of Eq. (16). 9 components were regularly defined on the grid, and their affine transformations were chosen randomly. A gradient descent on the energy (16) was performed, and the result is shown in fig. 18. Note how the 9 components all converge toward a Log-Euclidean mean of the original affine transformations as the degree of regularization increases.

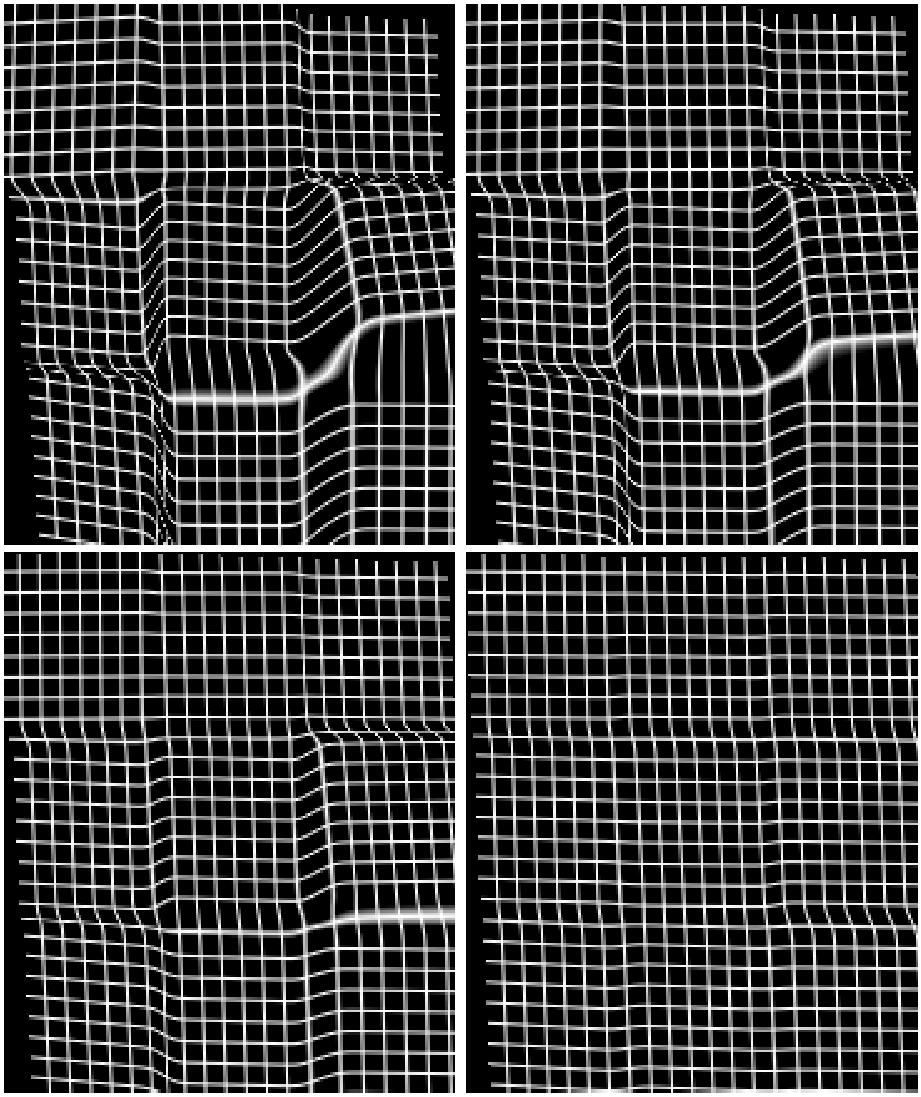

Figure 18: **Log-Euclidean regularization of a polyaffine transformation. From left to right and then from top to bottom:** regular grid deformed by original polyaffine transformation (with 9 affine components), the same regular grid deformed by the increasingly regularized polyaffine transformation. Note how the 9 components all converge toward a Log-Euclidean mean of the original affine transformations as the regularization increases.

A.4 Numerical Implementation of Matrix Logarithm.

In this work, we have used the 'Inverse Scaling and Squaring' method [12] to compute matrix logarithms. It is the equivalent of the 'Scaling and Squaring' method for logarithms, and is based on the idea that computing the logarithm of a matrix *close to the identity* is easier than computing the logarithm of a general matrix. Like in the case of the matrix exponential, this computation can be done very accurately and at a very small computational cost using Padé approximants.

In order transform a given matrix into another one closer to the identity, the 'Inverse Scaling and Squaring' method uses the computation of successive *square roots*. Once the $2^{N^{\text{th}}}$ root of a matrix M has been computed, one can use the following equality to compute the logarithm of M:

$$\log(M) = 2^N . \log\left(M^{2^{-N}}\right). \tag{17}$$

Actually, (17) is nothing more than Eq. (12) in the domain of logarithms.

More details on how square roots can be iteratively computed and on the choice of the level of squarings N can be found in [12].

References

[1] J.B.A. Maintz and M.A Viergever. A survey of medical registration. *Medical image analysis*, 2(1):1–36, 1998.

[2] X. Papademetris, D.P. Dione, L.W. Dobrucki, L.H. Staib, and A.J. Sinusas. Articulated rigid registration for serial lower-limb mouse imaging. In *MICCAI'05 (2)*, pages 919–926, 2005.

[3] A. Pitiot, E. Bardinet, P.M. Thompson, and G. Malandain. Piecewise affine registration of biological images for volume reconstruction. *MedIA*, 2005. accepted for publication.

[4] V. Arsigny, X. Pennec, and N. Ayache. Polyrigid and polyaffine transformations: a novel geometrical tool to deal with non-rigid deformations - application to the registration of histological slices. *Med. Im. Anal.*, 9(6):507–523, December 2005.

[5] O. Commowick, V. Arsigny, J. Costa, G. Malandain, and N. Ayache. An efficient multi-affine framework for the registration of anatomical structures. In *Proceedings of ISBI'2006*. IEEE, 2006. To appear.

[6] R. Narayanan, J.A. Fessler, H. Park, and C.R. Meyer. Diffeomorphic nonlinear transformations: A local parametric approach for image registration. In *Proceedings of IPMI'05*, volume 3565 of LNCS, pages 174–185, 2005.

[7] A. Cuzol, P. Hellier, and E. Mémin. A novel parametric method for non-rigid image registration. In *Proc. of IPMI'05*, number 3565 in LNCS, pages 456–467, 2005.

[8] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes. Non-rigid registration using free-form deformations: Application to breast MR images. *IEEE Trans. Medecal Imaging*, 18(8):712–721, 1999.

[9] Vincent Arsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. Fast and simple calculus on tensors in the log-Euclidean framework. In J. Duncan and G. Gerig, editors, *Proceedings of the 8th Int. Conf. on Medical Image Computing and Computer-Assisted Intervention - MICCAI 2005, Part I*, volume 3749 of *LNCS*, pages 115–122, Palm Springs, CA, USA, October 26-29, 2005. Springer Verlag.

[10] J.A. Little, D.L.G. Hill, and D.J. Hawkes. Deformations incorpotationg rigid structures. *CVIU*, 66(2):223–232, May 1996.

[11] D. Sheppard. A two-dimensionnal interpolation function for irregularly spaced data. In *23rd National Conference of the ACM*, pages 517–524, 1968.

[12] S. Hun Cheng, N. J. Higham, C. S. Kenney, and A. J. Laub. Approximating the logarithm of a matrix to specified accuracy. *SIAM J. Matrix Anal. Appl.*, 22(4):1112–1125, 2001.

[13] Shlomo Sternberg. *Lectures on Differential Geometry.* Prentice Hall Mathematics Series. Prentice Hall Inc., 1964.

[14] M. Tenenbaum and H. Pollard. *Ordinary Differential Equations.* Dover, 1985.

[15] Roger Godement. *Introduction à la Théorie des Groupes de Lie.* Publications Mathématiques de l'Université Paris VII, 1982.

[16] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache. Fast and simple computations on tensors with Log-Euclidean metrics. Research Report RR-5584, INRIA, Sophia-Antipolis, France, May 2005.

[17] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM jour. of Matr. Anal. and Appl.*, 20(4):801–836, October 1978.

[18] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179–1193, 2005.

[19] J. D. Lambert. *Numerical methods for ordinary differential systems: the initial value problem.* John Wiley & Sons, Inc., New York, NY, USA, 1991.

[20] R. Stefanescu, X. Pennec, and N. Ayache. Grid powered nonlinear image registration with locally adaptive regularization. *Med. Im. Anal.*, 8(3):325–342, September 2004.

[21] Xavier Pennec, Radu Stefanescu, Vincent Arsigny, Pierre Fillard, and Nicholas Ayache. Riemannian elasticity: A statistical regularization framework for non-linear registration. In J. Duncan and G. Gerig, editors, *Proceedings of the 8th Int. Conf. on Medical Image Computing and Computer-Assisted Intervention - MICCAI 2005, Part II*, volume 3750 of *LNCS*, pages 943–950, Palm Springs, CA, USA, October 26-29, 2005. Springer Verlag.

[22] Olivier Commowick, Radu Stefanescu, Pierre Fillard, Vincent Arsigny, Nicholas Ayache, Xavier Pennec, and Grégoire Malandain. Incorporating statistical measures of anatomical variability in atlas-to-subject registration for conformal brain radiotherapy. In J. Duncan and G. Gerig, editors, *Proceedings of the 8th Int. Conf. on Medical Image Computing and Computer-Assisted Intervention - MICCAI 2005, Part II*, volume 3750 of *LNCS*, pages 927–934, Palm Springs, CA, USA, October 26-29, 2005. Springer Verlag.

[23] R.P. Woods. Characterizing volume and surface deformations in an atlas framework: theory, applications, and implementation. *Neuroimage*, 18(3), 2003.

[24] Radu Stefanescu. *Parallel nonlinear registration of medical images with a priori information on anatomy and pathology.* PhD thesis, Université de Nice – Sophia-Antipolis, March 2005.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399