itk::Transforms supporting spatial derivatives

Release 1.0

Marius Staring[1] and Stefan Klein[2]

September 8, 2010

[1]Division of Image Processing, Leiden University Medical Center, Leiden, The Netherlands
[2]Biomedical Imaging Group Rotterdam, Departments of Radiology & Medical Informatics, Erasmus MC, Rotterdam, The Netherlands

Abstract

This document describes the use and implementation of first and second order spatial derivatives of coordinate transformations in the Insight Toolkit (www.itk.org). Spatial derivatives are useful for many types of regularising or penalty terms frequently used in image registration. These derivatives are dubbed 'SpatialJacobian' and 'SpatialHessian' to distinguish with the derivative to the transformation parameters themselves, which is called 'Jacobian' in the ITK.

In addition to the spatial derivatives, we derived and implemented the derivatives to the registration/transform parameters of these spatial derivatives, required for gradient descent type optimisation routines. These derivatives are implemented in a sparse manner, reducing the computation time for transformations which have local support. All of these derivatives are implemented for the most common ITK coordinate transformation, such as the rigid, affine and B-spline transformation. In addition we derive formulae and code for arbitrary compositions of transformations. The spatial derivatives were subsequently exploited by implementing the bending energy penalty term.

This paper is accompanied with the source code, input data, parameters and output data that the authors used for validating the algorithm described in this paper. This adheres to the fundamental principle that scientific publications must facilitate reproducibility of the reported results.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/1338]

Contents

1 Introduction **2**

2 Support for penalty terms in the ITK **3**

3 Affine transformation **6**

4 B-spline transformation **6**

5 Combining transformations **8**

6 Bending energy penalty term **9**

7 Discussion and Conclusion **10**

1 Introduction

Image registration is the process of aligning images, and can be defined as an optimisation problem [2]:

$$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{\mu}} \mathcal{C}(I_F, I_M; \boldsymbol{\mu}), \tag{1}$$

with I_F and I_M the d-dimensional fixed and moving image, respectively, and $\boldsymbol{\mu}$ the vector of parameters of size N that parameterise the transformation $\boldsymbol{T}_{\boldsymbol{\mu}} = [T_{\boldsymbol{\mu},1}, \ldots, T_{\boldsymbol{\mu},d}]^\dagger = [T_1, \ldots, T_d]^\dagger$, where we have dropped $\boldsymbol{\mu}$ for short notation, and where † denotes transposition. The cost function $\mathcal{C}$ consists of a similarity measure $\mathcal{S}(I_F, I_M; \boldsymbol{\mu})$ that defines the quality of alignment. Examples are the mean square difference, normalised correlation, and mutual information measure. In order to regularise the transformation $\boldsymbol{T}_{\boldsymbol{\mu}}$ often a penalty term $\mathcal{P}(\boldsymbol{\mu})$ is added to the cost function, so the problem becomes:

$$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{\mu}} \mathcal{S}(I_F, I_M; \boldsymbol{\mu}) + \alpha \mathcal{P}(\boldsymbol{\mu}), \tag{2}$$

where α is a user-defined constant that weighs similarity against regularity.

Penalty term are often based on the first or second order spatial derivatives of the transformation [5, 6]. For example the bending energy of the transformation, which is arguably the most common penalty term, is defined in 2D as:

$$\mathcal{P}_{\mathrm{BE}}(\boldsymbol{\mu}) = \frac{1}{P} \sum_{\widetilde{\boldsymbol{x}}_i} \left\| \frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger}(\widetilde{\boldsymbol{x}}_i) \right\|_F^2 \tag{3}$$

$$= \frac{1}{P} \sum_{\widetilde{\boldsymbol{x}}_i} \sum_{j=1}^{2} \left(\frac{\partial^2 T_j}{\partial x_1^2}(\widetilde{\boldsymbol{x}}_i) \right)^2 + 2 \left(\frac{\partial^2 T_j}{\partial x_1 \partial x_2}(\widetilde{\boldsymbol{x}}_i) \right)^2 + \left(\frac{\partial^2 T_j}{\partial x_2^2}(\widetilde{\boldsymbol{x}}_i) \right)^2, \tag{4}$$

where P is the number of points $\widetilde{\boldsymbol{x}}_i$, and the tilde denotes the difference between a variable and a given point over which a term is evaluated.

The optimisation problem (1) is frequently solved using an iterative gradient descent routine [3]:

$$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - a_k \frac{\partial \mathcal{C}}{\partial \boldsymbol{\mu}} = \boldsymbol{\mu}_k - a_k \left(\frac{\partial \mathcal{S}}{\partial \boldsymbol{\mu}} + \alpha \frac{\partial \mathcal{P}}{\partial \boldsymbol{\mu}} \right), \tag{5}$$

with a_k a user-defined (or automatically determined [1]) declining function that defines the step size.

The derivative of the similarity measure usually involves computation of the spatial derivative of the moving image: $\partial I_M / \partial \boldsymbol{x}$, and the derivative of the transformation to its parameters: $\partial \boldsymbol{T} / \partial \boldsymbol{\mu}$. In the ITK the last derivative is implemented using `transform->GetJacobian()`, i.e. the derivative to the transformation parameters $\boldsymbol{\mu}$ is referred to as 'Jacobian'.

Penalty terms usually consist of the first and second order *spatial* derivatives of the transformation, i.e. $\partial \boldsymbol{T} / \partial \boldsymbol{x}$ and $\partial^2 \boldsymbol{T} / \partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger$. We will refer to these derivatives as the 'SpatialJacobian' and the 'SpatialHessian'

Name	definition	matrix size	written out in 2D
Transformation			
	$\boldsymbol{T} = \boldsymbol{T}_{\boldsymbol{\mu}}(\widetilde{\boldsymbol{x}})$	$d \times 1$	$\begin{bmatrix} T_1(\widetilde{\boldsymbol{x}}) \\ T_2(\widetilde{\boldsymbol{x}}) \end{bmatrix}$
Jacobian			
	$\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{\mu}}(\widetilde{\boldsymbol{x}})$	$d \times N$	$\begin{bmatrix} \frac{\partial T_1}{\partial \mu_1}(\widetilde{\boldsymbol{x}}) & \cdots & \frac{\partial T_1}{\partial \mu_N}(\widetilde{\boldsymbol{x}}) \\ \frac{\partial T_2}{\partial \mu_1}(\widetilde{\boldsymbol{x}}) & \cdots & \frac{\partial T_2}{\partial \mu_N}(\widetilde{\boldsymbol{x}}) \end{bmatrix}$
SpatialJacobian			
	$\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x}}(\widetilde{\boldsymbol{x}})$	$d \times d$	$\begin{bmatrix} \frac{\partial T_1}{\partial x_1}(\widetilde{\boldsymbol{x}}) & \frac{\partial T_1}{\partial x_2}(\widetilde{\boldsymbol{x}}) \\ \frac{\partial T_2}{\partial x_1}(\widetilde{\boldsymbol{x}}) & \frac{\partial T_2}{\partial x_2}(\widetilde{\boldsymbol{x}}) \end{bmatrix}$
JacobianOfSpatialJacobian			
	$\frac{\partial}{\partial \boldsymbol{\mu}} \frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x}}(\widetilde{\boldsymbol{x}})$	$d \times d \times N$	$\begin{bmatrix} \frac{\partial}{\partial \mu_1} \frac{\partial \boldsymbol{T}_{\boldsymbol{\mu}}}{\partial \boldsymbol{x}}(\widetilde{\boldsymbol{x}}) & \cdots & \frac{\partial}{\partial \mu_N} \frac{\partial \boldsymbol{T}_{\boldsymbol{\mu}}}{\partial \boldsymbol{x}}(\widetilde{\boldsymbol{x}}) \end{bmatrix}$
SpatialHessian			
	$\frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger}(\widetilde{\boldsymbol{x}})$	$d \times d \times d$	$\left\{ \begin{bmatrix} \frac{\partial^2 T_1}{\partial x_1 \partial x_1}(\widetilde{\boldsymbol{x}}) & \frac{\partial^2 T_1}{\partial x_2 \partial x_1}(\widetilde{\boldsymbol{x}}) \\ \frac{\partial^2 T_1}{\partial x_2 \partial x_1}(\widetilde{\boldsymbol{x}}) & \frac{\partial^2 T_1}{\partial x_2 \partial x_2}(\widetilde{\boldsymbol{x}}) \end{bmatrix}, \begin{bmatrix} \frac{\partial^2 T_2}{\partial x_1 \partial x_1}(\widetilde{\boldsymbol{x}}) & \frac{\partial^2 T_2}{\partial x_2 \partial x_1}(\widetilde{\boldsymbol{x}}) \\ \frac{\partial^2 T_2}{\partial x_2 \partial x_1}(\widetilde{\boldsymbol{x}}) & \frac{\partial^2 T_2}{\partial x_2 \partial x_2}(\widetilde{\boldsymbol{x}}) \end{bmatrix} \right\}$
JacobianOfSpatialHessian			
	$\frac{\partial}{\partial \boldsymbol{\mu}} \frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger}(\widetilde{\boldsymbol{x}})$	$d \times d \times d \times N$	$\begin{bmatrix} \frac{\partial}{\partial \mu_1} \frac{\partial^2 \boldsymbol{T}_{\boldsymbol{\mu}}}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger}(\widetilde{\boldsymbol{x}}) & \cdots & \frac{\partial}{\partial \mu_N} \frac{\partial^2 \boldsymbol{T}_{\boldsymbol{\mu}}}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger}(\widetilde{\boldsymbol{x}}) \end{bmatrix}$

Table 1: Naming conventions and definitions for the transformation and its derivatives used in this paper.

to clearly distinguish between these derivatives and the 'Jacobian'. In order to apply the gradient descent optimisation routine (5), we additionally need the derivatives $\frac{\partial}{\partial \boldsymbol{\mu}} \partial \boldsymbol{T} / \partial \boldsymbol{x}$ and $\frac{\partial}{\partial \boldsymbol{\mu}} \partial^2 \boldsymbol{T} / \partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger$. These we call the 'JacobianOfSpatialJacobian' and 'JacobianOfSpatialHessian', respectively. See Table 1 for details.

The paper is organised as follows: The interface of the proposed functions, and the chosen data structures is described in Section 2. In Sections 3 and 4 we give the mathematics for the 2D case of the spatial derivatives for the affine and B-spline transform, respectively. Attention is paid to the combination of multiple transforms in Section 5. In that section, equations are derived for the spatial derivatives, both for transformations that are combined using addition as well as composition. Finally, in Section 6, the spatial derivatives are utilised for the computation of the value $\mathcal{P}_{\mathrm{BE}}(\boldsymbol{\mu})$ and derivative $\frac{\partial}{\partial \boldsymbol{\mu}} \mathcal{P}_{\mathrm{BE}}(\boldsymbol{\mu})$ of the bending energy penalty term. The described new functionality was released in the registration toolkit `elastix` [2] previously. With this contribution we hope to make the functionality available to a greater audience.

2 Support for penalty terms in the ITK

The spatial derivative of the transform is not supported in the ITK. We propose to add the following functions in the `itk::Transform` class:

Name	ITK structure
Jacobian	`Array2D = vnl_matrix`
SpatialJacobian	`Matrix = vnl_matrix_fixed`
JacobianOfSpatialJacobian	`std::vector< Matrix >`
SpatialHessian	`FixedArray< Matrix >`[1]
JacobianOfSpatialHessian	`std::vector< FixedArray< Matrix > >`
NonZeroJacobianIndices	`std::vector< unsigned long >`

Table 2: The ITK structures that store the data.

```
virtual void GetSpatialJacobian(
  const InputPointType &,
  SpatialJacobianType & ) const;

virtual void GetSpatialHessian(
  const InputPointType &,
  SpatialHessianType & ) const;

virtual void GetJacobianOfSpatialJacobian(
  const InputPointType &,
  JacobianOfSpatialJacobianType &,
  NonZeroJacobianIndicesType & ) const;

virtual void GetJacobianOfSpatialHessian(
  const InputPointType &,
  JacobianOfSpatialHessianType &,
  NonZeroJacobianIndicesType & ) const;
```

and additionally a function to implement a sparse version of the Jacobian:

```
virtual void GetJacobian(
    const InputPointType &,
    JacobianType &,
    NonZeroJacobianIndicesType & ) const;
```

The ITK structures that were used to store the data are given in Table 2. The Jacobian is of size $d \times N$, and since the number of transformation parameters is flexible for some transformations, the data structure used for storing the Jacobian is an `itk::Array2D` object, which inherits from the `vnl_matrix`. This was already chosen previously in the ITK. The SpatialJacobian is of fixed size $d \times d$, and therefore (and for performance reasons) we choose to use the `itk::Matrix` to store the SpatialJacobian, which inherits from the `vnl_matrix_fixed`. For derivatives to $\boldsymbol{\mu}$ we choose to use the `std::vector`. The SpatialHessian gives us some problems, since we really need a 3D matrix, but currently no such thing exists in the ITK or in vnl. Therefore, we opt for an `itk::FixedArray` of `itk::Matrix`'s.

Notice the `NonZeroJacobianIndicesType` in the function definitions. These are meant for the support of sparse `Jacobians`, `JacobianOfSpatialJacobians`, etc. For local transformations like the B-

[1] by lack of a good 3D matrix structure

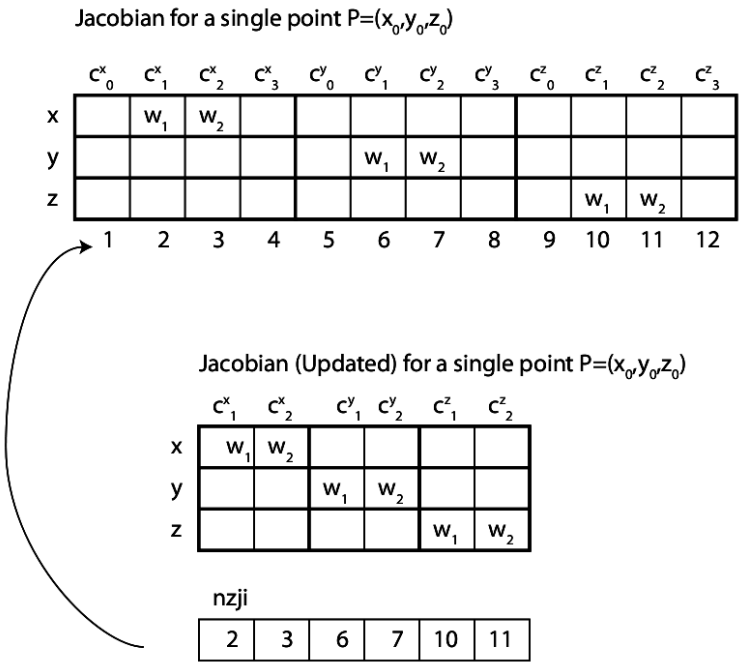

Figure 1: Illustrating sparse Jacobians. This figure is based on a 3D B-spline transform, where the parameters $\boldsymbol{\mu}$ are formed by the control point displacements $\boldsymbol{c}_i = (c_i^x, c_i^y, c_i^z)$. The principle applies to any transformation with sparse Jacobians though. The top matrix shows the non-zero entries of the full Jacobian matrix. The same information can be represented shorter by only storing these non-zero entries and the corresponding indices (nzji), see bottom two images. Image courtesy of M. Motes.

spline, only a small subset of the parameters $\boldsymbol{\mu}$ are needed to compute $\partial \boldsymbol{T}/\partial \boldsymbol{\mu}$ for a given coordinate $\boldsymbol{x}$ (and also for $\boldsymbol{T}(\boldsymbol{x})$), see Figure 1. The same holds for the `JacobianOfSpatialJacobian` $\frac{\partial}{\partial \boldsymbol{\mu}} \frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x}}$, and the `JacobianOfSpatialHessian` $\frac{\partial}{\partial \boldsymbol{\mu}} \frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\intercal}$, although the `NonZeroJacobianIndicesType` are based on that of $\partial \boldsymbol{T}/\partial \boldsymbol{\mu}$. For some transformations, e.g. the affine transform, the `SpatialHessian` or the `JacobianOfSpatialHessian` are completely filled with zeros. For computational purposes the following functions were therefore added

```
/** Whether the advanced transform has nonzero matrices. */
itkGetConstMacro( HasNonZeroSpatialHessian, bool );
itkGetConstMacro( HasNonZeroJacobianOfSpatialHessian, bool );
```

These functions allow skipping parts of a computation involving these matrices.

As a side note, in the ITK `GetJacobian()` is declared as:

```
virtual const JacobianType & GetJacobian( const InputPointType  & ) const;
```

and the result is stored in a protected member variable `m_Jacobian`. Although only subclasses can access this member, it should be noted that the result is only valid in combination with the provided input point, for transformations with a derivative dependent on the spatial position, which are most. Therefore, it may be better to remove this member variable altogether.

We have implemented the above for many of the `itk::Transform`'s, in new classes which are copies of the original ITK classes. The names of the new classes are prepended with 'Advanced'. In the end it would be best to integrate the new functionality in the original ITK classes. Advanced versions of the `itk::IdentityTransform`, `itk::Rigid2DTransform`, `itk::Rigid3DTransform`, `itk::MatrixOffsetTransformBase`, `itk::BSplineDeformableTransform`, and additionally for the `itk::CombinationTransform` (see [4]) are available.

3 Affine transformation

For the affine transformation, the derivatives evaluate to the following in 2D:

$$\boldsymbol{T}(\widetilde{\boldsymbol{x}}) = A(\boldsymbol{x}-\boldsymbol{c})+\boldsymbol{t}+\boldsymbol{c} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1-c_1 \\ x_2-c_2 \end{bmatrix} + \begin{bmatrix} t_1+c_1 \\ t_2+c_2 \end{bmatrix} \tag{6}$$

$$= \begin{bmatrix} \mu_0 & \mu_1 \\ \mu_2 & \mu_3 \end{bmatrix} \begin{bmatrix} x_1-c_1 \\ x_2-c_2 \end{bmatrix} + \begin{bmatrix} \mu_4+c_1 \\ \mu_5+c_2 \end{bmatrix}, \tag{7}$$

with A a matrix, $\boldsymbol{c}$ the center of rotation, and $\boldsymbol{t}$ a translation. Then

$$\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{\mu}}(\widetilde{\boldsymbol{x}}) = \begin{bmatrix} \widetilde{x}_1-c_1 & \widetilde{x}_2-c_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & \widetilde{x}_1-c_1 & \widetilde{x}_2-c_2 & 0 & 1 \end{bmatrix}, \tag{8}$$

$$\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x}}(\widetilde{\boldsymbol{x}}) = \begin{bmatrix} \mu_0 & \mu_1 \\ \mu_2 & \mu_3 \end{bmatrix}, \tag{9}$$

$$\frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger}(\widetilde{\boldsymbol{x}}) = O_{d\times d\times d}, \tag{10}$$

$$\frac{\partial}{\partial \boldsymbol{\mu}} \frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x}}(\widetilde{\boldsymbol{x}}) = \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, O_{d\times d}, O_{d\times d} \right\}, \tag{11}$$

$$\frac{\partial}{\partial \boldsymbol{\mu}} \frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger}(\widetilde{\boldsymbol{x}}) = O_{d\times d\times d\times N}, \tag{12}$$

where O_s is a zero matrix of size s.

The `GetJacobianOfSpatialJacobian()` returns `nonZeroJacobianIndices` = $[0,1,2,3,4,5]$, since $\partial \boldsymbol{T}/\partial \mu_i$ is nonzero for all i. The internal booleans `m_HasNonZeroSpatialHessian` and `m_HasNonZeroJacobianOfSpatialHessian` are set to `false` for the affine transform. The implementation of the penalty term is assumed to check for these booleans. In case of an affine transform the penalty term can simply return zero. This is a performance benefit compared to walking over the zero matrix, and adding and multiplying everything, which in the end also gives zero.

These derivatives are implemented in the `itk::AdvancedMatrixOffsetTransformBase` class.

4 B-spline transformation

A transformation parameterised by third order B-splines can be written in 2D as follows:

$$\boldsymbol{T}(\widetilde{\boldsymbol{x}}) = \begin{bmatrix} T_1(\widetilde{\boldsymbol{x}}) \\ T_2(\widetilde{\boldsymbol{x}}) \end{bmatrix} = \begin{bmatrix} \widetilde{x}_1 \\ \widetilde{x}_2 \end{bmatrix} + \begin{bmatrix} \sum_i \mu_{i1} \beta^3\left(\frac{\widetilde{x}_1 - x_1^i}{\sigma_1}\right) \beta^3\left(\frac{\widetilde{x}_2 - x_2^i}{\sigma_2}\right) \\ \sum_i \mu_{i2} \beta^3\left(\frac{\widetilde{x}_1 - x_1^i}{\sigma_1}\right) \beta^3\left(\frac{\widetilde{x}_2 - x_2^i}{\sigma_2}\right) \end{bmatrix}, \tag{13}$$

with $\boldsymbol{x}^i$ the control points, $\beta^3(\cdot)$ the third-order B-spline basis functions, σ_1 and σ_2 the B-spline grid spacing, and μ_{ij} the B-spline parameters. Since $\beta^3\left((\widetilde{x}_j - x_j^i)/\sigma_j\right)$ is zero outside the compact support region, the summations in (13) can be performed over a subset of the parameters, instead of over the complete $\boldsymbol{\mu}$. As mentioned previously, this is implemented via the nonzero Jacobian indices.

For short notation, define:

$$b_{33}^i = \beta^3\left((\widetilde{x}_1 - x_1^i)/\sigma_1\right) \cdot \beta^3\left((\widetilde{x}_2 - x_2^i)/\sigma_2\right), \tag{14}$$

$$b_{23}^i = \left[\beta^2\left((\widetilde{x}_1 - x_1^i)/\sigma_1 + \tfrac{1}{2}\right) - \beta^2\left((\widetilde{x}_1 - x_1^i)/\sigma_1 - \tfrac{1}{2}\right)\right] \cdot \beta^3\left((\widetilde{x}_2 - x_2^i)/\sigma_2\right)/\sigma_1, \tag{15}$$

$$b_{32}^i = \beta^3\left((\widetilde{x}_1 - x_1^i)/\sigma_1\right) \cdot \left[\beta^2\left((\widetilde{x}_2 - x_2^i)/\sigma_2 + \tfrac{1}{2}\right) - \beta^2\left((\widetilde{x}_2 - x_2^i)/\sigma_2 - \tfrac{1}{2}\right)\right]/\sigma_2, \tag{16}$$

$$\begin{aligned} b_{22}^i = &\left[\beta^2\left((\widetilde{x}_1 - x_1^i)/\sigma_1 + \tfrac{1}{2}\right) - \beta^2\left((\widetilde{x}_1 - x_1^i)/\sigma_1 - \tfrac{1}{2}\right)\right] \\ &\cdot \left[\beta^2\left((\widetilde{x}_2 - x_2^i)/\sigma_2 + \tfrac{1}{2}\right) - \beta^2\left((\widetilde{x}_2 - x_2^i)/\sigma_2 - \tfrac{1}{2}\right)\right]/(\sigma_1\sigma_2), \end{aligned} \tag{17}$$

$$b_{13}^i = \left[\beta^1\left((\widetilde{x}_1 - x_1^i)/\sigma_1 + 1\right) - 2\beta^1\left((\widetilde{x}_1 - x_1^i)/\sigma_1\right) + \beta^1\left((\widetilde{x}_1 - x_1^i)/\sigma_1 - 1\right)\right] \cdot \beta^3\left((\widetilde{x}_2 - x_2^i)/\sigma_2\right)/\sigma_1^2, \tag{18}$$

$$b_{31}^i = \beta^3\left((\widetilde{x}_1 - x_1^i)/\sigma_1\right) \cdot \left[\beta^1\left((\widetilde{x}_2 - x_2^i)/\sigma_2 + 1\right) - 2\beta^1\left((\widetilde{x}_2 - x_2^i)/\sigma_2\right) + \beta^1\left((\widetilde{x}_2 - x_2^i)/\sigma_2 - 1\right)\right]/\sigma_2^2. \tag{19}$$

Let $S = (3+1)^d$ be the number of control points in the support of the multi-dimensional third order B-spline. Then, we derive from the above equations:

$$\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{\mu}}(\widetilde{\boldsymbol{x}}) = \begin{bmatrix} b_{33}^0 & \cdots & b_{33}^{S-1} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & b_{33}^0 & \cdots & b_{33}^{S-1} \end{bmatrix}, \tag{20}$$

$$\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x}}(\widetilde{\boldsymbol{x}}) = \begin{bmatrix} 1 + \sum_i \mu_{i1} b_{23}^i & \sum_i \mu_{i1} b_{32}^i \\ \sum_i \mu_{i2} b_{23}^i & 1 + \sum_i \mu_{i2} b_{32}^i \end{bmatrix}, \tag{21}$$

$$\frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger}(\widetilde{\boldsymbol{x}}) = \left\{ \begin{bmatrix} \sum_i \mu_{i1} b_{13}^i & \sum_i \mu_{i1} b_{22}^i \\ \sum_i \mu_{i1} b_{22}^i & \sum_i \mu_{i1} b_{31}^i \end{bmatrix}, \begin{bmatrix} \sum_i \mu_{i2} b_{13}^i & \sum_i \mu_{i2} b_{22}^i \\ \sum_i \mu_{i2} b_{22}^i & \sum_i \mu_{i2} b_{31}^i \end{bmatrix} \right\} \tag{22}$$

$$\frac{\partial}{\partial \boldsymbol{\mu}} \frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x}}(\widetilde{\boldsymbol{x}}) = \left\{ \begin{bmatrix} b_{23}^0 & b_{32}^0 \\ 0 & 0 \end{bmatrix}, \cdots, \begin{bmatrix} b_{23}^{S-1} & b_{32}^{S-1} \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ b_{23}^0 & b_{32}^0 \end{bmatrix}, \cdots, \begin{bmatrix} 0 & 0 \\ b_{23}^{S-1} & b_{32}^{S-1} \end{bmatrix} \right\} \tag{23}$$

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\mu}} \frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger}(\widetilde{\boldsymbol{x}}) = &\left\{ \left\{ \begin{bmatrix} b_{13}^0 & b_{22}^0 \\ b_{22}^0 & b_{31}^0 \end{bmatrix}, O_{d\times d} \right\}, \cdots, \left\{ \begin{bmatrix} b_{13}^{S-1} & b_{22}^{S-1} \\ b_{22}^{S-1} & b_{31}^{S-1} \end{bmatrix}, O_{d\times d} \right\}, \right. \\ &\left. \left\{ O_{d\times d}, \begin{bmatrix} b_{13}^0 & b_{22}^0 \\ b_{22}^0 & b_{31}^0 \end{bmatrix} \right\}, \cdots, \left\{ O_{d\times d}, \begin{bmatrix} b_{13}^{S-1} & b_{22}^{S-1} \\ b_{22}^{S-1} & b_{31}^{S-1} \end{bmatrix} \right\} \right\}, \end{aligned} \tag{24}$$

where in (20), (23) and (24) we only show the nonzero derivatives. These derivatives are implemented in the `itk::AdvancedBSplineDeformableTransform` class. In addition, we needed some helper classes, such as the

`itk::BSplineInterpolationWeightFunctionBase`,
`itk::BSplineInterpolationDerivativeWeightFunction`,
`itk::BSplineInterpolationSecondOrderDerivativeWeightFunction`,

which implement the B-spline interpolation weights b^i from above. Furthermore, we changed some of the existing B-spline kernel functions and interpolators for performance enhancements:

`itk::BSplineInterpolationWeightFunction2`,
`itk::BSplineKernelFunction2`,
`itk::BSplineDerivativeKernelFunction2`,
`itk::BSplineSecondOrderDerivativeKernelFunction2`.

For example the `itk::BSplineKernelFunction2` omits the evaluation of if-statements, and the last two classes explicitly write out the relationship $\frac{\partial \beta^o}{\partial x}(x) = \frac{\partial \beta^{o-1}}{\partial x}(x+\frac{1}{2}) - \frac{\partial \beta^{o-1}}{\partial x}(x-\frac{1}{2})$.

5 Combining transformations

In the Insight Journal submission [4] we proposed the class `itk::CombinationTransform` for combining multiple transformations by addition or composition. Adding transformations is done via:

$$\boldsymbol{T}(\boldsymbol{x}) = \boldsymbol{T}_0(\boldsymbol{x}) + \boldsymbol{T}_1(\boldsymbol{x}) - \boldsymbol{x}, \tag{25}$$

where $\boldsymbol{T}_0(\boldsymbol{x})$ is the initial transformation and $\boldsymbol{T}_1(\boldsymbol{x})$ the current transformation. As explained in [4] only the current transformation is optimised during the registration. Composition of transformations is defined by:

$$\boldsymbol{T}(\boldsymbol{x}) = \boldsymbol{T}_1(\boldsymbol{T}_0(\boldsymbol{x})). \tag{26}$$

For these combined transformations we need to derive the relations for the (spatial) derivatives. Define $\boldsymbol{y} = \boldsymbol{T}_0(\widetilde{\boldsymbol{x}})$, then:

name	combo	formulae
Jacobian		
	add	$\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{\mu}}(\widetilde{\boldsymbol{x}}) = \frac{\partial}{\partial \boldsymbol{\mu}}(\boldsymbol{T}_0(\widetilde{\boldsymbol{x}}) + \boldsymbol{T}_1(\widetilde{\boldsymbol{x}}) - \widetilde{\boldsymbol{x}}) = \frac{\partial}{\partial \boldsymbol{\mu}}\boldsymbol{T}_1(\widetilde{\boldsymbol{x}})$
	compose	$\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{\mu}}(\widetilde{\boldsymbol{x}}) = \frac{\partial}{\partial \boldsymbol{\mu}}(\boldsymbol{T}_1(\boldsymbol{T}_0(\widetilde{\boldsymbol{x}}))) = \frac{\partial}{\partial \boldsymbol{\mu}}\boldsymbol{T}_1(\boldsymbol{y})$
SpatialJacobian		
	add	$\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x}}(\widetilde{\boldsymbol{x}}) = \frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{T}_0(\widetilde{\boldsymbol{x}}) + \boldsymbol{T}_1(\widetilde{\boldsymbol{x}}) - \widetilde{\boldsymbol{x}}) = \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{T}_0(\widetilde{\boldsymbol{x}}) + \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{T}_1(\widetilde{\boldsymbol{x}}) - \boldsymbol{I}$
	compose	$\frac{\partial T_k}{\partial x_i}(\widetilde{\boldsymbol{x}}) = \left(\frac{\partial T_{1,k}}{\partial \boldsymbol{x}}(\boldsymbol{y})\right)^\dagger \frac{\partial \boldsymbol{T}_0}{\partial x_i}(\widetilde{\boldsymbol{x}}) = \left(\frac{\partial \boldsymbol{T}_0}{\partial x_i}(\widetilde{\boldsymbol{x}})\right)^\dagger \frac{\partial T_{1,k}}{\partial \boldsymbol{x}}(\boldsymbol{y})$
JacobianOfSpatialJacobian		
	add	$\frac{\partial}{\partial \boldsymbol{\mu}}\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x}}(\widetilde{\boldsymbol{x}}) = \frac{\partial}{\partial \boldsymbol{\mu}}\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{T}_0(\widetilde{\boldsymbol{x}}) + \boldsymbol{T}_1(\widetilde{\boldsymbol{x}}) - \widetilde{\boldsymbol{x}}) = \frac{\partial}{\partial \boldsymbol{\mu}}\frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{T}_1(\widetilde{\boldsymbol{x}})$
	compose	$\frac{\partial}{\partial \boldsymbol{\mu}}\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x}}(\widetilde{\boldsymbol{x}}) = \frac{\partial}{\partial \boldsymbol{\mu}}\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{T}_1(\boldsymbol{T}_0(\boldsymbol{x}))) = \frac{\partial}{\partial \boldsymbol{\mu}}\frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{T}_1(\boldsymbol{y}) \cdot \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{T}_0(\widetilde{\boldsymbol{x}})$
SpatialHessian		
	add	$\frac{\partial^2 \boldsymbol{T}}{\partial x_i \partial x_j}(\widetilde{\boldsymbol{x}}) = \frac{\partial^2}{\partial x_i \partial x_j}(\boldsymbol{T}_0(\widetilde{\boldsymbol{x}}) + \boldsymbol{T}_1(\widetilde{\boldsymbol{x}}) - \widetilde{\boldsymbol{x}}) = \frac{\partial^2}{\partial x_i \partial x_j}\boldsymbol{T}_0(\widetilde{\boldsymbol{x}}) + \frac{\partial^2}{\partial x_i \partial x_j}\boldsymbol{T}_1(\widetilde{\boldsymbol{x}})$
	compose	$\frac{\partial^2 T_k}{\partial x_i \partial x_j}(\widetilde{\boldsymbol{x}}) = \left(\frac{\partial T_{1,k}}{\partial \boldsymbol{x}}(\boldsymbol{y})\right)^\dagger \frac{\partial^2 \boldsymbol{T}_0}{\partial x_i \partial x_j}(\widetilde{\boldsymbol{x}}) + \left(\frac{\partial \boldsymbol{T}_0}{\partial x_i}(\widetilde{\boldsymbol{x}})\right)^\dagger \frac{\partial^2 T_{1,k}}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger}(\boldsymbol{y}) \frac{\partial \boldsymbol{T}_0}{\partial x_j}(\widetilde{\boldsymbol{x}})$
JacobianOfSpatialHessian		
	add	$\frac{\partial}{\partial \boldsymbol{\mu}}\frac{\partial^2 \boldsymbol{T}}{\partial x_i \partial x_j}(\widetilde{\boldsymbol{x}}) = \frac{\partial}{\partial \boldsymbol{\mu}}\frac{\partial^2}{\partial x_i \partial x_j}(\boldsymbol{T}_0(\widetilde{\boldsymbol{x}}) + \boldsymbol{T}_1(\widetilde{\boldsymbol{x}}) - \widetilde{\boldsymbol{x}}) = \frac{\partial}{\partial \boldsymbol{\mu}}\frac{\partial^2}{\partial x_i \partial x_j}\boldsymbol{T}_1(\widetilde{\boldsymbol{x}})$
	compose	$\frac{\partial}{\partial \boldsymbol{\mu}}\frac{\partial^2 T_k}{\partial x_i \partial x_j}(\widetilde{\boldsymbol{x}}) = \frac{\partial^2 T_{1,k}}{\partial \boldsymbol{\mu} \partial \boldsymbol{x}^\dagger}(\boldsymbol{y}) \frac{\partial^2 \boldsymbol{T}_0}{\partial x_i \partial x_j}(\widetilde{\boldsymbol{x}}) + \left(\frac{\partial \boldsymbol{T}_0}{\partial x_i}(\widetilde{\boldsymbol{x}})\right)^\dagger \left(\frac{\partial}{\partial \boldsymbol{\mu}}\frac{\partial^2 T_{1,k}}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\dagger}(\boldsymbol{y})\right) \frac{\partial \boldsymbol{T}_0}{\partial x_j}(\widetilde{\boldsymbol{x}})$

These derivatives are implemented in the `itk::AdvancedCombinationTransform` class.

6 Bending energy penalty term

To showcase the use of these spatial derivatives we implemented the bending energy penalty term, which was defined as:

$$\mathcal{P}_{\mathrm{BE}}(\boldsymbol{\mu}) = \frac{1}{P}\sum_{\widetilde{\boldsymbol{x}}_i}\sum_{k,l,m=1}^{2}\left(\frac{\partial^2 T_k}{\partial x_l \partial x_m}(\widetilde{\boldsymbol{x}}_i)\right)^2. \tag{27}$$

We constructed an `itk::TransformBendingEnergyPenaltyTerm` which (finally) inherits from the `itk::ImageToImageMetric`. The `GetValue()`-method is implemented like:

```
if ( !transform->GetHasNonZeroSpatialHessian() ) return 0;

SpatialHessianType spatialHessian;
for all samples
  this->GetTransform()->GetSpatialHessian( sample, spatialHessian );
  for all k, l, m
    measure += spatialHessian[ k ][ l ][ m ]^2;
  end
end
measure /= numberOfSamples;
```

For the `GetValueAndDerivative()` we have:

$$\frac{\partial}{\partial \boldsymbol{\mu}}\mathcal{P}_{\mathrm{BE}}(\boldsymbol{\mu}) = \frac{1}{P}\sum_{\widetilde{\boldsymbol{x}}_i}\sum_{k,l,m=1}^{2} 2\frac{\partial^2 T_k}{\partial x_l \partial x_m}(\widetilde{\boldsymbol{x}}_i)\frac{\partial}{\partial \boldsymbol{\mu}}\frac{\partial^2 T_k}{\partial x_l \partial x_m}(\widetilde{\boldsymbol{x}}_i), \tag{28}$$

which is implemented like:

```
if ( !transform->GetHasNonZeroSpatialHessian()
  && !transform->GetHasNonZeroJacobianOfSpatialHessian() )
  value = 0; derivative = 0; return;

SpatialHessianType spatialHessian;
JacobianOfSpatialHessianType jacobianOfSpatialHessian;
for all samples
  this->GetTransform()->GetSpatialHessian( sample, spatialHessian );
  this->GetTransform()->GetJacobianOfSpatialHessian( sample,
    jacobianOfSpatialHessian, nonZeroJacobianIndices );
  for all nonZeroJacobianIndices, k, l, m
    derivative[ nonZeroJacobianIndices[ mu ] ] += 2.0
      * spatialHessian[ k ][ i ][ j ]
      * jacobianOfSpatialHessian[ mu ][ k ][ i ][ j ];
  end
end
derivative /= numberOfSamples;
```

It must be noted that the `itk::TransformBendingEnergyPenaltyTerm` class inherits from a derived version of the `itk::ImageToImageMetric` class, which adds support for image samplers. The image

sampling framework is described in another Insight Journal paper, see [7]. As such the compilation of this class will only succeed in combination with these 'enhanced' classes.

7 Discussion and Conclusion

This document describes the use and implementation of spatial derivatives of coordinate transformations in the ITK. In addition, the derivatives to the transformation parameters of these spatial derivatives are given, required for gradient descent like optimisation routines. The latter are implemented in a sparse manner. We created new versions of the identity, rigid, affine and B-spline transformation. Also, combinations of transformations benefit from the proposed enhancements.

The spatial derivatives were subsequently exploited by the bending energy penalty term. Their usage is, however, not limited to that penalty term, and many more penalty terms can be implemented using the new functionality.

References

[1] S. Klein, J. P. W. Pluim, M. Staring, and M. A. Viergever. Adaptive stochastic gradient descent optimisation for image registration. *International Journal of Computer Vision*, 81(3):227–239, 2009. 1

[2] S. Klein, M. Staring, K. Murphy, M.A. Viergever, and J.P.W. Pluim. elastix: a toolbox for intensity-based medical image registration. *IEEE Transactions on Medical Imaging*, 29(1):196 – 205, 2010. 1, 1

[3] S. Klein, M. Staring, and J. P. W. Pluim. Evaluation of optimization methods for nonrigid medical image registration using mutual information and B-splines. *IEEE Transactions on Image Processing*, 16(12):2879 – 2890, December 2007. 1

[4] Stefan Klein and Marius Staring. Combining transforms in ITK. *Insight Journal*, 2006. http://hdl.handle.net/1926/197. 2, 5, 5

[5] T. Rohlfing, C. R. Maurer Jr., D. A Bluemke, and M. A. Jacobs. Volume-preserving nonrigid registration of MR breast images using free-form deformation with an incompressibility constraint. *IEEE Transactions on Medical Imaging*, 22(6):730 – 741, 2003. 1

[6] D. Rueckert, L. I. Sonoda, and C. Hayes *et al.* Nonrigid registration using free-form deformations: Application to breast MR images. *IEEE Transactions on Medical Imaging*, 18(8):712 – 721, 1999. 1

[7] Marius Staring and Stefan Klein. An image sampling framework for the ITK. *Insight Journal*, 2010. http://hdl.handle.net/10380/3190. 6