Preprint for Journal of Real-Time Image Processing
The final publication is available at Springer via http://dx.doi.org/10.1007/s11554-016-0564-4

Jan Rühaak · Lars König · Florian Tramnitzke · Harald Köstler · Jan Modersitzki

A Matrix-Free Approach to Efficient Affine-Linear Image Registration on CPU and GPU

Preprint of January 19th, 2016

Abstract This paper presents a generic approach to highly efficient image registration in two and three dimensions. Both monomodal and multimodal registration problems are considered. We focus on the important class of affine-linear transformations in a derivative-based optimization framework.

Our main contribution is an explicit formulation of the objective function gradient and Hessian approximation that allows for very efficient, parallel derivative calculation with virtually no memory requirements. The flexible parallelism of our concept allows for direct implementation on various hardware platforms. Derivative calculations are fully matrix-free and operate directly on the input data, thereby reducing the auxiliary space requirements from $\mathcal{O}(n)$ to $\mathcal{O}(1)$.

The proposed approach is implemented on multicore CPU and GPU. Our GPU code outperforms a conventional matrix-based CPU implementation by more than two orders of magnitude, thus enabling usage in real-time scenarios. The computational properties of our approach are extensively evaluated, thereby demonstrating the performance gain for a variety of real-life medical applications.

J. Rühaak · L. König · F. Tramnitzke
Fraunhofer MEVIS, Maria-Goeppert-Str. 3, 23562 Lübeck, Germany
E-mail: jan.ruehaak@mevis.fraunhofer.de

L. König
E-mail: lars.koenig@mevis.fraunhofer.de

F. Tramnitzke
E-mail: florian.tramnitzke@mevis.fraunhofer.de

H. Köstler
Universität Erlangen-Nürnberg, Lehrstuhl für Systemsimulation, Cauerstr. 11, 91058 Erlangen, Germany
E-mail: harald.koestler@uni-erlangen.de

J. Modersitzki
Universität zu Lübeck, Institute of Mathematics and Image Computing, Maria-Goeppert-Str. 3, 23562 Lübeck, Germany
E-mail: jan.modersitzki@mic.uni-luebeck.de

1 Introduction

The task of image registration consists in computing the spatial correspondence between two or more images. It is ubiquitous in medical imaging: Whenever images from different modalities such as computed tomography (CT) and magnetic resonance imaging (MRI) or from different points in time need to be combined, spatial alignment of these images is required to fuse their information, i.e. an image registration problem has to be solved. A large amount of research has been devoted to the study of image registration algorithms in the last two decades, for an overview see the survey articles [5, 27, 51, 42] and references therein. Clinical applications are manifold and include improved reconstruction of nuclear images by cardiac and respiratory gating with subsequent registration [14], navigation support during surgical interventions [25], and three-dimensional reconstructions of histological serial sections [4], see Figure 1.

In many clinical use cases, the execution time of image registration algorithms is crucial. A clinical workflow can be greatly impeded if the staff has to wait for registration results, and some applications as e.g. tumor motion tracking for radiation treatment even require real-time capability [48]. Generally, due to the increasing number of clinically available image modalities with continually increasing image resolution, the demand for fast and efficient image registration solutions is likely to grow even further.

Consequently, various approaches have been proposed in the literature to reduce the computational costs of medical image registration algorithms. These can be classified into approaches that target the algorithmic structure of the chosen image registration model [17, 18, 13, 45] or use a particular hardware platform such as Graphics Processing Units (GPUs), Digital Signal Processors (DSPs) or Field Programmable Gate Arrays (FPGAs) for implementation [21, 2, 7].

In particular, the advent of general purpose programming frameworks [32, 44] for mainstream GPUs has trig-

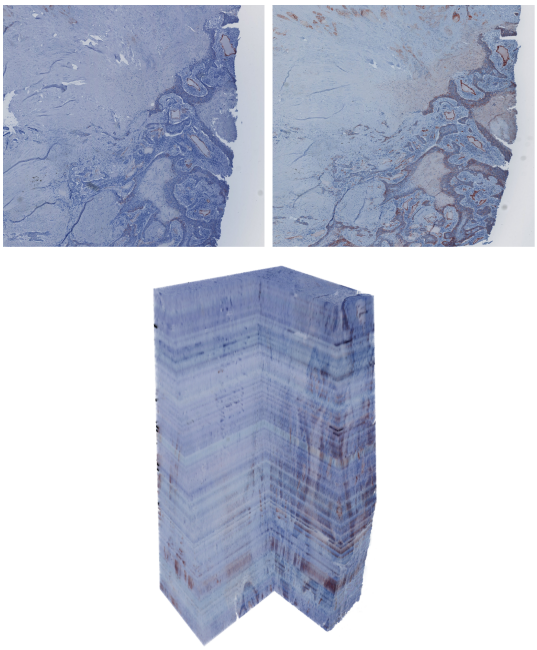


Fig. 1 Illustration of a 3D reconstruction of histological serial sections – ultra-thin slices created using specialized equipment and acquired with high resolution optical microscopes. Since the orientations of each slice on the microscope may vary, image registration is needed to correct the alignment and to reconstruct a three-dimensional representation of the sliced tissue sample. Image courtesy Johannes Lotz, Fraunhofer MEVIS, Lübeck.

gered a large number of publications that report on parallel implementations of various medical image processing algorithms including image registration, cf. [41, 39]. GPUs are very powerful computational units with comparably low costs [39], but require algorithms that can be executed in a massively parallel fashion to unlock their full potential. For many registration methods, a parallel formulation of the entire algorithm is unfortunately not straightforward [39]. For certain components such as transformation or interpolation, however, an exploitation of specific GPU features such as hardware-based linear interpolation [43] or texture caching for optimized access to neighboring voxels requires only few modifications and can already lead to massive performance boosts [40].

In this paper, we present a general algorithmic concept to efficient image registration that is directly suitable for implementation on both multi-core CPU and GPU. We employ a classical optimization-based image registration approach [29] using the Gauss-Newton algorithm. The key contribution of our paper is an explicit matrix-free formulation of the objective function gradient and Hessian approximation that allows for very efficient, parallel derivative calculation with virtually no memory requirements. We focus on the important class of affine-linear transformations that allow for rotation, translation, shearing and scaling and are widely used in practical image registration applications, see e.g. [37, 12, 36]. Moreover, almost all deformable image registration schemes rely on an affine-linear registration step for initialization [29, 30]. The costs for derivative calculation by far dominate the overall computational costs of affine-linear image registration and motivate the optimization of this particular step.

We study two distance measures, the Sum of Squared Differences (SSD) and the Normalized Gradient Fields (NGF, [16]), which serve as representatives for monomodal and multimodal image registration problems. In addition, the applicability of the proposed concept to alternative distance measures is discussed. The underlying idea of our approach has initially been presented in the conference paper [35] for the CPU. It has subsequently been successfully used for implementations of two-dimensional rigid and affine image registration algorithms on multicore DSPs [2] and GPUs [46] and was in a modified form applied to non-linear registration problems, see [22, 24]. In this work, the derivation of the new approach is explained in much greater detail together with an extended evaluation of its computational properties. Additionally, the GPU implementation is also presented for the three-dimensional case.

On the CPU, the new concept leads to a speed-up of more than one order of magnitude when compared to conventional matrix-based code. A further speed-up of another order of magnitude is gained by implementing the approach on the GPU. For both SSD and NGF, the registration performance scales well with increasing number of computational cores on a multi-core CPU. In contrast to the conventional matrix-based approach, the auxiliary space requirements for derivative computation are reduced from $\mathcal{O}(n)$ to $\mathcal{O}(1)$.

The paper is organized as follows. In Section 2, the employed optimization-based image registration concept is explained. Special focus is laid on the derivative calculations and the derivation of fully matrix-free computation rules. The realization on multi-core CPU and GPU is discussed afterwards. In Section 3, the potential of the proposed approach is extensively studied on both CPU and GPU. Real-world examples from different medical image registration problems underline the tremendous speed gain of the new concept. The paper proceeds with a discussion of the results in Section 4 and concludes with Section 5. The rather technical calculation of matrix-free computation rules for three-dimensional image registration are given in Appendix A.

2 Methods

The task of image registration is to compute the spatial correspondence between two or more images. Following [28], we denote the first image as *reference image* $\mathcal{R}$ and the second image as *template image* $\mathcal{T}$. We model images as functions $\mathcal{R} : \mathbb{R}^d \to \mathbb{R}$ and $\mathcal{T} : \mathbb{R}^d \to \mathbb{R}$, $d = 2, 3$, with compact support in domains $\Omega_{\mathcal{R}} \subseteq \mathbb{R}^d$ and

$\Omega_\mathcal{T} \subseteq \mathbb{R}^d$, respectively. With these definitions, a function $y : \Omega_\mathcal{R} \to \mathbb{R}^d$ is sought that maps points from the reference image domain $\Omega_\mathcal{R}$ onto corresponding locations in the template image domain $\Omega_\mathcal{T}$.

As discussed in the introduction, we focus on the important class of affine-linear transformations for y. An affine-linear transformation y is given by a matrix $A \in \mathbb{R}^{d\times d}$ and a vector $b \in \mathbb{R}^d$, such that $y(x) = Ax + b$. For convenience, we collect the coefficients of A and b in a vector $w \in \mathbb{R}^{6(d-1)}$, $d = 2, 3$, and denote the corresponding transformation by y_w. We express spatial correspondence by a so-called distance measure $\mathcal{D}$ measuring image similarity, cf. [28]. In accordance to common practice in affine-linear image registration, we do not perform explicit regularization though the problem is inherently ill-posed, see [29] for extended discussion. Setting $\mathcal{T}(y_w) := \mathcal{T} \circ y_w$ as the concatenation of functions, affine-linear image registration is modeled as optimization problem

$$\min_w \mathcal{D}\left(\mathcal{R}, \mathcal{T}(y_w)\right). \tag{1}$$

The choice of the distance measure $\mathcal{D}$ is fundamental for the registration algorithm, and various options have been proposed in the literature, see [27, 51, 42] and references therein for an overview. In this paper, we focus on the Sum of the Squared Differences (SSD) and the Normalized Gradient Fields (NGF) distance measures. The SSD distance measure is very versatile and has been successfully used in numerous applications, see e.g. [14, 20, 38]. Its basic assumption states that corresponding locations are characterized by common image intensity. In multimodal registration problems, however, this generally does not hold, and also some monomodal medical image registration tasks may violate this assumption, e.g. due to density changes on CT scans related to inflow of air in the lungs [30]. For such registration problems, we therefore consider the edge-based NGF distance measure [16] which is designed for multimodal image registration and exhibits favorable computational properties, see e.g. [34, 36, 25] for exemplary successful applications. In addition, the suitability of the proposed concept for arbitrary distance measures will be discussed.

For the numerical solution of the image registration problem (1), we employ the so-called discretize-optimize approach [15, 29]. The images are first discretized using regular grids, yielding a finite dimensional, continuous optimization problem. This enables the usage of Newton-type optimization schemes featuring super-linear convergence [3] for which well-established stopping criteria are available [31]. We choose the Gauss-Newton algorithm [3] that has been used in different image registration applications with great success [47, 14]. It is specifically designed for non-linear least squares problems as those occurring in image registration [3, 29]. We employ a multiresolution strategy ranging from coarse to fine, cf. [29].

The minimization of a discretized version of (1) with the Gauss-Newton algorithm requires the computation of the gradient and an approximation to the Hessian matrix of the objective function at each iteration step. Conventionally, the objective function associated with the image registration task is viewed as a concatenation of several functions that represent modules of the registration framework such as the interpolation scheme, transformation model and distance measure [29]. The derivative calculation is then performed individually for these functions, the final objective function derivatives are obtained by exploiting the chain rule for differentiation. As this computational approach entails the computation of several Jacobian matrices, we call it *matrix-based*.

The main advantage of the matrix-based approach lies in its great flexibility: All key components of the registration model can easily be exchanged and recombined without any changes to the overall scheme. From a computational point of view, however, the approach comes with a number of disadvantages both with regard to memory requirements and execution time. The individual Jacobian matrices have to be stored in memory, requiring memory for coefficients in an amount that linearly depends on the image size. Additionally, sparse matrix index administration requires further memory resources. The comparably large coefficient buffers are traversed repeatedly, thereby reducing cache efficiency, and the sparse matrix structures also challenge effective parallelization, see e.g. [6].

Fortunately, all these computational drawbacks can be eliminated by using a fully *matrix-free* approach for derivative calculation. The key idea is to break up the concept of first computing individual Jacobians for the various building blocks and then multiplying them to get the final objective function derivatives. Instead, the structure of the occurring matrices is beforehand analyzed in detail and subsequently exploited to arrive at a joint, explicit formulation of objective function gradient and Hessian approximation that does not contain any Jacobian matrices. The structure of the involved Jacobians is a priori exactly known for the SSD and NGF distance measures: only the coefficients depend on the actual images and the current transformation, not the sparsity patterns themselves. The matrices mostly exhibit diagonal bands and block structures, thus reducing the complexity of explicit calculation rules. Our matrix-free formulation directly allows for parallel computation with an up to pixelwise level of parallelism, reduces the auxiliary space requirements for derivative computation from $\mathcal{O}(n)$ to $\mathcal{O}(1)$ and leads to a tremendous overall speed-up of the entire registration algorithm.

In the following, we will first outline the employed optimization-based registration setting. Particular weight is laid on a fine-grained description of the objective function derivatives as these will be targeted by our reformulation. Hereupon, our alternative matrix-free computation scheme will be presented. The description is first given for the SSD distance measure and afterwards generalized to the more involved computations for NGF.

2.1 Sum of Squared Differences (SSD)

Our description of the registration framework and the derivative computations follows [29, 2]. We start by discussing the two-dimensional case. The calculations for three-dimensional images are along the same lines, but rather lengthy and technical, and are therefore given in full detail in Appendix A.1. For any transformation $y : \Omega_{\mathcal{R}} \to \mathbb{R}^2$, the Sum of Squared Differences (SSD) distance measure [28] is given by

$$\mathcal{D}_{\mathrm{SSD}}(\mathcal{R}, \mathcal{T}; y) := \frac{1}{2} \int_{\Omega_{\mathcal{R}}} \left(\mathcal{T}(y(\mathbf{x})) - \mathcal{R}(\mathbf{x})\right)^2 \mathrm{d}\mathbf{x}.$$

Let $y_w : \mathbb{R}^2 \to \mathbb{R}^2,\ x \mapsto Ax + b$ denote an affine-linear transformation with parameters $w = (w_1, \ldots, w_6)$ and

$$A := \begin{pmatrix} w_1 & w_2 \\ w_4 & w_5 \end{pmatrix},\ b := \begin{pmatrix} w_3 \\ w_6 \end{pmatrix}.$$

Setting $\mathcal{D}_{\mathrm{SSD}}(w) := \mathcal{D}_{\mathrm{SSD}}(\mathcal{R}, \mathcal{T}; y_w)$ yields the formulation of affine-linear image registration with SSD as minimization problem

$$\min_w\ \mathcal{D}_{\mathrm{SSD}}(w). \tag{2}$$

Note that $\mathcal{D}_{\mathrm{SSD}} : \mathbb{R}^6 \to \mathbb{R}$. In order to compute a numerical solution to the minimization problem (2), the continuous formulation is discretized. We assume the domain $\Omega_{\mathcal{R}}$ to be rectangular and decompose it into n cells of equal size with center points $\mathbf{x}_i,\ i = 1, \ldots, n$, arranged in lexicographical ordering. Using the midpoint quadrature rule for numerical integration, a discretized version of (2) is given by

$$\min_w\ D_{\mathrm{SSD}}(w) := \frac{\bar{h}}{2} \sum_{i=1}^{n} \left(\mathcal{T}(y_w(\mathbf{x}_i)) - \mathcal{R}(\mathbf{x}_i)\right)^2,$$

where $\bar{h}$ denotes the area of each cell. As the transformed coordinates $y_w(\mathbf{x}_i)$ will in general not coincide with template image cell-centered points, multilinear interpolation is used to evaluate the discrete template image at arbitrary coordinates. Since medical images typically exhibit zero background values, we apply Dirichlet zero boundary conditions.

The function D_{SSD} can be decomposed into a concatenation of vector-valued functions involving all n sampling points $\mathbf{x}_i$ at once, allowing for a direct calculation of the objective function derivatives. Let $(\mathbf{x}_i)_j$ denote the j-th component of $\mathbf{x}_i \in \mathbb{R}^2$. For transformation parameters $w \in \mathbb{R}^6$, we first define the vector

$$v_i := \begin{pmatrix} (A\mathbf{x}_1 + b)_i \\ (A\mathbf{x}_2 + b)_i \\ \vdots \\ (A\mathbf{x}_n + b)_i \end{pmatrix} \in \mathbb{R}^n,\ i = 1, 2,$$

and use it to construct the function

$$y : \mathbb{R}^6 \to \mathbb{R}^{2n},\ w \mapsto \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \tag{3}$$

which maps the parameters w to a vector of all n deformed sampling points, i.e. of $2n$ elements. Additionally, using $\mathbf{y}_i = (y_i, y_{i+n})^\top$, we define

$$T : \mathbb{R}^{2n} \to \mathbb{R}^n,\ \begin{pmatrix} y_1 \\ \vdots \\ y_{2n} \end{pmatrix} \mapsto \begin{pmatrix} \mathcal{T}(\mathbf{y}_1) \\ \vdots \\ \mathcal{T}(\mathbf{y}_n) \end{pmatrix}, \tag{4}$$

i.e. T evaluates the template image at every single of the n deformed points, thereby creating a vector of the n deformed template image intensities. Setting $R_i := \mathcal{R}(\mathbf{x}_i)$, we continue by formulating

$$r : \mathbb{R}^n \to \mathbb{R}^n,\ \begin{pmatrix} T_1 \\ \vdots \\ T_n \end{pmatrix} \mapsto \begin{pmatrix} T_1 - R_1 \\ \vdots \\ T_n - R_n \end{pmatrix}$$

as the vector-valued residual function and finally

$$\psi : \mathbb{R}^n \to \mathbb{R},\ \begin{pmatrix} r_1 \\ \vdots \\ r_n \end{pmatrix} \mapsto \frac{\bar{h}}{2} \sum_{i=1}^{n} r_i^2$$

as the sum of all squared residual elements. Now, the discrete objective function D_{SSD} can be written as a concatenation of four functions:

$$D_{\mathrm{SSD}} : \mathbb{R}^6 \xrightarrow{y} \mathbb{R}^{2n} \xrightarrow{T} \mathbb{R}^n \xrightarrow{r} \mathbb{R}^n \xrightarrow{\psi} \mathbb{R}. \tag{5}$$

2.1.1 Matrix-Based Differentiation

The benefit of the formulation (5) lies in a straightforward calculation of the *analytical* objective function gradient and the Gauss-Newton approximation to the Hessian [31, 11] using the chain rule as

$$\nabla D_{\mathrm{SSD}}(w) = \frac{\partial \psi}{\partial r} \frac{\partial r}{\partial T} \frac{\partial T}{\partial y} \frac{\partial y}{\partial w} \tag{6}$$

and

$$\nabla^2 D_{\mathrm{SSD}}(w) \approx \mathrm{d}r^\top \mathrm{d}_2 \psi \mathrm{d}r \tag{7}$$

using

$$\mathrm{d}r := \frac{\partial r}{\partial T} \frac{\partial T}{\partial y} \frac{\partial y}{\partial w} \in \mathbb{R}^{n \times 6}. \tag{8}$$

For convenience, we define the gradient as a row vector and denote the Gauss-Newton approximation to the

Hessian by H_{SSD}. The first two individual derivatives in (6) are given by

$$\frac{\partial \psi}{\partial r}[r] = \bar{h}(r_1, \ldots, r_n) \text{ and} \qquad (9)$$
$$\frac{\partial r}{\partial T}[T] = I_n,$$

with $I_n \in \mathbb{R}^{n \times n}$ as the identity matrix. This implies that the derivative of the residual function r can be omitted from the computations in the case of SSD. For the Normalized Gradient Fields distance measure, this is however not the case as will be discussed in Section 2.2.

Using the notation ∂_i for the partial derivative with respect to the i-th component and defining the n-by-n matrix $\partial_i \mathcal{T}[y]$ as

$$\partial_i \mathcal{T}[y] := \begin{pmatrix} \partial_i \mathcal{T}(\mathbf{y}_1) & & \\ & \ddots & \\ & & \partial_i \mathcal{T}(\mathbf{y}_n) \end{pmatrix}$$

for $i = 1, 2$, it holds that

$$\frac{\partial T}{\partial y}[y] = \begin{pmatrix} \partial_1 \mathcal{T} & \partial_2 \mathcal{T} \end{pmatrix} \in \mathbb{R}^{n \times 2n}. \qquad (10)$$

Finally, the derivative of the function y, which maps the parameters w to a transformed grid, is given by

$$\frac{\partial y}{\partial w}[w] = I_2 \otimes \mathbf{X} \in \mathbb{R}^{2n \times 6} \qquad (11)$$

with the Kronecker product $\otimes$ and the grid matrix $\mathbf{X}$ defined as

$$\mathbf{X} := \begin{pmatrix} (\mathbf{x}_1)_1 & (\mathbf{x}_1)_2 & 1 \\ (\mathbf{x}_2)_1 & (\mathbf{x}_2)_2 & 1 \\ \vdots & \vdots & \vdots \\ (\mathbf{x}_n)_1 & (\mathbf{x}_n)_2 & 1 \end{pmatrix} \in \mathbb{R}^{n \times 3},$$

thus completing the analysis of the gradient components from (6). See also Figure 2 for a schematic illustration of the derivative structure of the objective function gradient. The computation of the Gauss-Newton approximation to the Hessian (7) is finalized by noting $\mathrm{d}_2 \psi = \bar{h}$.

2.1.2 Matrix-Free Derivative Calculation

Based on the derivative presentation in the previous section, we now describe the first contribution of this paper, the formulation of explicit matrix-free computation rules for the objective function derivatives with SSD. The key component for efficient computation of the gradient (6) and the Hessian approximation (7) is the matrix product $\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}$ that occurs in both terms. Using (10) and (11), it follows that

$$\left(\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}\right)_{i,j} = \begin{cases} \partial_1 \mathcal{T}(\mathbf{y}_i)\mathbf{X}_{i,j} & 1 \le j \le 3 \\ \partial_2 \mathcal{T}(\mathbf{y}_i)\mathbf{X}_{i,j-3} & 4 \le j \le 6 \end{cases}. \qquad (12)$$

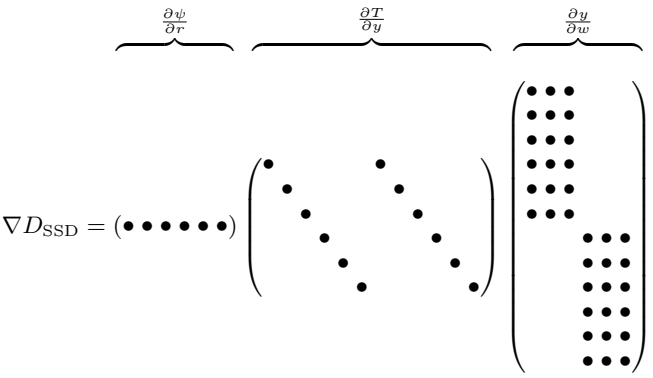


Fig. 2 Schematic view of the sparse matrix structure of the objective function gradient ∇D_{SSD} for affine-linear image registration with the SSD distance measure.

Since using (9)

$$\left(\frac{\partial \psi}{\partial r}\right)_i = \mathcal{T}_w(\mathbf{x}_i) - \mathcal{R}(\mathbf{x}_i)$$

with $\mathcal{T}_w(\mathbf{x}_i) := \mathcal{T}(\varphi_w(\mathbf{x}_i))$, a short calculation yields the explicit formula

$$\nabla D_{\text{SSD}}(w) = \bar{h} \sum_{i=1}^{n} (\mathcal{T}_w(\mathbf{x}_i) - \mathcal{R}(\mathbf{x}_i)) \begin{pmatrix} \partial_1 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_1 \\ \partial_1 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_2 \\ \partial_1 \mathcal{T}_w(\mathbf{x}_i) \\ \partial_2 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_1 \\ \partial_2 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_2 \\ \partial_2 \mathcal{T}_w(\mathbf{x}_i) \end{pmatrix}^{\top} \qquad (13)$$

for the objective function gradient. Note that each of the n summands can be computed independently, directly permitting a fully parallel computation with one final parallel reduction step. No sparse matrices are required any more, all coefficients can be directly computed from reference image, template image and transformation parameters w.

The Gauss-Newton approximation to the Hessian for the SSD distance measure is given by

$$H_{\text{SSD}} = \mathrm{d}r^{\top} \mathrm{d}_2 \psi \mathrm{d}r$$
$$= \bar{h} \left(\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}\right)^{\top} \left(\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}\right) \in \mathbb{R}^{6 \times 6}.$$

By utilizing (12), it can be computed in the same fashion as the objective function gradient. Setting

$$l_k := \left(\left(\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}\right)_{k,i} \cdot \left(\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}\right)_{k,j} \right)_{1 \le i,j \le 6}, \qquad (14)$$

it directly follows that

$$H_{\text{SSD}}(w) = \bar{h} \sum_{k=1}^{n} l_k.$$

Analogous to the gradient computation, the coefficient matrices can be computed independently, also allowing for a fully parallel computation with one final reduction step. Hence, the derivation of matrix-free computation rules for the SSD distance measure is complete. In the following, it will be shown how the above concept can be applied to the more involved derivative computations of the NGF distance measure.

2.2 Normalized Gradient Fields (NGF)

The Normalized Gradient Fields (NGF) distance measure [16] was proposed as an alternative to the computationally expensive Mutual Information [8, 49] for multimodal image registration tasks. In particular, NGF utilizes edge information and is designed to be well suited for numerical optimization [16, 29]. We discuss the two-dimensional case, the rather technical extension to three dimensions is given in Appendix A.2. We study a slightly modified variant $\mathcal{D}_{\mathrm{NGF}} := \mathcal{D}_{\mathrm{NGF}}(\mathcal{R}, \mathcal{T}; y)$ given by

$$\mathcal{D}_{\mathrm{NGF}} := \frac{1}{2} \int_{\Omega_{\mathcal{R}}} 1 - \left(\frac{\langle \nabla \mathcal{R}(\mathbf{x}), \nabla \mathcal{T}(y(\mathbf{x})) \rangle_{\varrho,\tau}}{\|\nabla \mathcal{R}(\mathbf{x})\|_{\varrho} \, \|\nabla \mathcal{T}(y(\mathbf{x}))\|_{\tau}} \right)^2 \mathrm{d}\mathbf{x}$$

with $\langle a, b \rangle_{\alpha,\beta} := \sum_{i=1}^{2} a_i b_i + \alpha\beta$, $a, b \in \mathbb{R}^2$, $\|a\|_{\varepsilon} := \sqrt{\sum_{i=1}^{2} a_i^2 + \varepsilon^2}$, cf. [35]. The edge parameters $\varrho, \tau > 0$ allow to distinguish between image edges and noise. In contrast to the original approach [16], the above formulation features separate edge parameters for reference and template image. Setting $\mathcal{D}_{\mathrm{NGF}}(w) := \mathcal{D}_{\mathrm{NGF}}(\mathcal{R}, \mathcal{T}; y_w)$, affine-linear image registration with the NGF distance measure translates to the minimization problem

$$\min_{w} \; \mathcal{D}_{\mathrm{NGF}}(w). \tag{15}$$

For numerical optimization, the continuous formulation in (15) needs to be discretized. Given a reference image of size $n_1 \times n_2$ and an index i, $i = 1, \ldots, n$, let $i', j' \in \mathbb{N}, 1 \leq i' \leq n_1$, $1 \leq j' \leq n_2$ such that $i = i' + j' n_1$. We define neighboring indices in x and y direction as

$$\begin{aligned} i_{-x} &= \max(i' - 1, 1) + j' n_1, \\ i_{+x} &= \min(i' + 1, n_1) + j' n_1, \\ i_{-y} &= i' + \max(j' - 1, 1) n_1, \\ i_{+y} &= i' + \min(j' + 1, n_2) n_1. \end{aligned} \tag{16}$$

Note that Neumann zero boundary conditions are used in order not to introduce artificial edges at the domain border. We further define functions

$$g_i : \mathbb{R}^n \to \mathbb{R}^2, \qquad T \mapsto \begin{pmatrix} \frac{1}{2h_1}(-T_{i_{-x}} + T_{i_{+x}}) \\ \frac{1}{2h_2}(-T_{i_{-y}} + T_{i_{+y}}) \end{pmatrix}$$

and

$$s_i : \mathbb{R}^2 \to \mathbb{R}, \qquad a \mapsto \langle g_i(R), a \rangle + \varrho\tau.$$

These functions are denoted g_i and s_i to indicate gradient and scalar product type operations at the position i, respectively. Further setting

$$n_{\varepsilon} : \mathbb{R}^2 \to \mathbb{R}, \qquad a \mapsto \sqrt{a_1^2 + a_2^2 + \varepsilon^2},$$

the discretized version of the minimization problem (15) is given by

$$\min_{w} \; D_{\mathrm{NGF}}(w) := \frac{\bar{h}}{2} \sum_{i=1}^{n} 1 - \left(\frac{s_i(g_i(T_w))}{n_{\varrho}(g_i(R)) \; n_{\tau}(g_i(T_w))} \right)^2$$

with $T_w \in \mathbb{R}^n$ as the deformed discrete template image, i.e. $(T_w)_i = \mathcal{T}(y_w(\mathbf{x}_i))$.

2.2.1 Matrix-Based Differentiation

Analogously to the SSD distance measure, the objective function is now decomposed into smaller parts for the computation of the derivatives. Let y and T as in (3) and (4). We define the residual function $r : \mathbb{R}^n \to \mathbb{R}^n$ by setting the i-th component function $r_i : \mathbb{R}^n \to \mathbb{R}$ to

$$r_i : T \mapsto \frac{s_i(g_i(T))}{n_{\varrho}(g_i(R)) \; n_{\tau}(g_i(T))}. \tag{17}$$

Finally, the reduction function $\psi : \mathbb{R}^n \to \mathbb{R}$ is given by

$$\psi(r) = \frac{\bar{h}}{2} \sum_{i=1}^{n} 1 - r_i^2.$$

Just as in the SSD case, the discretized objective function for affine-linear registration with the NGF distance measure can now be written as a concatenation of four functions:

$$D_{\mathrm{NGF}} : \mathbb{R}^6 \xrightarrow{y} \mathbb{R}^{2n} \xrightarrow{T} \mathbb{R}^n \xrightarrow{r} \mathbb{R}^n \xrightarrow{\psi} \mathbb{R}.$$

The derivatives of T and y have already been computed in (10) and (11). For the reduction function ψ, the derivative is given by

$$\frac{\partial \psi}{\partial r} = -\bar{h} r^{\top} \in \mathbb{R}^{1 \times n}. \tag{18}$$

The calculation of the residual function derivative $\frac{\partial r}{\partial T}$ is performed by differentiating the component functions r_i, $i = 1, \ldots, n$, using the quotient rule. The functions r_i from (17) are composed of the functions s_i, g_i and n_{ε} whose derivatives are given by

$$\frac{\partial s_i}{\partial a} = g_i(R)^{\top} \in \mathbb{R}^{1 \times 2},$$

$$\frac{\partial g_i}{\partial T} = \begin{array}{c} \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccccc} & & i_{-y} & & & i_{-x} & i & i_{+x} & & i_{+y} & \\ \end{array} \\ \begin{pmatrix} \cdots & 0 & \cdots & -\frac{1}{2h_1} & 0 & \frac{1}{2h_1} & \cdots & 0 & \cdots \\ \cdots & -\frac{1}{2h_2} & \cdots & 0 & 0 & 0 & \cdots & \frac{1}{2h_2} & \cdots \end{pmatrix} \end{array} \in \mathbb{R}^{2 \times n}$$

and

$$\frac{\partial n_\varepsilon}{\partial a} = \frac{1}{n_\varepsilon(a)} a^\top \in \mathbb{R}^{1\times 2}.$$

Applying the chain rule in both numerator and denominator of r_i yields

$$\frac{\partial r_i}{\partial T} = \begin{pmatrix} \vdots \\ \frac{1}{2h_2}\left[\frac{-g_i(R)_2}{n_\varrho(g_i(R))n_\tau(g_i(T))} + \frac{s_i(g_i(T))g_i(T)_2}{n_\varrho(g_i(R))n_\tau(g_i(T))^3}\right] \\ \vdots \\ \frac{1}{2h_1}\left[\frac{-g_i(R)_1}{n_\varrho(g_i(R))n_\tau(g_i(T))} + \frac{s_i(g_i(T))g_i(T)_1}{n_\varrho(g_i(R))n_\tau(g_i(T))^3}\right] \\ 0 \\ \frac{1}{2h_1}\left[\frac{g_i(R)_1}{n_\varrho(g_i(R))n_\tau(g_i(T))} - \frac{s_i(g_i(T))g_i(T)_1}{n_\varrho(g_i(R))n_\tau(g_i(T))^3}\right] \\ \vdots \\ \frac{1}{2h_2}\left[\frac{g_i(R)_2}{n_\varrho(g_i(R))n_\tau(g_i(T))} - \frac{s_i(g_i(T))g_i(T)_2}{n_\varrho(g_i(R))n_\tau(g_i(T))^3}\right] \\ \vdots \end{pmatrix}^\top \tag{19}$$

with the entries at positions i_{-y}, i_{-x}, i_{+x}, and i_{+y} (in that order) as defined in (16). Note that these positions may coincide, in which case the values are added. Finally, the Gauss-Newton approximation H_{NGF} to the Hessian is given by

$$H_{\mathrm{NGF}}(w) := \mathrm{d}r^\top \mathrm{d}_2\psi \mathrm{d}r \approx \nabla^2 D_{\mathrm{NGF}}(w)$$

with $\mathrm{d}r$ defined as in (8) and $\mathrm{d}_2\psi = -\bar{h}$. This finalizes the NGF derivative calculations, see also Figure 3 for an illustration of the sparse matrix pattern of the NGF residual function derivative in two and three dimensions, cf. Appendix A.2 for the three-dimensional case.

2.2.2 Matrix-Free Derivative Calculation

The second contribution of this paper, the derivation of fully matrix-free formulas for gradient and Hessian approximation for the NGF distance measure, is performed similarly as for SSD. The main difference lies in the residual derivative matrix $\frac{\partial r}{\partial T}$, which exhibits a banded structure (see (19) and Figure 3) as opposed to being the identity in the case of SSD. This leads to substantially more involved formulas. The final result, however, exhibits the same favorable computational properties as in the case of SSD. In particular, it can be computed in parallel with up to pixelwise parallelization level, and the computation does not require additional memory.

Setting $r_i := \frac{s_i(g_i(T))}{n_\varrho(g_i(R))\ n_\tau(g_i(T))}$ and $\mathrm{d}r_i := \frac{\partial r_i}{\partial T}\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}$, it holds with (18) that

$$\nabla D_{\mathrm{NGF}}(w) = -\bar{h}\sum_{i=1}^{n} r_i \mathrm{d}r_i. \tag{20}$$

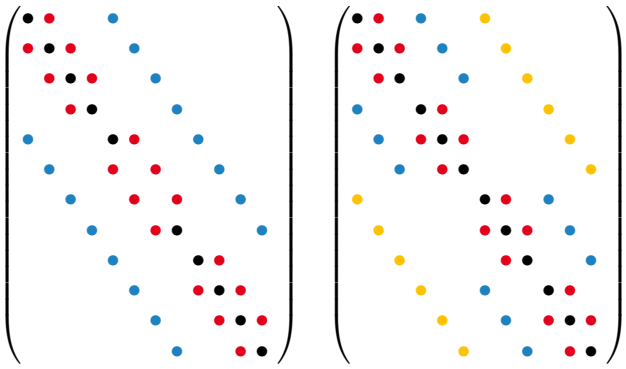


Fig. 3 Schematic view of the sparse matrix structure of $\frac{\partial r}{\partial T}$ for the NGF distance measure in two (left) and three dimensions (right). The residual matrix exhibits a banded structure with five diagonals (3D: seven) containing non-zero values. These correspond to the pixels themselves (main diagonal, black color), the neighbors in x direction (diagonals ± 1, red color), the neighbors in y direction (diagonals $\pm n_1$, blue color) and in the 3D case also the neighbors in z direction (diagonals $\pm n_1 n_2$, yellow color), cf. also Appendix A.2.

As $r_i \in \mathbb{R}$ are just scalars, it suffices to derive a matrix-free description of the vectors $\mathrm{d}r_i \in \mathbb{R}^6$ to arrive at a matrix-free formulation of the entire objective function gradient. Let $1 \le i \le n$ and define indices i_{-y}, i_{-x}, i_{+x}, and i_{+y} as in (16). Further, let $\partial r_i[j] := \left(\frac{\partial r_i}{\partial T}[T]\right)_j$. Since the product $\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}$ has already been discussed in (12), we are able to exploit the description in (19) and define coefficients $d_i^{j,k} \in \mathbb{R}$ as

$$\begin{aligned} d_i^{j,k} := {} & \partial r_i[i_{-y}]\partial_j \mathcal{T}(\mathbf{y}_{i_{-y}})\mathbf{X}_{i_{-y},k} \\ & + \partial r_i[i_{-x}]\partial_j \mathcal{T}(\mathbf{y}_{i_{-x}})\mathbf{X}_{i_{-x},k} \\ & + \partial r_i[i_{+x}]\partial_j \mathcal{T}(\mathbf{y}_{i_{+x}})\mathbf{X}_{i_{+x},k} \\ & + \partial r_i[i_{+y}]\partial_j \mathcal{T}(\mathbf{y}_{i_{+y}})\mathbf{X}_{i_{+y},k} \end{aligned}$$

for $i = 1, \dots, n$, $j = 1, 2$, and $k = 1, 2, 3$. This leads to the formulation of $\mathrm{d}r_i$ as

$$\mathrm{d}r_i = \left(d_i^{1,1}, d_i^{1,2}, d_i^{1,3}, d_i^{2,1}, d_i^{2,2}, d_i^{2,3}\right)^\top, \tag{21}$$

which implies that the objective function gradient (20) is given by

$$\nabla D_{\mathrm{NGF}}(w) = -\bar{h}\sum_{i=1}^{n} \frac{s_i(g_i(T))}{n_\varrho(g_i(R))\ n_\tau(g_i(T))} \begin{pmatrix} \mathrm{d}r_i[1] \\ \mathrm{d}r_i[2] \\ \vdots \\ \mathrm{d}r_i[6] \end{pmatrix}^\top. \tag{22}$$

A matrix-free formulation of the Gauss-Newton approximation to the Hessian is derived analogously. Since

$$H_{\mathrm{NGF}}(w) = \left(\frac{\partial r}{\partial T}\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}\right)^{\top} \mathrm{d}_2\psi \left(\frac{\partial r}{\partial T}\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}\right) = \left(\mathrm{d}r_1^{\top} \ \dots \ \mathrm{d}r_n^{\top}\right) \mathrm{d}_2\psi \begin{pmatrix} \mathrm{d}r_1 \\ \vdots \\ \mathrm{d}r_n \end{pmatrix},$$

the calculation of the Hessian approximation can directly be performed with the matrix-free representation of $\mathrm{d}r_i$ from (21). By defining matrices $l_k \in \mathbb{R}^{6\times 6}$ as

$$l_k := \left(\mathrm{d}r_k[i] \cdot \mathrm{d}r_k[j]\right)_{1\le i,j\le 6}, \tag{23}$$

the matrix-free formulation for the Gauss-Newton approximation to the Hessian is given by

$$H_{\mathrm{NGF}}(w) = \bar{h}\sum_{k=1}^{n} l_k.$$

This finalizes the derivation of matrix-free calculation rules for objective function gradient and Gauss-Newton approximation to the Hessian also for the Normalized Gradient Fields distance measure. The extension to the three-dimensional case is given in Appendix A.2.

2.2.3 Alternative Distance Measures

The proposed matrix-free computational approach is not restricted to the SSD and NGF distance measures, which were exemplarily discussed as representatives for monomodal and multimodal image registration. In the employed optimization-based concept, the choice of the distance measure $\mathcal{D}$ is completely independent of the transformation model and the interpolation scheme, in other words from the functions y and T. Hence, the matrix-free formulation of the matrix product $\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}$ from (12) can be utilized for arbitrary distance measures.

The suitability of the distance measure for a matrix-free computational approach depends on the derivative structure. Trivially, every matrix-based computation can in principle be performed in a matrix-free manner – if the matrices can be built up, all coefficients must be known, which implies that also a matrix-free operation involving these coefficients can be implemented. From a computational point of view, however, this is not always reasonable. Two properties of the SSD and NGF distance measures come in very beneficially: the structure of the residual derivative $\frac{\partial r}{\partial T}$ is independent of the images, and it is banded with only very few nonzero entries. In the case of SSD, the residual derivative is even the identity and can thus be omitted from the actual computations. If a distance measure exhibits a similar structure with few matrix bands or any other comparably simple structure, it can thus be considered suitable for the chosen approach.

Unfortunately, the popular distance measure Mutual Information [8, 49] does not fit directly into the matrix-free approach. The general integration of Mutual Information into the discretize-optimize framework using Parzen-window estimators is described in [29]. Coarsely speaking, the residual function r is replaced by a joint density estimator ρ, and the outer function ψ computes the Mutual Information from the density estimation. For the Gauss-Newton scheme, the derivative $\partial\rho$ of the joint density estimator is required.

The sparsity pattern of $\partial\rho$, however, depends on the image intensity distribution of the deformed template image $\mathcal{T}(y_w)$ and thus also on the current transformation parameters w. Hence, the pattern may change at each iteration step, which prohibits the derivation of a static matrix-free computation rule as in the case of SSD and NGF. Depending on the image intensity distribution, the matrix $\partial\rho$ may in addition be rather dense and will generally not exhibit a favorable structure such as block or band patterns. Hence, it cannot be considered suitable for efficient matrix-free implementation. However, a hybrid scheme using the matrix-free representation for $\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}$ and matrices for all other terms will still substantially reduce the memory requirements. For an alternative approach within a derivative-free optimization framework using a modified version of Powell's optimization method [33], see [40].

2.3 Computational Properties

The computational properties of the proposed matrix-free approach are now described at a general, implementation-independent level. Three major aspects are considered: Parallel computation, auxiliary space requirements for derivative calculation, and recalculations of coefficients of the Jacobian matrices during the matrix-free computation.

2.3.1 Parallel Computation

A major advantage of the new approach is the fully parallel computation of the objective function derivatives. The calculation rules for both gradient and Hessian approximation are essentially sums ranging over the reference image discretization points. As the individual summands can be computed independently of each other, the calculation can be performed with pixelwise level of parallelism, and the distribution of the summands to computational units is completely at the disposal of the developer. Hence, both architectures with comparably few computational cores such as the classic multicore CPU and massively parallel platforms such as the GPU can directly be addressed. The derivative calculation is by far

	$\frac{\partial \psi}{\partial r}$	$\frac{\partial r}{\partial T}$	$\frac{\partial T}{\partial y}$	$\frac{\partial y}{\partial w}$	Total
SSD, 2D	n	0	$2n$	$6n$	$9n$
NGF, 2D	n	$4n$	$2n$	$6n$	$13n$
SSD, 3D	n	0	$3n$	$12n$	$16n$
NGF, 3D	n	$6n$	$3n$	$12n$	$20n$

Table 1 Conventional matrix-based approach: auxiliary space requirements of the individual derivatives needed for computation of the objective function gradient and Hessian. Note that the Hessian computation does not require the derivative of ψ. The value n denotes the number of pixels or voxels of the reference image.

Algorithm	ψ	r	T	y
SSD, 2D, m-based	n	0	$2n$	$6n$
SSD, 2D, m-free	n	0	$6n$	$6n$
NGF, 2D, m-based	n	$5n$	$2n$	$6n$
NGF, 2D, m-free	n	$24n$	$24n$	$24n$
SSD, 3D, m-based	n	0	$3n$	$12n$
SSD, 3D, m-free	n	0	$12n$	$12n$
NGF, 3D, m-based	n	$7n$	$3n$	$12n$
NGF, 3D, m-free	n	$72n$	$72n$	$72n$

Table 2 Number of matrix coefficient calculations for objective function gradient computation with matrix-based and matrix-free approach. For Hessian computation, the coefficients for ψ are not required. The value n denotes the number of pixels or voxels of the reference image.

the computationally most expensive operation in affine-linear image registration. Hence, computing derivatives in parallel is virtually equivalent to parallelizing the entire image registration algorithm.

2.3.2 Auxiliary Space Requirements

The second benefit of the matrix-free approach is the reduction in auxiliary space requirements for derivative computation. For the matrix-based approach, let us assume a sparse matrix format such as the compressed sparse column format [9] is used for implementation. The auxiliary space requirements for the computation of gradient and Hessian approximation are given in Table 1, neglecting possible additional storage requirements for matrix index administration. For the SSD and the NGF distance measure, the auxiliary space requirement for computing the gradient and Hessian approximation is $\mathcal{O}(n)$ in two and three dimensions, with n denoting the number of discretization points (pixels/voxels) in the reference image. For images of size 512^3, e.g., this corresponds to a memory consumption of 16.0 GiB for SSD and 20.0 GiB for the NGF distance measure only for storing the values occurring in the individual derivatives at double precision. For the actual computation of the matrix product and the sparse matrix administration, additional memory may be required.

In the matrix-free case, the space requirements are given by the calculation rules for the derivatives. Here, the size of the summand vectors $\mathrm{d}r_i \in \mathbb{R}^{6(d-1)}$ for the gradient and the summand matrices $l_i \in \mathbb{R}^{6(d-1)\times 6(d-1)}$ for the Hessian approximation is independent of the image size. Thus, the memory requirements for both Hessian and gradient computation are constant, implying a reduction of the auxiliary space requirements from $\mathcal{O}(n)$ to $\mathcal{O}(1)$ as compared to the matrix-based approach.

2.3.3 Matrix Coefficient Recalculations

The matrix-free approach intentionally refrains from storing matrix coefficients even if they may be needed multiple times to reduce memory consumption and eliminate data structures that hinder parallel execution. Naturally, this may lead to recalculations of matrix coefficients. This effect is now analyzed in more detail.

We first consider the computation of the objective function gradient $\nabla D = \frac{\partial \psi}{\partial r}\frac{\partial r}{\partial T}\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}$. For the matrix-based case, the gradient computation requires to calculate each entry of the Jacobians of the four functions ψ, r, T and y exactly once, with the total number of coefficient calculations being determined by the matrix sparsity patterns. In the matrix-free case, however, some coefficients need to be recomputed multiple times. Using the explicit gradient representations (13) for SSD (3D: (35) in Appendix A.1) and (22) for NGF (3D: (42) in Appendix A.2), the exact number of matrix coefficient recomputations can directly be determined by counting the occuring matrix entries. See Table 2 for results and comparison to the matrix-based case.

The computation of the Hessian requires the matrix entries of $\frac{\partial r}{\partial T}$, $\frac{\partial T}{\partial y}$ and $\frac{\partial y}{\partial w}$. Since the explicit formulas (14) for the Hessian approximation for SSD (3D: (36) in Appendix A.1) and (23) for NGF (3D: (43) in Appendix A.2) require exactly the same matrix entries from $\frac{\partial r}{\partial T}$, $\frac{\partial T}{\partial y}$ and $\frac{\partial y}{\partial w}$ as in the case of the gradient, the number of matrix coefficient recalculations for the Hessian can be deduced from Table 2 by ignoring the column for ψ. While the matrix-free approach requires a higher number of matrix coefficient calculations for objective function gradient and Hessian approximation, the total number is $\mathcal{O}(n)$ in both approaches.

2.4 Implementation Remarks

The matrix-free computation rules for the SSD and NGF distance measure form the core of the proposed approach to increased efficiency of affine-linear image registration. For an actual implementation, several other characteristics of the registration problem can be taken into account, further improving efficiency.

2.4.1 General Observations

In the course of the optimization with the Gauss-Newton algorithm, the Hessian approximation H and the objective function gradient ∇f always have to be computed together for the solution of a linear equation system $Hp = -\nabla f$, which yields the descent direction p. As the term $\mathrm{d}r = \frac{\partial r}{\partial T}\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}$ occurs in both Hessian and gradient, it can be used for both together, substantially reducing the number of required floating point operations. It is therefore recommended to combine the pixelwise formulas for gradient and Hessian and compute both in one go. By exploiting the residual elements r_i, this procedure also yields the objective function value basically for free, which is required for evaluation of standard stopping criteria [31].

For the NGF distance measure, a symmetry in the terms $\frac{\partial r_i}{\partial T}$ can be exploited. Due to the employed finite difference scheme for image gradient calculation, it holds that $\partial r_i[i_{-x}] = -\partial r_i[i_{+x}]$, analogous also for i_{-y} and i_{-z}, respectively. Hence, half of the required calculations for the residual derivative can be substituted by just negating the already computed coefficients.

Finally, as the Hessian approximation is symmetric by definition, it suffices to only compute the upper triangular part and mirror the results along the main diagonal. This reduces the number of variables for which a parallel reduction needs to be performed, which is especially beneficial on the GPU platform with its large number of concurrent threads that each need storage for local result accumulation. In the 2D case, 28 reduction variables are needed (one for function value, six for gradient, 21 for the Hessian approximation), while the 3D case required 91 variables (1+12+78).

2.4.2 GPU Implementation

We have implemented the entire registration algorithm on the GPU using Nvidias Compute Unified Device Architecture (CUDA). CUDA offers a C-like development environment, is steadily updated and well documented. Furthermore, due to its specific development for Nvidia GPUs, it offers many high- and low-level features. Compared to Khronos Group's Open Computing Language (OpenCL, [44]), CUDA typically supports most features of new Nvidia GPUs and thus is faster in some cases [26, 41].

Since the proposed matrix-free algorithm is fully pixelwise parallelizable and has very low memory requirements, the GPU represents an optimal platform for implementation. Compared to CPUs, GPUs generally provide more computational power and thus can handle more floating-point operations per second [32]. The thousands of threads that can compute in parallel and the different memory spaces, each with their own advantages, give reason to expect very high computing performance. For the given registration algorithm, the challenging part of the GPU implementation is the parallel reduction step. Efficient ways need to be found to accumulate the various sums needed for the calculation of function value, gradient and approximation to the Hessian. In addition, the frequently occurring calculation of interpolated image values and especially image derivatives needs special attention.

The first problem, the parallel reduction step, is more grave, because the reduction of sums is not fully parallelizable, making it the main performance bottleneck on the GPU. This issue is generally well studied, we followed the ideas presented in [19, 50] closely. Extensive testing showed that for our algorithm the best reduction kernel is a combination of the various techniques presented in [19]. Instead of simply using the kernel with the allegedly best theoretical performance, our code benefited substantially from not unrolling the for loops.

The second problem, the calculation of image intensities and derivatives, is tackled with hardware-based interpolation as offered by the GPU. This feature is enabled by binding the images to so-called *texture memory*. Texture memory is cached on chip and can be accessed very fast. Moreover, the access is optimized for a 2D/3D read-out pattern and minimizes cache misses for addresses close in 2D/3D.

Instead of calculating an interpolation from known pixel values as common on the CPU, the interpolated pixel values can directly be fetched from the texture cache, thereby improving performance. Additionally, the analytical derivative can be computed with only a few texture fetches. In 2D, linear interpolation can be written as

$$\begin{aligned} p = {} & (1-r_1)((1-r_2)k_{00} + r_2 k_{01}) \\ & + r_1((1-r_2)k_{10} + r_2 k_{11}) \end{aligned} \tag{24}$$

where $k_{00} \cdots k_{11}$ are known pixel values and $r_i = x_i - \lfloor x_i \rfloor$, $i = 1, 2$, are remainders as illustrated in Figure 4. On the GPU, the computation in (24) is replaced by one single texture fetch $p = f(x_1, x_2)$ where f represents the texture fetch operation at coordinates (x_1, x_2).

The analytical derivative of (24) is given by

$$\begin{aligned} \frac{\partial p}{\partial x_1} &= (1-r_2)(k_{10} - k_{00}) + r_2(k_{11} - k_{01}), \\ \frac{\partial p}{\partial x_2} &= (1-r_1)(k_{01} - k_{00}) + r_1(k_{11} - k_{10}). \end{aligned}$$

Using textures, the derivative can directly be computed by the following four texture fetches:

$$\begin{aligned} \frac{\partial p}{\partial x_1} &= f(1, x_2) - f(0, x_2), \\ \frac{\partial p}{\partial x_2} &= f(x_1, 1) - f(x_1, 0). \end{aligned}$$

Thus, the interpolated pixel value and derivatives can be computed by just 9 operations (5 texture fetches, 2

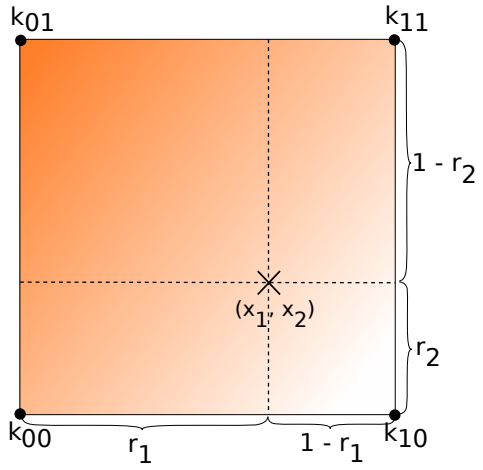


Fig. 4 Schematic representation of bilinear image interpolation at coordinates (x_1, x_2).

Function	Size	Serial	Parallel	Speed-up
SSD 2D	512^2	0.014 s	0.002 s	7.25
	4096^2	0.807 s	0.115 s	6.99
SSD 3D	64^3	0.034 s	0.006 s	5.52
	256^3	1.804 s	0.320 s	5.63
NGF 2D	512^2	0.052 s	0.007 s	7.57
	4096^2	3.485 s	0.482 s	7.22
NGF 3D	64^3	0.152 s	0.026 s	5.87
	256^3	9.762 s	1.505 s	6.49

Table 3 Runtimes for matrix-free SSD/NGF derivative calculation on the CPU in 2D and 3D for different image sizes. Gradient, Hessian approximation and objective function value were jointly computed. Timings were performed on a six-core CPU using hyper-threading. Times are given in seconds and averaged over ten executions.

subtractions, 2 memory writes). The conventional computation needs 48 operations (24 mathematical operations, 22 memory reads, 2 memory writes), not counting any preliminary calculations. Additionally, the four texture fetches needed for the derivative computation also benefit from a specialized 2D read-out pattern optimized for texture memory since they are close together in 2D. For the 3D case, the number of operations is reduced from 184 (104 memory reads, 76 mathematical operations, 4 memory writes) to 14 (7 texture fetches, 4 memory writes, 3 subtractions), thereby also making the code easier to understand and less error-prone.

For more detailed technical information on optimizing GPU code for affine-linear image registration, the reader is referred to [46].

3 Experiments

In this chapter, the computational properties of the proposed matrix-free approach to more efficient affine-linear image registration are analyzed in detail. A MATLAB implementation of the conventional matrix-based ansatz using the FAIR toolbox [29] serves as a reference. Although MATLAB is optimized for fast matrix computations, the code in FAIR was developed for research purposes and not explicitly designed for high computing performance. The achieved speed-up factors should therefore not be overrated.

Our matrix-free concept was implemented on CPU and GPU for the SSD and NGF distance measures in two and three dimensions. The CPU code was written in C++ and parallelized using OpenMP, conventional profiling tools were used for code optimization. For all experiments, the gcc 4.82 compiler was used. The GPU version was implemented using Nvidia's CUDA framework in version 6.5. All computations in this chapter were performed on a six-core Intel Xeon E5-2630 workstation equipped with 32 GB DDR3 RAM and running Ubuntu Linux 14.04. The GPU is an Nvidia GeForce GTX 980 graphics card with Maxwell architecture and compute capability 5.2. The theoretical peak performance and bandwidth of the CPU is 134 GFLOPS and 42.6 GB/s, respectively, compared to 4,612 GFLOPS and 224 GB/s for the GPU. Note that all CUDA timings include data transfer to the device and device initialization.

Our contribution targets parallel computation of the objective function derivatives since they form the computationally most expensive step in the considered affine-linear image registration setting. Consequently, our first experiment consists in computing the objective function gradient and Hessian approximation both serially (without OpenMP) and in parallel using our C++ implementation. Moreover, the computations were executed with a different number of active CPU cores to further examine scalability. The cores not required for a measurement were deactivated on operating system level.

In the second experiment, the derivative computations were performed with all three implementations. The goal of the experiment is to assess the possible speed-up by replacing matrix-based code with a matrix-free implementation, and further by using a GPU instead of a multicore CPU. Both SSD and NGF implementations were executed on images with different sizes.

Finally, the matrix-free scheme is embedded in a multilevel image registration to evaluate the benefits of the proposed approach in a medical real-world scenario. We consider two use cases: the registration of two-dimensional histological serial sections with the SSD distance measure and the multimodal registration of positron emission tomography (PET) and computed tomography (CT) scans using NGF.

Histological serial tumor sections [4], see Figure 7, are ultra-thin slices created using specialized equipment and acquired with high-resolution optical microscopes. Image registration is used to reconstruct a three-dimensional representation of the sliced tissue sample. The orienta-

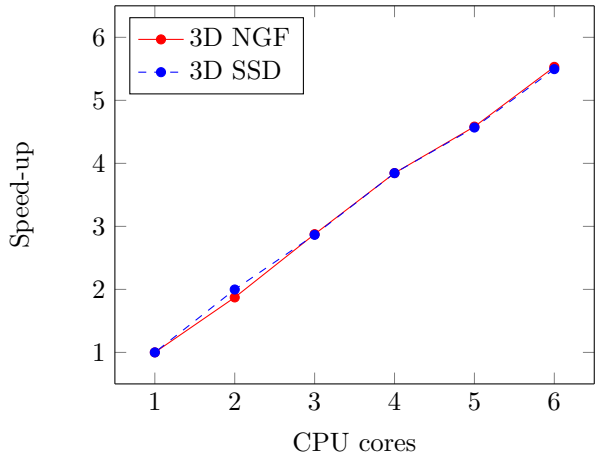


Fig. 5 Runtimes for matrix-free SSD/NGF derivative calculation for image size 256^3 with varying number of active CPU cores, evaluated on a 6-core CPU workstation. Hyper-threading was disabled for this experiment.

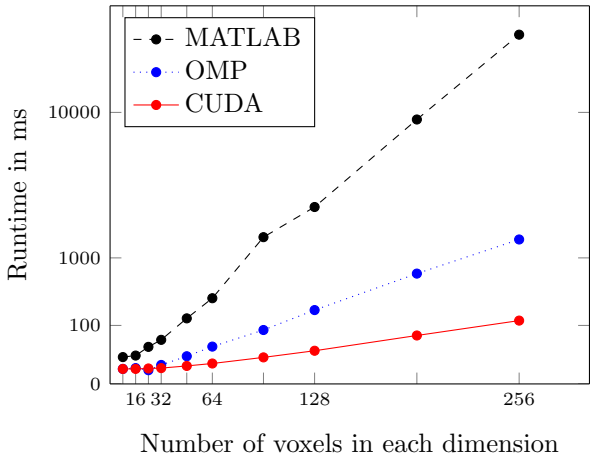


Fig. 6 Runtimes for NGF derivative calculation for different image sizes in 3D. Note y-axis is scaled in third root of runtime.

tions of each slice on the microscope often vary and image registration is needed to correct the alignment.

As a typical 3D example common in medical imaging, a PET/CT registration has been performed similar as in [35]. CT scans clearly depict the anatomy, while PET scans can provide valuable functional information about metabolic processes in the body, see e.g. [1]. By registering images from both modalities into one coordinate frame, the information can be fused, facilitating image interpretation and diagnosis. PET and CT images are shown in Figure 8.

3.1 Scalability on the CPU

The scalability of our implementation of the proposed matrix-free derivative calculation was evaluated for comparably small (512^2 in 2D, 64^3 in 3D) and for larger images (4096^2 in 2D, 256^3 in 3D). Gradient and Hessian approximation were computed both serially and in parallel. The computations were performed for both SSD and NGF in two and three dimensions, results of the experiments are given in Table 3. Executing the code with all six CPU cores and hyper-threading enabled leads to a speed-up of up to 7.57 for two-dimensional images and up to 6.49 in 3D. Furthermore, the code also scales well already for rather small images.

The scalability was further examined by using a different number of active CPU cores for computation. To focus on the gain by activating additional physical cores, hyper-threading was deactivated for this experiment. Execution times were measured for SSD and NGF in three dimensions and are shown in Figure 5. The speed-up increases linearly, depending on the number of active cores, and thus indicates excellent scalability on multicore systems. The same experiment with two dimensional images yielded analogous results and is thus not depicted here.

3.2 Comparison on Different Platforms

After the general scalability of the matrix-free schemes was studied in the previous section, the new algorithms for SSD and NGF are now executed on different platforms and compared to the conventional matrix-based approach. Gradient and Hessian approximation calculation were timed for SSD in 2D and for NGF with 3D images. These configurations were chosen according to the considered medical image registration tasks, the alignment of two-dimensional histological serial sections and the registration of three-dimensional PET and CT scans. For this experiment, timings were measured using randomly generated images of five different sizes for each implementation. Results are shown in Table 4 and Figure 6.

While the different implementations still exhibit the same time complexity and scale linearly depending on the total number of voxels, the speed-up for the different platforms can easily be seen. In general, the matrix-free CPU code with OpenMP outperforms the matrix-based, research-oriented MATLAB code for every image size. The CUDA code however needs large enough images to exceed the performance of the CPU code. This is due to initialization of the CUDA device and data transfer, which is included in the measurements. The CPU can directly access data in the memory whereas the images need to be transferred to the GPU before accessing them, resulting in an unavoidable overhead.

As the SSD distance measure calculations require relatively few arithmetic operations, the benefit of a GPU implementation is not as high as for NGF. The CUDA code performs the derivative computation for images of size 4096^2 in 36 ms, resulting in a speed-up of 3.2 compared to the OpenMP version. Again, a comparatively large amount of time is spent for transferring data to the device, explaining the low speed-up compared to the

Function	Size	MATLAB	OMP	CUDA	$\frac{\text{MATLAB}}{\text{OMP}}$	$\frac{\text{OMP}}{\text{CUDA}}$	$\frac{\text{MATLAB}}{\text{CUDA}}$
SSD 2D	256^2	25.78 ms	0.51 ms	1.43 ms	50.5	0.4	18.0
	512^2	94.33 ms	2.05 ms	2.15 ms	46.0	1.0	43.9
	1024^2	609.96 ms	7.10 ms	3.51 ms	85.9	2.0	173.8
	2048^2	4027.8 ms	28.78 ms	10.49 ms	140.0	2.7	384.0
	4096^2	12735 ms	115.61 ms	36.62 ms	110.2	3.2	347.8
NGF 3D	16^3	11.40 ms	1.99 ms	1.70 ms	5.7	1.2	6.7
	32^3	42.79 ms	3.41 ms	1.98 ms	12.5	1.7	21.6
	64^3	314.45 ms	25.98 ms	4.26 ms	12.1	6.1	73.8
	128^3	2765.9 ms	201.17 ms	18.23 ms	13.7	11.0	151.7
	256^3	21280 ms	1505.4 ms	126.88 ms	14.1	11.9	167.7

Table 4 Runtimes for SSD/NGF derivative calculation for different 2D and 3D image sizes. Times are given in milliseconds and averaged over ten executions. The three rightmost columns show speed-up factors between the three implementations.

OpenMP code. The effect of using the matrix-free computation scheme, however, is enormous and alone leads to a speed-up of two orders of magnitude.

Compared to SSD, NGF is more complex and involves more floating-point operations, promising a higher speed-up for the GPU implementation. For images of size 256^3, we were able to reduce the runtime for NGF derivative computation to merely 127 ms on the GPU from over 21 s using the MATLAB version and 1.5 s with OpenMP. For these images, the OpenMP code for the CPU is 14.1 times faster than MATLAB code from FAIR [29]. Furthermore, the CUDA implementation outperforms the OpenMP code, increasing the total speed-up by 11.9 to a total of 167.7. This underlines the tremendous speed gain of our GPU code for the NGF distance measure.

The different speed-up characteristics for SSD and NGF can be explained with the help of Table 2. For SSD, the number of matrix coefficient recalculations only moderately increases in the matrix-free case, while memory accesses are massively reduced. This allows for a large speed-up of more than two orders of magnitude when switching from the matrix-based to the matrix-free implementation on the CPU. When using the GPU platform, however, the speed-up of at most 3.2 to the multicore CPU is comparably low and close to the difference in memory throughput, indicating that at least parts of the algorithm are memory bound and can thus not profit from the increased computational power.

For NGF, the matrix-free approach leads to a much larger increase in the number of matrix coefficient recomputations. Since memory accesses are however drastically reduced at the same time, a speed-up of about one order of magnitude can still be realized on the CPU. The GPU platform with its high computational power is able to accelerate the algorithm by another order of magnitude, suggesting at least parts of the algorithm are compute bound.

Summarizing, by applying the matrix-free concept, the calculation time was substantially decreased for both SSD and NGF, while the auxiliary memory consumption was at the same time reduced from $\mathcal{O}(n)$ to $\mathcal{O}(1)$.

Size	MATLAB	OMP	CUDA
256^2	1.220 s	0.010 s	0.013 s
512^2	4.451 s	0.036 s	0.021 s
1024^2	12.837 s	0.094 s	0.028 s
2048^2	50.514 s	0.354 s	0.057 s
4096^2	159.25 s	1.734 s	0.184 s

Table 5 Runtimes of a complete registration in 2D with SSD using the histological serial section images. Times are given in seconds and are averaged over 10 executions.

Size	$\frac{\text{MATLAB}}{\text{OMP}}$	$\frac{\text{OMP}}{\text{CUDA}}$	$\frac{\text{MATLAB}}{\text{CUDA}}$
256^2	122.0	0.8	93.8
512^2	123.6	1.7	212.0
1024^2	136.6	3.4	458.5
2048^2	142.7	6.2	886.2
4096^2	91.8	9.4	865.5

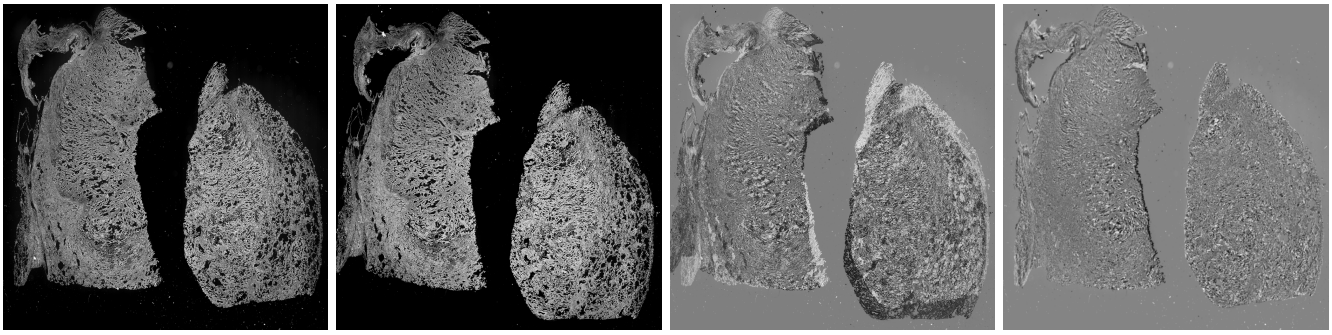
Table 6 Speed-up factors of a complete registration in 2D with SSD using the histological serial section images, based on Table 5.

3.3 Registration Performance

In the previous sections, it has been assured that the proposed schemes for matrix-free computation of gradient and Hessian approximation exhibit favorable scalability and increase performance. Excellent scalability was shown for the OpenMP code on the CPU, in addition, the GPU implementation featured superior computation speed. Now, the derivative calculations are embedded into a complete multilevel image registration algorithm on all three platforms and used for the registration of real-world medical images.

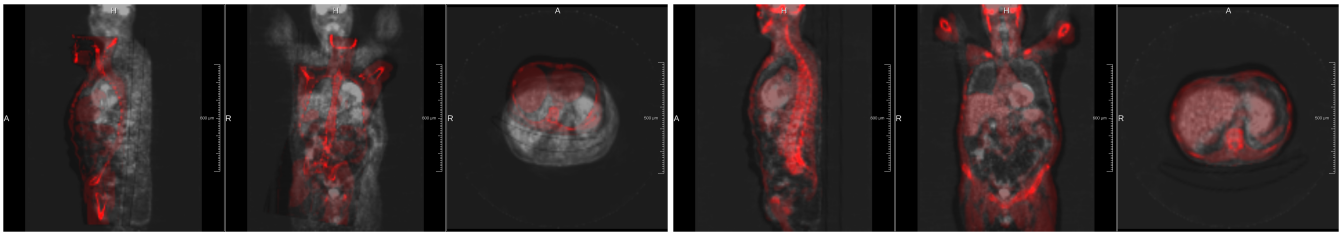
The registration results for the two-dimensional histological serial sections are shown in Figure 7 and for the three-dimensional PET-CT scans in Figure 8. All three implementations of the registration algorithm yield visually satisfying results which are numerically identical.

However, the runtimes of the implementations differ tremendously, see Tables 5 and 7 for absolute runtimes



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

Fig. 7 Registration of histological serial section images: reference (a) and template image (b); difference image before (c) and after (d) registration.



(a) (b)

Fig. 8 PET-CT registration: reference (PET) and template (CT) image before (a) and after (b) registration in sagittal, coronal and transversal view. The CT image is displayed in red.

Size	MATLAB	OMP	CUDA
16^3	0.897 s	0.015 s	0.028 s
32^3	1.340 s	0.035 s	0.038 s
64^3	4.165 s	0.112 s	0.048 s
128^3	28.140 s	0.634 s	0.077 s
256^3	219.70 s	5.342 s	0.329 s

Table 7 Runtimes of a complete registration in 3D with NGF using the PET-CT images. Times are given in seconds and are averaged over 10 executions.

Size	$\frac{\text{MATLAB}}{\text{OMP}}$	$\frac{\text{OMP}}{\text{CUDA}}$	$\frac{\text{MATLAB}}{\text{CUDA}}$
256^2	59.8	0.5	32.0
512^2	38.3	0.9	35.3
1024^2	37.2	2.3	86.8
2048^2	44.4	8.2	365.5
4096^2	41.1	16.2	667.8

Table 8 Speed-up factors of a complete registration in 3D with NGF using the PET-CT images, based on Table 7.

and Tables 6 and 8 for speed-up factors. Employing our CUDA implementation, we were able to perform a 2D SSD registration of two serial sections of size 4096^2 in 184 ms. The code was considerably slower on the other platforms with a runtime of 1.7 s for the OpenMP code and over 2.5 min using MATLAB. This results in a total speed-up of 9.4 for the matrix-free GPU implementation compared to the OpenMP version. Compared to the matrix-based approach, the application of the matrix-free concept again leads to a speed-up of two orders of magnitude.

Similar behavior can be seen for 3D image registration using the NGF distance measure. Here, the total runtime for the largest image was decreased from over 3.5 min in MATLAB to 5.3 s using OpenMP. Once more, the GPU based CUDA implementation is the fastest routine with a runtime of only 329 ms for an image size of 256^3, resulting in a speed-up of 16.2 compared to the OpenMP CPU version. In comparison to the research-oriented, matrix-based implementation with FAIR, the achieved speed-up of the CUDA code is > 600.

As stated before, FAIR is not particularly optimized for performance in terms of execution speed, but represents a typical class of "research prototype" implementation level. Similar to this, the other platforms chosen for comparison also represent typical classes of implementation levels: parallelized C++ code, representing typical multi-purpose product level implementations, and GPU code representing highly specialized code for powerful many-core architectures. Obviously, these three exemplary platforms cannot cover the full range of possible targets, platforms and high-performance techniques. However, representing typical use cases, the results may give some insight onto expectable speed-ups for typical

scenarios and show to which extent research code can benefit from specialized implementations on massively parallel platforms.

4 Discussion

All achieved speed-ups lie well within our expectations. The theoretical peak performance and bandwidth, as listed in Section 3, allow for a theoretical peak speed-up of 34.4, if the entire algorithm is completely compute bound. Yet, due to many parallel reductions that are inherently inefficient on many-core architectures and the fact that many parts of the implementation are limited by memory bandwidth, an inevitable performance loss occurs, which impairs the overall speed-up. Furthermore, the theoretical peak values are hardly ever realizable as the numerical algorithm prevents an optimal processor load.

The results obtained in Section 3 clearly emphasize the high potential of applying high performance computing techniques to medical image registration. In comparison to the traditional matrix-based computation scheme, our GPU implementation of the proposed approach yields a speed-up of more than two orders of magnitude. Together with the drastically decreased memory usage, the matrix-free approach allows for usage of derivative-based affine-linear image registration in new areas where strict memory and runtime requirements must be met.

A potential drawback of the proposed approach is the reduced flexibility in a practical implementation. In the traditional matrix-based scheme, the components of the registration such as distance measure, transformation model or interpolation scheme are easily interchangeable as they serve as independent building blocks, and their derivatives are computed separately. In the matrix-free case, the derivative computation is fused over all building blocks, thereby intentionally breaking up the modular structure to improve computational performance. On the other hand, also in the matrix-free case existing components such as interpolation and translation can be reused if, e.g., only the distance measure is exchanged, cf. Section 2.2.3. Concluding, it is fair to state that preserving maintainability and flexibility within the matrix-free concept is significantly more challenging for the developer than it is in the matrix-based case.

While this paper exclusively covers affine-linear image registration, the presented ideas can be utilized to derive similar matrix-free computation rules also for non-linear image registration algorithms. Initial results from [22, 24] are very encouraging and enabled the successful usage of non-linear image registration in a real-time 2D vessel tracking application [10, 23]. Hence, there is every reason to believe that the proposed approach has the potential to yield substantial computational improvements in a large number of different medical image registration tasks.

5 Conclusions

In this paper, we have studied the efficiency of affine-linear image registration. A classical optimization-based registration approach was analyzed in detail for representative distance measures suitable for monomodal and multimodal image registration. The computationally by far most expensive part, the calculation of objective function gradient and Gauss-Newton approximation to the Hessian, was reformulated in a fully matrix-free manner. The derived formulas allow for parallel computation with excellent scalability, auxiliary space requirements of the derivative calculations were reduced from $\mathcal{O}(n)$ to $\mathcal{O}(1)$.

With the proposed matrix-free approach and code that uses the advantages offered by the GPU platform, it is possible to achieve registration results in just a few hundredths of a second for medium-sized medical images. Depending on the image size and required frames per second, our approach can thus be used in a real-time image registration setting, thereby opening up new applications in the clinic.

Acknowledgements J. Rühaak, L. König, F. Tramnitzke and J. Modersitzki received funding from the European Union, European Regional Development Fund, grant no. 122-10-002. All authors declare that they have no conflicts of interest.

References

1. Alavi A, et al (2007) Is PET-CT the only option? European Journal of Nuclear Medicine and Molecular Imaging 34:819–821
2. Berg R, König L, Rühaak J, Lausen R, Fischer B (2014) Highly efficient image registration for embedded systems using a distributed multicore DSP architecture. Journal of Real-Time Image Processing (in press)
3. Björck A (1996) Numerical methods for least squares problems. SIAM, Philadelphia
4. Bronsert P, Enderle-Ammour K, Bader M, Timme S, Kuehs M, Csanadi A, Kayser G, Kohler I, Bausch D, Hoeppner J, et al (2014) Cancer cell invasion and EMT marker expression: a three-dimensional study of the human cancer–host interface. The Journal of Pathology 234(3):410–422
5. Brown LG (1992) A survey of image registration techniques. ACM Computing Surveys (CSUR) 24(4):325–376
6. Buluc A, Gilbert JR (2012) Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. SIAM Journal on Scientific Computing 34(4):C170–C191
7. Castro-Pareja CR, Jagadeesh JM, Shekhar R (2003) FAIR: a hardware architecture for real-time 3-D image registration. Information Technology in Biomedicine, IEEE Transactions on 7(4):426–434

8. Collignon A, Maes F, Delaere D, Vandermeulen D, Suetens P, Marchal G (1995) Automated multimodality image registration based on information theory. In: Information Processing in Medical Imaging, vol 3, pp 264–274
9. Davis TA (2006) Direct methods for sparse linear systems, vol 2. SIAM, Philadelphia
10. De Luca V, Benz T, Kondo S, König L, Lübke D, Rothlübbers S, Somphone O, Allaire S, Bell ML, Chung D, et al (2015) The 2014 liver ultrasound tracking benchmark. Physics in Medicine and Biology 60(14):5571
11. Dennis Jr JE, Schnabel RB (1996) Numerical methods for unconstrained optimization and nonlinear equations, vol 16. SIAM, Philadelphia
12. Ferroli P, Franzini A, Marras C, Maccagnano E, D'Incerti L, Broggi G (2004) A simple method to assess accuracy of deep brain stimulation electrode placement: pre-operative stereotactic CT + post-operative MR image fusion. Stereotactic and Functional Neurosurgery 82(1):14–19
13. Fischer B, Modersitzki J (2002) Fast diffusion registration. Contemporary Mathematics 313:117–128
14. Gigengack F, Ruthotto L, Burger M, Wolters CH, Jiang X, Schafers KP (2012) Motion correction in dual gated cardiac PET using mass-preserving image registration. Medical Imaging, IEEE Transactions on 31(3):698–712
15. Haber E, Modersitzki J (2006) A multilevel method for image registration. SIAM Journal on Scientific Computing 27(5):1594–1607
16. Haber E, Modersitzki J (2007) Intensity gradient based registration and fusion of multi-modal images. Methods of Information in Medicine 46:292–9
17. Haber E, Heldmann S, Modersitzki J (2007) An octree method for parametric image registration. SIAM Journal on Scientific Computing 29(5):2008–2023
18. Haber E, Heldmann S, Modersitzki J (2008) Adaptive mesh refinement for nonparametric image registration. SIAM Journal on Scientific Computing 30(6):3012–3027
19. Harris M, et al (2007) Optimizing parallel reduction in CUDA. NVIDIA Developer Technology 2(4)
20. Kabus S, Lorenz C (2010) Fast elastic image registration. Medical Image Analysis for the Clinic: A Grand Challenge pp 81–89
21. Köhn A, Drexl J, Ritter F, König M, Peitgen HO (2006) GPU accelerated image registration in two and three dimensions. In: Bildverarbeitung für die Medizin 2006, Springer, pp 261–265
22. König L, Rühaak J (2014) A fast and accurate parallel algorithm for non-linear image registration using normalized gradient fields. In: Biomedical Imaging (ISBI), 2014 IEEE 11th International Symposium on, pp 580–583
23. König L, Kipshagen T, Rühaak J (2014) A non-linear image registration scheme for real-time liver ultrasound tracking using normalized gradient fields. In: Proc. MICCAI Challenge on Liver Ultrasound Tracking (CLUST 2014)
24. König L, Derksen A, Hallmann M, Papenberg N (2015) Parallel and memory efficient multimodal image registration for radiotherapy using normalized gradient fields. In: Biomedical Imaging (ISBI), 2015 IEEE 12th International Symposium on
25. Lange T, Papenberg N, Heldmann S, Modersitzki J, Fischer B, Lamecker H, Schlag PM (2009) 3D ultrasound-CT registration of the liver using combined landmark-intensity information. International Journal of Computer Assisted Radiology and Surgery 4(1):79–88
26. Lombardi F, Spigler R (2014) The evolution of the approach to scientific computing: a survey. Journal of Parallel & Cloud Computing 3(2):32–42
27. Maintz J, Viergever MA (1998) A survey of medical image registration. Medical Image Analysis 2(1):1–36
28. Modersitzki J (2004) Numerical methods for image registration. Oxford University Press
29. Modersitzki J (2009) FAIR: Flexible algorithms for image registration, vol 6. SIAM, Philadelphia
30. Murphy K, Van Ginneken B, Reinhardt JM, Kabus S, Ding K, Deng X, Cao K, Du K, Christensen GE, Garcia V, et al (2011) Evaluation of registration methods on thoracic CT: the EMPIRE10 challenge. Medical Imaging, IEEE Transactions on 30(11):1901–1920
31. Nocedal J, Wright S (1999) Numerical optimization. Springer
32. NVIDIA Corporation (2014) NVIDIA CUDA C programming guide
33. Powell MJ (1964) An efficient method for finding the minimum of a function of several variables without calculating derivatives. The Computer Journal 7(2):155–162
34. Rühaak J, Heldmann S, Kipshagen T, Fischer B (2013) Highly accurate fast lung CT registration. In: SPIE Medical Imaging 2013, Image Processing, pp 86,690Y–86,690Y–9
35. Rühaak J, König L, Hallmann M, Papenberg N, Heldmann S, Schumacher H, Fischer B (2013) A fully parallel algorithm for multimodal image registration using normalized gradient fields. In: Biomedical Imaging (ISBI), 2013 IEEE 10th International Symposium on, pp 572–575
36. Rühaak J, Derksen A, Heldmann S, Hallmann M, Meine H (2015) Accurate CT-MR image registration for Deep Brain Stimulation: a multi-observer evaluation study. In: SPIE Medical Imaging 2015: Image Processing
37. Salas Gonzalez D, Górriz J, Ramírez J, Lassl A, Puntonet C (2008) Improved Gauss-Newton optimisa-

tion methods in affine registration of SPECT brain images. Electronics Letters 44(22):1291–1292
38. Schmitt O, Modersitzki J, Heldmann S, Wirtz S, Fischer B (2007) Image registration of sectioned brains. International Journal of Computer Vision 73(1):5–39
39. Shams R, Sadeghi P, Kennedy R, Hartley R (2010) A survey of medical image registration on multicore and the GPU. Signal Processing Magazine, IEEE 27(2):50–60
40. Shams R, Sadeghi P, Kennedy R, Hartley R (2010) Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images. Computer Methods and Programs in Biomedicine 99(2):133–146
41. Shi L, Liu W, Zhang H, Xie Y, Wang D (2012) A survey of GPU-based medical image computing techniques. Quantitative Imaging in Medicine and Surgery 2(3):188
42. Sotiras A, Davatzikos C, Paragios N (2013) Deformable medical image registration: a survey. Medical Imaging, IEEE Transactions on 32(7):1153–1190
43. Soza G, Bauer M, Hastreiter P, Nimsky C, Greiner G (2002) Non-rigid registration with use of hardware-based 3D Bézier functions. In: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2002, Springer, pp 549–556
44. Stone JE, Gohara D, Shi G (2010) OpenCL: A parallel programming standard for heterogeneous computing systems. Computing in Science & Engineering 12(3):66
45. Stürmer M, Köstler H, Rüde U (2008) A fast full multigrid solver for applications in image processing. Numerical Linear Algebra with Applications 15(2-3):187–200
46. Tramnitzke F, Rühaak J, König L, Modersitzki J, Köstler H (2014) GPU based affine linear image registration using normalized gradient fields. In: Proc. Seventh International Workshop on High Performance Computing for Biomedical Image Analysis (HPC-MICCAI)
47. Vercauteren T, Pennec X, Perchant A, Ayache N (2009) Diffeomorphic demons: Efficient non-parametric image registration. NeuroImage 45(1):S61–S72
48. Verma PS, Wu H, Langer MP, Das IJ, Sandison G (2011) Survey: real-time tumor motion prediction for image-guided radiation treatment. Computing in Science & Engineering 13(5):24–35
49. Viola P, Wells III WM (1997) Alignment by maximization of mutual information. International Journal of Computer Vision 24(2):137–154
50. Wilt N (2013) The CUDA handbook: a comprehensive guide to GPU programming. Pearson Education
51. Zitova B, Flusser J (2003) Image registration methods: a survey. Image and Vision Computing 21(11):977–1000

A Extension to the Three-Dimensional Case

In this appendix, explicit matrix-free calculation rules will be derived for affine-linear registration of three-dimensional images with the SSD and NGF distance measures. Most definitions of the occuring functions are briefly repeated here to improve readability.

A.1 Sum of Squared Differences (SSD)

For any $y : \Omega_{\mathcal{R}} \to \mathbb{R}^3$, the Sum of Squared Differences (SSD) distance measure [28] is given by

$$\mathcal{D}_{\mathrm{SSD}}(\mathcal{R}, \mathcal{T}; y) := \frac{1}{2} \int_{\Omega_{\mathcal{R}}} \left(\mathcal{T}(y(\mathbf{x})) - \mathcal{R}(\mathbf{x})\right)^2 \mathrm{d}\mathbf{x}.$$

Let $y_w : \mathbb{R}^3 \to \mathbb{R}^3,\ x \mapsto Ax + b$ denote a three-dimensional affine-linear transformation with $w = (w_1, \ldots, w_{12})$ and

$$A := \begin{pmatrix} w_1 & w_2 & w_3 \\ w_5 & w_6 & w_7 \\ w_9 & w_{10} & w_{11} \end{pmatrix},\ b := \begin{pmatrix} w_4 \\ w_8 \\ w_{12} \end{pmatrix}.$$

Setting $\mathcal{D}_{\mathrm{SSD}}(w) := \mathcal{D}_{\mathrm{SSD}}(\mathcal{R}, \mathcal{T}; y_w)$ yields the formulation of affine-linear image registration with SSD as minimization problem

$$\min_w\ \mathcal{D}_{\mathrm{SSD}}(w) \tag{25}$$

with $\mathcal{D}_{\mathrm{SSD}} : \mathbb{R}^{12} \to \mathbb{R}$. For discretization, the domain $\Omega_{\mathcal{R}}$ is assumed to be cuboid and decomposed into n cells of equal size with center points $\mathbf{x}_i,\ i = 1, \ldots, n$, arranged in lexicographical ordering. Using the midpoint quadrature rule for numerical integration, a discretized version of (25) reads

$$\min_w\ D_{\mathrm{SSD}}(w) := \frac{\bar{h}}{2} \sum_{i=1}^{n} \left(\mathcal{T}(y_w(\mathbf{x}_i)) - \mathcal{R}(\mathbf{x}_i)\right)^2,$$

where $\bar{h}$ denotes the volume of each cell. Multilinear interpolation with Dirichlet zero boundary conditions is used to evaluate the discrete template image at arbitrary coordinates.

Let $(\mathbf{x}_i)_j$ denote the j-th component of $\mathbf{x}_i \in \mathbb{R}^3$. For transformation parameters $w \in \mathbb{R}^{12}$, we define the vector

$$v_i := \begin{pmatrix} (A\mathbf{x}_1 + b)_i \\ (A\mathbf{x}_2 + b)_i \\ \vdots \\ (A\mathbf{x}_n + b)_i \end{pmatrix} \in \mathbb{R}^n,\ i = 1, 2, 3,$$

to construct the function

$$y : \mathbb{R}^{12} \to \mathbb{R}^{3n},\ w \mapsto \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}. \tag{26}$$

Using $\mathbf{y}_i = (y_i, y_{i+n}, y_{i+2n})^\top$, we define

$$T : \mathbb{R}^{3n} \to \mathbb{R}^n,\ \begin{pmatrix} y_1 \\ \vdots \\ y_{3n} \end{pmatrix} \mapsto \begin{pmatrix} \mathcal{T}(\mathbf{y}_1) \\ \vdots \\ \mathcal{T}(\mathbf{y}_n) \end{pmatrix}. \tag{27}$$

With $R_i := \mathcal{R}(\mathbf{x}_i)$, we set

$$r : \mathbb{R}^n \to \mathbb{R}^n,\ \begin{pmatrix} T_1 \\ \vdots \\ T_n \end{pmatrix} \mapsto \begin{pmatrix} T_1 - R_1 \\ \vdots \\ T_n - R_n \end{pmatrix}$$

as residual function and finally

$$\psi : \mathbb{R}^n \to \mathbb{R}, \ \begin{pmatrix} r_1 \\ \vdots \\ r_n \end{pmatrix} \mapsto \frac{\bar{h}}{2} \sum_{i=1}^{n} r_i^2$$

as the sum of all squared residual elements. Now, D_{SSD} can be written as a concatenation of four functions:

$$D_{\text{SSD}} : \mathbb{R}^{12} \xrightarrow{y} \mathbb{R}^{3n} \xrightarrow{T} \mathbb{R}^n \xrightarrow{r} \mathbb{R}^n \xrightarrow{\psi} \mathbb{R}. \tag{28}$$

A.1.1 Matrix-Based Differentiation

The differentiation of (28) is performed with the chain rule as

$$\nabla D_{\text{SSD}}(w) = \frac{\partial \psi}{\partial r} \frac{\partial r}{\partial T} \frac{\partial T}{\partial y} \frac{\partial y}{\partial w} \tag{29}$$

just as in the two-dimensional case. Again, we define the gradient as a row vector. The first two individual derivatives in (29) are given by

$$\begin{aligned} &\frac{\partial \psi}{\partial r}[r] = \bar{h}(r_1, \dots, r_n) \text{ and} \\ &\frac{\partial r}{\partial T}[T] = I_n, \end{aligned} \tag{30}$$

with $I_n \in \mathbb{R}^{n \times n}$ as the identity matrix. Denoting the partial derivative with respect to the i-th component by ∂_i and defining $\partial_i \mathcal{T}[y]$ as

$$\partial_i \mathcal{T}[y] := \begin{pmatrix} \partial_i \mathcal{T}(\mathbf{y}_1) & & \\ & \ddots & \\ & & \partial_i \mathcal{T}(\mathbf{y}_n) \end{pmatrix}, i = 1, 2, 3,$$

it holds that

$$\frac{\partial T}{\partial y}[y] = (\partial_1 \mathcal{T} \ \partial_2 \mathcal{T} \ \partial_3 \mathcal{T}) \in \mathbb{R}^{n \times 3n}. \tag{31}$$

Finally, the derivative of the function y is given by

$$\frac{\partial y}{\partial w}[w] = I_3 \otimes \mathbf{X} \in \mathbb{R}^{3n \times 12} \tag{32}$$

with the Kronecker product $\otimes$ and the grid matrix $\mathbf{X}$ as

$$\mathbf{X} := \begin{pmatrix} (\mathbf{x}_1)_1 & (\mathbf{x}_1)_2 & (\mathbf{x}_1)_3 & 1 \\ (\mathbf{x}_2)_1 & (\mathbf{x}_2)_2 & (\mathbf{x}_2)_3 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ (\mathbf{x}_n)_1 & (\mathbf{x}_n)_2 & (\mathbf{x}_n)_3 & 1 \end{pmatrix} \in \mathbb{R}^{n \times 4},$$

thus completing the analysis of the gradient components from (29). With

$$\mathrm{d}r := \frac{\partial r}{\partial T} \frac{\partial T}{\partial y} \frac{\partial y}{\partial w} \in \mathbb{R}^{n \times 12}, \tag{33}$$

the Gauss-Newton approximation H_{SSD} of the Hessian matrix is given by

$$H_{\text{SSD}}(w) := \mathrm{d}r^\top \mathrm{d}_2 \psi \mathrm{d}r$$

with $\mathrm{d}_2 \psi = \bar{h}$. Again, note $\frac{\partial r}{\partial T} = I_n$.

A.1.2 Matrix-Free Derivative Calculation

With (31) and (32), it follows that

$$\left(\frac{\partial T}{\partial y} \frac{\partial y}{\partial w} \right)_{i,j} = \begin{cases} \partial_1 \mathcal{T}(\mathbf{y}_i) \mathbf{X}_{i,j} & 1 \le j \le 4 \\ \partial_2 \mathcal{T}(\mathbf{y}_i) \mathbf{X}_{i,j-4} & 5 \le j \le 8 \\ \partial_3 \mathcal{T}(\mathbf{y}_i) \mathbf{X}_{i,j-8} & 9 \le j \le 12 \end{cases}. \tag{34}$$

Using (30), it holds that

$$\left(\frac{\partial \psi}{\partial r} \right)_i = \mathcal{T}_w(\mathbf{x}_i) - \mathcal{R}(\mathbf{x}_i)$$

with $\mathcal{T}_w(\mathbf{x}_i) := \mathcal{T}(\varphi_w(\mathbf{x}_i))$. The explicit calculation rule for the objective function gradient in the three-dimensional case is therefore given by

$$\nabla D_{\text{SSD}}(w) = \bar{h} \sum_{i=1}^{n} (\mathcal{T}_w(\mathbf{x}_i) - \mathcal{R}(\mathbf{x}_i)) \begin{pmatrix} \partial_1 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_1 \\ \partial_1 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_2 \\ \partial_1 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_3 \\ \partial_1 \mathcal{T}_w(\mathbf{x}_i) \\ \partial_2 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_1 \\ \partial_2 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_2 \\ \partial_2 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_3 \\ \partial_2 \mathcal{T}_w(\mathbf{x}_i) \\ \partial_3 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_1 \\ \partial_3 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_2 \\ \partial_3 \mathcal{T}_w(\mathbf{x}_i)(\mathbf{x}_i)_3 \\ \partial_3 \mathcal{T}_w(\mathbf{x}_i) \end{pmatrix}^\top. \tag{35}$$

The Gauss-Newton approximation to the Hessian for the SSD distance measure is defined as

$$\begin{aligned} H_{\text{SSD}} &= \mathrm{d}r^\top \mathrm{d}_2 \psi \mathrm{d}r \\ &= \bar{h} \left(\frac{\partial T}{\partial y} \frac{\partial y}{\partial w} \right)^\top \left(\frac{\partial T}{\partial y} \frac{\partial y}{\partial w} \right) \in \mathbb{R}^{12 \times 12}. \end{aligned}$$

By utilizing (34) and setting

$$l_k := \left(\left(\tfrac{\partial T}{\partial y} \tfrac{\partial y}{\partial w} \right)_{k,i} \cdot \left(\tfrac{\partial T}{\partial y} \tfrac{\partial y}{\partial w} \right)_{k,j} \right)_{1 \le i,j \le 12}, \tag{36}$$

it directly follows that

$$H_{\text{SSD}}(w) = \bar{h} \sum_{k=1}^{n} l_k.$$

A.2 Normalized Gradient Fields (NGF)

We consider the NGF distance measure [16]

$$\mathcal{D}_{\text{NGF}} := \frac{1}{2} \int_{\Omega_{\mathcal{R}}} 1 - \left(\frac{\langle \nabla \mathcal{R}(\mathbf{x}), \nabla \mathcal{T}(y(\mathbf{x})) \rangle_{\varrho,\tau}}{\|\nabla \mathcal{R}(\mathbf{x})\|_\varrho \, \|\nabla \mathcal{T}(y(\mathbf{x}))\|_\tau} \right)^2 \mathrm{d}\mathbf{x},$$

$\langle a, b \rangle_{\alpha,\beta} := \sum_{i=1}^{3} a_i b_i + \alpha\beta, \ a, b \in \mathbb{R}^3, \|a\|_\varepsilon := \sqrt{\sum_{i=1}^{3} a_i^2 + \varepsilon^2}$, with separate edge parameters for reference and template image, cf. [35]. Setting $\mathcal{D}_{\text{NGF}}(w) := \mathcal{D}_{\text{NGF}}(\mathcal{R}, \mathcal{T}; y_w)$, affine-linear image registration with NGF translates to

$$\min_w \ \mathcal{D}_{\text{NGF}}(w). \tag{37}$$

For numerical optimization, the continuous formulation in (37) is discretized. For a reference image of size $n_1 \times n_2 \times n_3$ and an index $i,\ i = 1, \dots, n$, let $i', j', k' \in \mathbb{N}, 1 \le i' \le n_1,\ 1 \le j' \le n_2,\ 1 \le k' \le n_3$ such that $i = i' + j'n_1 + k'n_1n_2$. The indices of neighboring points with Neumann zero boundary conditions are given by

$$\begin{aligned} i_{-x} &= \max(i'-1, 1) + j'n_1 + k'n_1n_2, \\ i_{+x} &= \min(i'+1, n_1) + j'n_1 + k'n_1n_2, \\ i_{-y} &= i' + \max(j'-1, 1)n_1 + k'n_1n_2, \\ i_{+y} &= i' + \min(j'+1, n_2)n_1 + k'n_1n_2, \\ i_{-z} &= i' + j'n_1 + \max(k'-1, 1)n_1n_2, \\ i_{+z} &= i' + j'n_1 + \min(k'+1, n_3)n_1n_2. \end{aligned} \tag{38}$$

We define functions

$$g_i : \mathbb{R}^n \to \mathbb{R}^3, \qquad T \mapsto \begin{pmatrix} \frac{1}{2h_1}(-T_{i_{-x}} + T_{i_{+x}}) \\ \frac{1}{2h_2}(-T_{i_{-y}} + T_{i_{+y}}) \\ \frac{1}{2h_3}(-T_{i_{-z}} + T_{i_{+z}}) \end{pmatrix}$$

and

$$s_i : \mathbb{R}^3 \to \mathbb{R}, \qquad a \mapsto \langle g_i(R), a \rangle + \varrho\tau$$

for gradient and scalar product type operations at the position i, respectively. Further setting

$$n_\varepsilon : \mathbb{R}^3 \to \mathbb{R}, \qquad a \mapsto \sqrt{a_1^2 + a_2^2 + a_3^2 + \varepsilon^2},$$

the discretized version of (37) is given by

$$\min_w \ D_{\mathrm{NGF}}(w) := \frac{\bar{h}}{2} \sum_{i=1}^{n} 1 - \left(\frac{s_i(g_i(T_w))}{n_\varrho(g_i(R))\ n_\tau(g_i(T_w))} \right)^2$$

with $(T_w)_i = \mathcal{T}(y_w(\mathbf{x}_i))$.

A.2.1 Matrix-Based Differentiation

Let y and T as in (26) and (27). We define the residual function $r : \mathbb{R}^n \to \mathbb{R}^n$ by setting the i-th component function $r_i : \mathbb{R}^n \to \mathbb{R}$ to

$$r_i : T \mapsto \frac{s_i(g_i(T))}{n_\varrho(g_i(R))\ n_\tau(g_i(T))}.$$

The reduction function $\psi : \mathbb{R}^n \to \mathbb{R}$ is given by

$$\psi(r) = \frac{\bar{h}}{2} \sum_{i=1}^{n} 1 - r_i^2,$$

yielding the function chain

$$D_{\mathrm{NGF}} : \mathbb{R}^{12} \xrightarrow{y} \mathbb{R}^{3n} \xrightarrow{T} \mathbb{R}^n \xrightarrow{r} \mathbb{R}^n \xrightarrow{\psi} \mathbb{R}.$$

The derivatives of T and y have already been computed in (31) and (32). For the reduction function ψ, it holds that

$$\frac{\partial \psi}{\partial r} = -\bar{h} r^\top \in \mathbb{R}^{1 \times n}. \tag{39}$$

The calculation of $\frac{\partial r}{\partial T}$ is performed by differentiating the component functions $r_i,\ i = 1, \dots, n$. The functions r_i are composed of s_i, g_i and n_ε whose derivatives are given by

$$\frac{\partial s_i}{\partial a} = g_i(R)^\top \in \mathbb{R}^{1 \times 3},$$

$$\frac{\partial g_i}{\partial T} = \begin{pmatrix} & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ 0 & \cdots & 0 & \cdots & -\frac{1}{2h_1} & 0 & \frac{1}{2h_1} & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & -\frac{1}{2h_2} & \cdots & 0 & 0 & 0 & \cdots & \frac{1}{2h_2} & \cdots & 0 \\ -\frac{1}{2h_3} & \cdots & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & \frac{1}{2h_3} \end{pmatrix} \begin{matrix} \\ \\ \\ i_{-z} \\ i_{-y} \\ i_{-x} \\ i \\ i_{+x} \\ i_{+y} \\ i_{+z} \end{matrix}$$

and

$$\frac{\partial n_\varepsilon}{\partial a} = \frac{1}{n_\varepsilon(a)} a^\top \in \mathbb{R}^{1 \times 3}$$

with $\frac{\partial g_i}{\partial T} \in \mathbb{R}^{3 \times n}$. Applying the chain rule in both numerator and denominator of r_i yields

$$\frac{\partial r_i}{\partial T} = \begin{pmatrix} \vdots \\ \frac{1}{2h_3}\left[\frac{-g_i(R)_3}{n_\varrho(g_i(R))n_\tau(g_i(T))} + \frac{s_i(g_i(T))g_i(T)_3}{n_\varrho(g_i(R))n_\tau(g_i(T))^3}\right] \\ \vdots \\ \frac{1}{2h_2}\left[\frac{-g_i(R)_2}{n_\varrho(g_i(R))n_\tau(g_i(T))} + \frac{s_i(g_i(T))g_i(T)_2}{n_\varrho(g_i(R))n_\tau(g_i(T))^3}\right] \\ \vdots \\ \frac{1}{2h_1}\left[\frac{-g_i(R)_1}{n_\varrho(g_i(R))n_\tau(g_i(T))} + \frac{s_i(g_i(T))g_i(T)_1}{n_\varrho(g_i(R))n_\tau(g_i(T))^3}\right] \\ 0 \\ \frac{1}{2h_1}\left[\frac{g_i(R)_1}{n_\varrho(g_i(R))n_\tau(g_i(T))} - \frac{s_i(g_i(T))g_i(T)_1}{n_\varrho(g_i(R))n_\tau(g_i(T))^3}\right] \\ \vdots \\ \frac{1}{2h_2}\left[\frac{g_i(R)_2}{n_\varrho(g_i(R))n_\tau(g_i(T))} - \frac{s_i(g_i(T))g_i(T)_2}{n_\varrho(g_i(R))n_\tau(g_i(T))^3}\right] \\ \vdots \\ \frac{1}{2h_3}\left[\frac{g_i(R)_3}{n_\varrho(g_i(R))n_\tau(g_i(T))} - \frac{s_i(g_i(T))g_i(T)_3}{n_\varrho(g_i(R))n_\tau(g_i(T))^3}\right] \\ \vdots \end{pmatrix}^\top$$

with the vector entries at positions $i_{-z}, i_{-y}, i_{-x}, i_{+x}, i_{+y}$, and i_{+z} (in that order) as defined in (38). Note that these positions may coincide, in which case the values are added.

The Gauss-Newton approximation H_{NGF} to the Hessian is given by

$$H_{\mathrm{NGF}}(w) := \mathrm{d}r^\top \mathrm{d}_2\psi \mathrm{d}r \approx \nabla^2 D_{\mathrm{NGF}}(w)$$

with $\mathrm{d}r$ defined as in (33) and $\mathrm{d}_2\psi = -\bar{h}$.

A.2.2 Matrix-Free Derivative Calculation

Setting $r_i := \frac{s_i(g_i(T))}{n_\varrho(g_i(R))\ n_\tau(g_i(T))}$ and $\mathrm{d}r_i := \frac{\partial r_i}{\partial T}\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}$, it holds with (39) that

$$\nabla D_{\mathrm{NGF}}(w) = -\bar{h} \sum_{i=1}^{n} r_i \mathrm{d}r_i. \tag{40}$$

As $r_i \in \mathbb{R}$ are scalars, it suffices to derive a matrix-free description of the vectors $\mathrm{d}r_i \in \mathbb{R}^{12}$ to achieve a fully matrix-free formulation of the objective function gradient. Let $1 \le i \le n$ and define indices $i_{-z}, i_{-y}, i_{-x}, i_{+x}, i_{+y}, i_{+z}$ as in (38). With the definition

$$\begin{aligned} d_i^{j,k} :&= \partial r_i[i_{-z}]\partial_j \mathcal{T}(\mathbf{y}_{i_{-z}})\mathbf{X}_{i_{-z},k} \\ &+ \partial r_i[i_{-y}]\partial_j \mathcal{T}(\mathbf{y}_{i_{-y}})\mathbf{X}_{i_{-y},k} \\ &+ \partial r_i[i_{-x}]\partial_j \mathcal{T}(\mathbf{y}_{i_{-x}})\mathbf{X}_{i_{-x},k} \\ &+ \partial r_i[i_{+x}]\partial_j \mathcal{T}(\mathbf{y}_{i_{+x}})\mathbf{X}_{i_{+x},k} \\ &+ \partial r_i[i_{+y}]\partial_j \mathcal{T}(\mathbf{y}_{i_{+y}})\mathbf{X}_{i_{+y},k} \\ &+ \partial r_i[i_{+z}]\partial_j \mathcal{T}(\mathbf{y}_{i_{+z}})\mathbf{X}_{i_{+z},k} \end{aligned}$$

for $i = 1, \dots, n$, $j = 1, 2, 3$, $k = 1, \dots, , 4$ and

$d_i^j := \left(d_i^{j,1}, d_i^{j,2}, d_i^{j,3}, d_i^{j,4}\right)$,

it follows that

$$\mathrm{d}r_i = \left(d_i^1 \; d_i^2 \; d_i^3\right)^\top \in \mathbb{R}^{12}, \tag{41}$$

which according to (40) yields

$$\nabla D_{\mathrm{NGF}}(w) = -\bar{h} \sum_{i=1}^{n} \frac{s_i(g_i(T))}{n_\varrho(g_i(R)) \; n_\tau(g_i(T))} \begin{pmatrix} \mathrm{d}r_i[1] \\ \mathrm{d}r_i[2] \\ \vdots \\ \mathrm{d}r_i[12] \end{pmatrix}^\top, \tag{42}$$

completing the gradient calculation for the three-dimensional case. Since

$$\begin{aligned} H_{\mathrm{NGF}}(w) &= \left(\frac{\partial r}{\partial T}\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}\right)^\top \mathrm{d}_2\psi \left(\frac{\partial r}{\partial T}\frac{\partial T}{\partial y}\frac{\partial y}{\partial w}\right) \\ &= \left(\mathrm{d}r_1^\top \; \dots \; \mathrm{d}r_n^\top\right) \mathrm{d}_2\psi \begin{pmatrix} \mathrm{d}r_1 \\ \vdots \\ \mathrm{d}r_n \end{pmatrix}, \end{aligned}$$

the calculation of the Hessian approximation can directly be performed with the help of the matrix-free formulation of $\mathrm{d}r_i$ from (41). By defining the matrices $l_k \in \mathbb{R}^{12\times 12}$ as

$$l_k := \left(\mathrm{d}r_k[i] \cdot \mathrm{d}r_k[j]\right)_{1\le i,j\le 12} \tag{43}$$

analog to the case of SSD, the matrix-free formulation for the Gauss-Newton approximation to the Hessian is given by

$$H_{\mathrm{NGF}}(w) = \bar{h} \sum_{k=1}^{n} l_k.$$

This finalizes the derivation of matrix-free calculation rules for objective function gradient and Gauss-Newton approximation to the Hessian also for the Normalized Gradient Fields distance measure with three-dimensional images.