A FAST FULL MULTIGRID SOLVER FOR APPLICATIONS IN IMAGE PROCESSING

M. STÜRMER[1], H. KÖSTLER[1], V. DAUM[2] AND U. RÜDE[1]
LEHRSTUHL FÜR SYSTEMSIMULATION[1], MUSTERERKENNUNG[2]
FRIEDRICH-ALEXANDER UNIVERSITY OF ERLANGEN-NUREMBERG
91058 ERLANGEN, GERMANY
{KOESTLER, STUERMER, VOLKER.DAUM, RUEDE}@INFORMATIK.UNI-ERLANGEN.DE

Abstract. We present a fast, cell-centered multigrid solver and apply it to image denoising and non-rigid diffusion based image registration. In both applications real time performance is required in 3D and the multigrid method has to be compared to solvers based on Fast Fourier Transform. The optimization of the underlying variational approach results for image denoising directly in one time step of a parabolic linear heat equation, for image registration a non-linear 2nd order system of partial differential equations is obtained. This system is solved by a fixpoint iteration using a semi-implicit time discretization, where each time step again results in an elliptic linear heat equation. The multigrid implementation comes close to real time performance for medium size medical images in 3D for both applications and is compared to a solver based on Fast Fourier Transform using available libraries.

1. Introduction. In recent years the data sizes in image processing applications have drastically increased due to the improved image acquisition systems. Modern computer tomography (CT) scanners can create volume data sets of 512^3 voxels or more [20, 29]. However, users expect real time image manipulation and analysis. Thus fast algorithms and implementations are needed to fulfill these tasks.

Many image processing problems can be formulated in a variational framework and require the solution of a large, sparse linear system arising from the discretization of partial differential equations (PDEs). Often these PDEs are inherently based on some kind of diffusion process. In simple cases, it is possible to use Fast Fourier Transform (FFT) based techniques to solve these PDEs that are of complexity $\mathcal{O}(n \log n)$. The FFT algorithm was introduced in 1965 by Cooley and Tukey [7], for an overview of Fourier Transform methods we refer e.g. to [8, 31, 30]. As an alternative, multigrid methods are more general and can reach an asymptotically optimal complexity of $\mathcal{O}(n)$.

For discrete Fourier transforms flexible and highly efficient libraries optimized for special CPU architectures such as the FFTW library [11] or the Intel Math Kernel Library (MKL) [1] are available. However, we are currently not aware of similarly tuned multigrid libraries in 3D and only of DiMEPACK [24] for 2D problems. The purpose of this paper is to close this gap and to implement a multigrid solver optimized especially for the Intel x86 architecture that is competitive to highly optimized FFT libraries and apply it to typical applications in the area of image processing.

The outline of our paper is as follows: We describe the multigrid scheme including some results on its convergence and discuss some implementation and optimization issues in Section 2. Then the variational approaches used for image denoising and non-rigid diffusion registration are introduced in Section 3. Finally, we compare computational times of our multigrid solver and the FFTW package as obtained for image denoising and non-rigid registration of medical CT images.

2. Multigrid. For a comprehensive overview on multigrid methods and further references, we e.g. refer to [4, 13, 5, 34, 40, 39]. In this paper we implement a multigrid solver for the linear heat equation

$$\frac{\partial u}{\partial t}(\boldsymbol{x}, t) - \Delta u(\boldsymbol{x}, t) = f(\boldsymbol{x}), \quad u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}) \tag{2.1}$$

with time $t \in \mathbb{R}^+$, $u, f : \Omega \subset \mathbb{R}^3 \to \mathbb{R}, \boldsymbol{x} \in \Omega$, initial solution $u_0 : \Omega \subset \mathbb{R}^3 \to \mathbb{R}$ and homogeneous Neumann boundary conditions. Note that in practice, $u(\boldsymbol{x}, t)$ is often computed for a finite t, only, and in the limit for $t \to \infty$, the solution tends to the well-known Poisson equation. We discretize (2.1) with finite differences

$$\frac{u_h(\boldsymbol{x}, \tau) - u_0(\boldsymbol{x})}{\tau} - \Delta_h u_h(\boldsymbol{x}, \tau) = f_h(\boldsymbol{x}), \tag{2.2}$$

on a regular grid Ω_h with mesh size h and time step τ. Δ_h denotes the well-known 7-point stencil for the Laplacian. We consider in the following only a single time step, where we have to solve the elliptic equation

$$(I - \tau\Delta_h)u_h(\boldsymbol{x}, \tau) = \tau f_h(\boldsymbol{x}) + u_0(\boldsymbol{x}). \tag{2.3}$$

In this paper, we are dealing with image processing problems, where we can think of the discrete voxels located in the cell centers. Therefore, we have chosen to use a cell-centered multigrid scheme with constant interpolation and 8-point restriction. Note that this combination of intergrid transfer operators will lead to multigrid convergence rates significantly worse than what could be ideally obtained ([39, 27]). This we will show by local Fourier analysis. However, this leads to a relatively simple algorithm that satisfies our numerical requirements and is quite suitable for a careful machine specific performance optimization. For relaxation we choose a ω-Red-Black Gauss-Seidel smoother (ωRBGS) using $\omega = 1.15$, that is known to be a better choice in 3D for the given problem than simple Gauss-Seidel relaxation ([34, 41]).

2.1. Efficient Multigrid implementation. This section describes our multigrid implementation. All floating point calculations are done with single precision (four bytes per value), as this accuracy is already far beyond that of the source image data. Performance of multigrid implementations can be improved significantly when code optimization techniques are used as shown in [3, 23, 22]. In this paper we will focus on the x86 processor architecture, since it is currently the most common desktop PC platform.

2.1.1. Memory Layout. Best performance on current x86 processors can be achieved when the SIMD (single instruction multiple data) unit is used, which was introduced to the architecture in 1999 with the Pentium III as Streaming SIMD Extension (SSE). These instructions perform vector-like operations on units of 16 Bytes, which can be seen as a SIMD vector data type containing four single precision floating point numbers in our case. Operating on naturally aligned (i.e. at addresses multiples of their size) SIMD vectors, the SSE unit provides high bandwidth especially to the caches. Consequently the memory layout must support aligned data accesses in all multigrid components as much as possible. To enable efficient handling of the boundary conditions, we chose to explicitly store boundary points around our grid; by copying the outer unknowns before smoothing or calculating the point wise residuals we need no special handling of the homogeneous Neumann boundary conditions. The first unknown of every line is further aligned to a multiple of 16 Bytes by padding, i.e. filling up the line with unused values up to a length a multiple of four. This enables SIMD processing for any line length, as boundary values which are generated just-in-time and the padding area can be overwritten with fake results.

2.1.2. SIMD-aware implementation. Unfortunately current compilers fail to generate SIMD instruction code from a scalar description of most real world programs.

The SIMD unit can be programmed in assembly language, but as it makes the code more portable and maintainable, our C++ implementation uses compiler intrinsics, which extend the programming language with assembly-like instructions for SIMD vector data types.

Implementing the ωRBGS relaxation in SIMD is not straightforward, as only red or black points must be updated, while every SIMD vector contains two values of each color. The idea of the SIMD-aware ωRBGS is to first calculate a SIMD vector of relaxed values, like for a Jacobi method. Subsequently a SIMD multiplication with appropriately initialized SIMD registers is performed such that either values are preserved and the others are relaxed, which can be illustrated as

$$\begin{bmatrix} u_{\text{new}}(x,y) \\ u_{\text{new}}(x+1,y) \\ u_{\text{new}}(x+2,y) \\ u_{\text{new}}(x+3,y) \end{bmatrix} = \begin{bmatrix} 1-\omega \\ 1 \\ 1-\omega \\ 1 \end{bmatrix} * \begin{bmatrix} u_{\text{old}}(x,y) \\ u_{\text{old}}(x+1,y) \\ u_{\text{old}}(x+2,y) \\ u_{\text{old}}(x+3,y) \end{bmatrix} + \begin{bmatrix} \omega \\ 0 \\ \omega \\ 0 \end{bmatrix} * \begin{bmatrix} u_{\text{relax}}(x,y) \\ u_{\text{relax}}(x+1,y) \\ u_{\text{relax}}(x+2,y) \\ u_{\text{relax}}(x+3,y) \end{bmatrix} .$$

The better internal and external bandwidth of SIMD over the scalar floating point unit leads to a real performance gain, even if we actually double the number of floating point operations.

The cell-centered approach is advantageous especially for restriction and interpolation. Coarsening is done by averaging eight neighboring fine grid residuals where every fine grid residual contributes only to a single coarse grid point. Hence, calculation of the residual and its restriction can be done in SIMD and without storing residuals to memory. The idea is to compute four SIMD registers containing residuals from four neighboring lines and averaging them into a single SIMD vectors first. Its values are reordered by special shuffle instructions, so that two coarse grid right hand side values can be generated by averaging its first and second, and its third and fourth value. By reusing some common expressions this can be further simplified. The constant interpolation can also be executed very efficiently in the SIMD unit with shuffle operations.

Additionally, the loops are unrolled and the instructions scheduled carefully by hand to support the compiler in producing fast code.

2.1.3. Blocking and Fusion of Components. SIMD optimization is most useful when combined with techniques to enhance spatial and temporal data locality developed in [38, 22, 32] to exploit the higher bandwith of the caches. For smaller grids the post-smoother uses a simple blocking method as illustrated in Fig. 2.1.3 (I). After preparing the first boundary (I a), it continues after the red update in line y, z immediately with the black update in line $y-1, z$ (I b) through the whole grid (I c) and finishes the sweep with a black update in the last plane (I d). As long as data from the last block can be held in the cache hierarchy, the solution and right-hand-side grid must only be transferred from and to memory once. For larger grids, this is not possible anymore and another blocking level must be introduced as illustrated in Fig. 2.1.3 (II). The grid is divided in x-z-direction then, and every resulting super-block is processed in a similar manner as in the simple case, but the red update in line y, z is followed by the black update in line $y-z, z-1$ to respect data dependencies between two super-blocks. Therefore the first and last super-block need a special boundary handling (II a, b and d). This two-fold blocking method is slightly less effective, since the super-blocks overlap and some values are read from main memory twice. The optimal super-block height depends on the cache size and the line length.

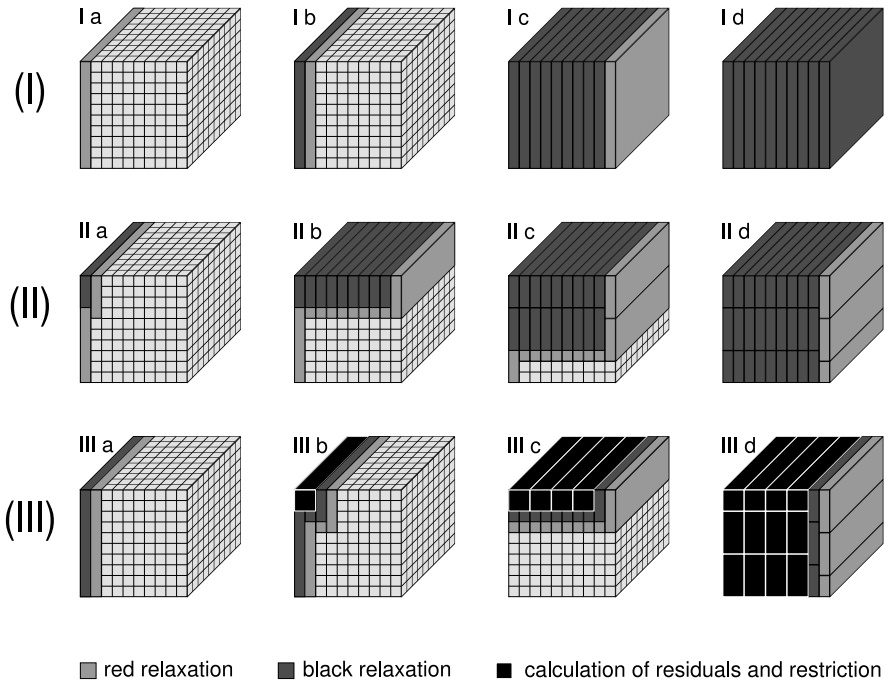

Figure 2.1: Illustration of the different blocking methods on a $10 \times 10 \times 10$ cube.

(I) simple plane blocking of one ωRBGS update a. initial boundary handling b. first block c. blocking complete d. final boundary handling
(II) super-blocking of one ωRBGS update a. first sub-block of first super-block b. first super-block complete c. middle super-block complete d. last super-block complete
(III) super-blocking of one ωRBGS update fused with calculation of residual and restriction a. initial boundary handling b. first sub-block of first super-block c. first super-block complete d. only final boundary handling missing

The pre-smoother extends these blocking methods further by fusing the smoothing step with calculation and restriction of the residuals. For smaller grids the simpler blocking method working on whole planes (I) is extended: right hand side values of the coarser grid plane z are computed immediately after smoothing in the planes $2z$ and $2z + 1$ is done. This leads to a slightly more complex handling at the first and last planes. For larger planes, however, super-blocks must be used again as depicted in Fig. 2.1.3 (III).

2.2. Convergence rates. The asymptotic convergence rates of our algorithm are evaluated in a vector iteration for equation (2.3), i. e. setting the right hand side f_h and u_0 to zero and scaling the discrete L_2-norm of the solution u_h to 1 after each multigrid V-cycle iteration step. In Table 2.1 asymptotic convergence rates (after 100 iterations) for different settings are shown. We compare these results to local Fourier analysis (LFA) predictions computed by the *lfa package* (cf. [40]) in Table 2.2. This confirms our observations that due to the constant interpolation the asymptotic convergence rates get worse for smaller mesh sizes. Note that setting

size	τ	V(1,1)	V(2,2)
64^3	10^4	0.26	0.07
128^3	10^4	0.28	0.07
256^3	10^4	0.29	0.07
512^3	10^4	0.32	0.07
64^3	10^{30}	0.27	0.07
128^3	10^{30}	0.29	0.07
256^3	10^{30}	0.32	0.07
512^3	10^{30}	0.34	0.07

Table 2.1: Asymptotic convergence rates for different time steps measured experimentally with mesh size $h = 1.0$ on the finest grid and 1 grid point on the coarsest level. For $\tau \to \infty$ this results effectively in the Poisson equation.

size	τ	interpolation	ω	V(1,1)		V(2,2)	
				ρ	$\rho(M_{3L})$	ρ	$\rho(M_{3L})$
64^3	10^{30}	constant	1.0	0.20	0.47	0.04	0.07
64^3	10^{30}	constant	1.15	0.08	0.20	0.04	0.06
64^3	10^{30}	trilinear	1.15	0.08	0.10	0.04	0.06

Table 2.2: Smoothing factor (ρ) and three-grid asymptotic convergence factor ($\rho(M_{3L})$) for different sizes and parameters obtained by LFA. Settings are equivalent to Table 2.2.

$\omega = 1$ the asymptotic convergence factor breaks down.

2.3. Performance results. Next we discuss performance results measured on two different test platforms. As reference we present run time for a forward and backward Fast Fourier Transform (FFT) used for periodic boundary conditions and Discrete Cosine Transform (DCT) used for Neumann boundary conditions, respectively. This does not contain the time necessary for actually solving the problem in Fourier space as described in subsection 3.5, which is highly dependent on the code quality. For our applications, the accuracy of a simple FMG-V(1,1) or even a simple V(1,1)-cycle is often sufficient, as will be explained in Section 3. On both platforms we compare performance of our code-optimized multigrid implementation with the performance of the well-known FFTW package [2] (version 3.1.2).

The first test platform is an AMD Opteron 248 cluster node. The CPUs run at 2.2 GHz and provide an 1 MB unified L2 and 64 KB L1 data cache and are connected to DDR-333 memory. For this platform the GNU C and C++ compiler (version 4.1.0 for 64 bit environment) was used. Measurements (see Table 2.3) show that a Full Multigrid with V(1,1)-cycles can outperform the FFTW's FFTs and is much faster than its DCTs even with V(2,2)-cycles.

The second test platform is an Intel Core2 Duo (Conroe) workstation. The CPU runs at 2.4 GHz, both cores have a L1 data cache of 16 KB, share 4 MB of unified L2 cache and are connected to DDR2-667 memory. For this platform the Intel 64 compiler suite (version 9.1) was used. We also present results for a beta version of the Intel MKL [1] (version 9.0.06 beta for 64 bit environment), which provides an FFTW-compatible interface for FFTs through wrapper functions, but no DCT functions at all.

Although a slightly different instruction scheduling more suitable for that CPU type is used, all multigrid variants are slower at smaller problem sizes than the FFTs of FFTW and the MKL on this platform (see Table 2.3), the FMG with V(2,2)-cycles even at all problem sizes. Again, the DCTs take much more time than the code-optimized multigrid at all problem sizes tested.

size	V (1,1)	FMG V(1,1)	FMG V(2,2)	FFT (fftw)	DCT (fftw)
32	0.63	0.80	1.38	0.85	2.27
64	6.97	9.55	14.9	10.4	19.1
128	56.0	78.7	122	107	197
256	445	622	976	992	2024
512	3669	5175	7943	9274	67766

Table 2.3: Wallclock times in ms for FFT (real type, out of place, forward and backward) and the optimized multigrid on an AMD Opteron 248 2.2 GHz cluster node.

size	V (1,1)	FMG V(1,1)	FMG V(2,2)	FFT (fftw)	DCT (fftw)	FFT (mkl)
32	0.43	0.55	0.93	0.40	1.43	0.71
64	3.33	4.29	7.12	3.73	12.2	5.27
128	31.6	44.1	68.3	50.4	123	45.8
256	264	370	574	473	1246	401
512	2168	3026	4699	4174	11067	3510

Table 2.4: Wallclock times in ms for FFT (real type, out of place, forward and backward) and the optimized multigrid on an Intel Core2 Duo 2.4 GHz (Conroe) workstation.

3. Variational approaches in image processing. Variational approaches in image processing are often considered as too slow for real time applications, especially in 3D. Nevertheless, they are attractive due to their flexibility and the quality of the results, see e.g. [17, 25, 19, 20, 26, 28, 36]. In the following we introduce two very simple variational prototype problems. Most of the more complicated image processing tasks consist of extensions of these approaches that include e.g. introducing local anisotropy in the PDEs. The reason why we restrict ourselves to these simple approaches is that they can be solved by FFT based methods and by multigrid and they are therefore good benchmark problems to test the best possible speed of variational image processing methods.

3.1. Image Denoising. The task of image denoising is to remove the noise from a given d-dimensional image $u_0 : \Omega \subset \mathbb{R}^d \to \mathbb{R}$. One simple variational based on Tikhonov regularization [33] is to minimize the functional

$$E_1(u) = \int_\Omega |u_0 - u|^2 + \alpha |\nabla u|^2 d\boldsymbol{x} \tag{3.1}$$

with $\boldsymbol{x} \in \mathbb{R}^d$ and $\alpha \in \mathbb{R}^+$ over the image domain $\Omega \subset \mathbb{R}^d$. A necessary condition for a minimizer $u : \Omega \to \mathbb{R}$, the denoised image, is characterized by the Euler-Lagrange

equations

$$u - u_0 - \alpha \Delta u = 0 \tag{3.2}$$

with homogeneous Neumann boundary conditions. This is equivalent to (2.3) with $f_h = 0$ and $\tau = \alpha$. In an infinite domain an explicit solution is given by

$$u(\boldsymbol{x}, t) = \int_{\mathbb{R}^d} G_{\sqrt{2t}}(\boldsymbol{x} - \boldsymbol{y}) u_0(\boldsymbol{y}) d\boldsymbol{y} = (G_{\sqrt{2t}} * u_0)(\boldsymbol{x}) \ , \tag{3.3}$$

where the operator $*$ denotes the convolution of the grid function u_0 and the Gaussian kernel

$$G_\sigma(\boldsymbol{x}) = \frac{1}{2\pi\sigma^2} e^{-|\boldsymbol{x}|^2/(2\sigma^2)} \ , \tag{3.4}$$

with standard deviation $\sigma \in \mathbb{R}^+$. This is equivalent to applying a low-pass filter and can be transformed into Fourier space, where a convolution corresponds to a multiplication of the transformed signals. If we denote by $F[f]$ the Fourier transform of a signal $f : \mathbb{R}^d \to \mathbb{R}$ and use

$$F[G_\sigma](\boldsymbol{w}) = e^{-|\boldsymbol{x}|^2/(2/\sigma^2)}, \boldsymbol{w} \in \mathbb{R}^d$$

it follows that

$$F[G_\sigma * u_0](\boldsymbol{w}) = e^{-|\boldsymbol{x}|^2/(2/\sigma^2)} F[u_0](\boldsymbol{w}) \ . \tag{3.5}$$

Summarizing, we have three choices to compute the denoised image:

1. the convolution of the image with a discrete version of the Gaussian kernel (3.4),
2. using a FFT to solve (3.5), or
3. apply a multigrid method to (2.3).

In the first two methods, we extend the image symmetrically and use periodic boundary conditions, while we assume homogeneous Neumann boundary conditions for the third method. In most applications applying a filter mask to the image constructed from a discrete version of the Gaussian kernel (3.4) is an easy and efficient way to denoise the image. However, if large σ (and thus large t) is required, the filter masks become large and computationally inefficient. To show this we add Gaussian noise to a rendered 3D MRI image (size $256 \times 256 \times 160$) of a human head (see Fig. 3.1) and filter it using a mask of size $7 \times 7 \times 7$ with $\sigma = 1.21$. We do not decompose the mask as described in [19] to speed up the computation, but directly apply it to the image, what took about 30 sec. A mask size of $3 \times 3 \times 3$ reduces the time to 681 ms. These times and all others in this section are measured on the AMD Opteron platform described in the performance results section.

Then, we use our cell-based multigrid method to solve (2.3). Fig. 3.1 shows the resulting blurred volume. The filtering took 390 ms using one $V(1,1)$-cycle that is enough for small time steps. Large time steps would e.g. blur image edges too much. Using the FFTW-package the computational time was 1140 ms for the forward and backward transform and to apply (3.5). The multiplication with the exponential was not optimized and took about 50% of the time. Note that the Laplacian has very strong isotropic smoothing properties and does not preserve edges. Therefore in practise the model (3.1) is not used to restore deteriorated images, but to presmooth the image e.g. in order to ensure a robust estimation of the image gradient.

Next, we turn to another prototype problem in image processing, that involves also the solution of several problems of the type (2.3).

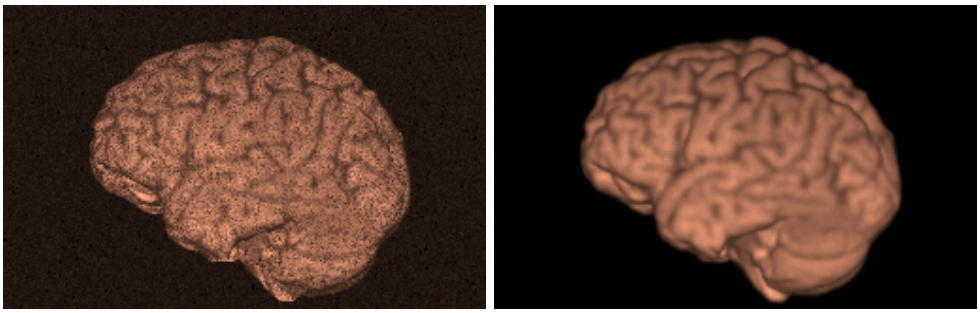

Figure 3.1: Rendered 3D MRI image with added Gaussian noise ($\sigma = 10$) added (left) and after denoising (right) using a V(1,1)-cycle of the cell-centered multigrid method.

3.2. Non-rigid image registration. The task of image registration is to align two or more images from the same or different modalities [16, 35]. We consider here only mono-modal registration. This requires finding a suitable spatial transformation such that a transformed image becomes similar to another one, see e.g. [26, 9, 12, 6, 10, 14]. This deformation is independent of the motion of the object, e.g. a rotation. For image registration two d-dimensional images are given by

$$T, R : \Omega \subset \mathbb{R}^d \to \mathbb{R}, \tag{3.6}$$

where T and R are template image and reference image, respectively, and Ω is the image domain. The task of non-rigid image registration is to find a transformation $\phi(x)$ such that the deformed image $T(\phi_u(x))$ can be matched to image $R(x)$. The transformation is defined as

$$\phi_{\boldsymbol{u}}(\cdot) : \mathbb{R}^d \to \mathbb{R}^d \ , \phi_{\boldsymbol{u}}(\boldsymbol{x}) := \boldsymbol{x} - \boldsymbol{u}(\boldsymbol{x}) \ , \boldsymbol{x} \subset \Omega \ ,$$

where the displacement $\boldsymbol{u}(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}^d, \boldsymbol{u} = (u_1, \ldots, u_d)$ is a d-dimensional vector field. Mathematically, we again use a variational approach to minimize the energy functional

$$E_2(u) = \int_\Omega (T(\boldsymbol{x} - \boldsymbol{u}(\boldsymbol{x})) - R(\boldsymbol{x}))^2 + \alpha \sum_{l=1}^{d} \|\nabla u_l\|^2 \, d\boldsymbol{x} \ . \tag{3.7}$$

that consists of two parts. The first term $(T(\boldsymbol{x}-\boldsymbol{u}(\boldsymbol{x}))-R(\boldsymbol{x}))^2$ is a distance measure that evaluates the similarity of the two images. Here we restrict ourselves to the sum of squared differences (SSD) as represented in the integral in (3.7). When discretized, this results in a point-wise "least squares" difference of gray values. The second term, the regularizer, controls the smoothness or regularity of the transformation. In the literature many different regularizers were discussed [26]. We restrict ourselves here to the so-called diffusion regularizer $\sum_{l=1}^{d} \|\nabla u_l\|^2$ [9]. By choosing different parameters $\alpha \in \mathbb{R}^+$ one can control the relative weight of the two terms in the functional [18, 21].

The optimization of the energy functional results in nonlinear Euler Lagrange equations

$$\nabla T(\boldsymbol{x} - \boldsymbol{u}(\boldsymbol{x})) \, (T(\boldsymbol{x} - \boldsymbol{u}(\boldsymbol{x})) - R(\boldsymbol{x})) + \alpha \Delta \boldsymbol{u} = 0 \tag{3.8}$$

Algorithm 1 Image registration scheme.

1: Set $\boldsymbol{u}^0$;$\boldsymbol{f}^0 = \nabla_h T(\boldsymbol{u}^k)\left(T(\boldsymbol{u}^k) - R\right)$;
2: **for** *timestep* $= 0$ to k **do**
3: Compute $\boldsymbol{f}^k = \nabla_h T(\boldsymbol{u}^k)\left(T(\boldsymbol{u}^k) - R\right)$;
4: Update $\tau := \epsilon_\tau \tau, \alpha := \epsilon_\alpha \alpha$ if necessary;
5: Compute $\boldsymbol{r}^k = \tau \boldsymbol{f}^k + \boldsymbol{u}^k$;
6: Solve $(I - \tau\alpha\Delta_h)\boldsymbol{u}^{k+1} = \boldsymbol{r}^k$;
7: **end for**

with homogeneous Neumann boundary conditions that can be discretized by finite differences on a regular grid Ω_h with mesh size h. To treat the nonlinearity often an artificial time is introduced

$$\partial_t \boldsymbol{u}(\boldsymbol{x}, t) - \alpha\Delta\boldsymbol{u}(\boldsymbol{x}, t) = \nabla T(\boldsymbol{x} - \boldsymbol{u}(\boldsymbol{x}, t))\left(T(\boldsymbol{x} - \boldsymbol{u}(\boldsymbol{x}, t)) - R(\boldsymbol{x})\right) \tag{3.9}$$

that is discretized by a semi-implicit scheme with a discrete time step τ, where the nonlinear term is evaluated at the old time level

$$\frac{(\boldsymbol{u}_h^{k+1} - \boldsymbol{u}_h^k)}{\tau} - \alpha\Delta_h \boldsymbol{u}_h^{k+1} = \nabla_h T(\boldsymbol{x} - \boldsymbol{u}_h^k)\left(T(\boldsymbol{x} - \boldsymbol{u}_h^k) - R(\boldsymbol{x})\right). \tag{3.10}$$

The complete image registration scheme can be found in Algorithm 1. Note that in each time step, line 6 of Algorithm 1 requires the solution of d decoupled scalar linear heat equations of type (2.3). This can be accomplished by the same multigrid algorithms as for the image denoising in the last section. To minimize the number of time steps we use a technique described in [15] to adapt the τ and α parameter. The idea is to start with large α and τ (we use $\alpha = 1000, \tau = 10$) penalizing higher oscillations in the solution and preferring global transformations, and then to decrease the parameters by factors $\epsilon_\alpha = 0.1$ and $\epsilon_\tau = 0.5$ when the improvement of the SSD stagnates. Note that for small α the transformations are localized and sensitive to discontinuities or noise in the images.

The development of the relative SSD error for an image registration example is found in Fig. 3.2. As initial deformation for the first time step we take an interpolated solution of the image registration from the next coarser grid, what explains that the initial relative SSD error is below 1.0. Fig. 3.3 shows slices of the corresponding medical data sets and the registration result. For medical applications it is not always useful to drive the registration problem to a very small SSD, but to maintain the topology of the medical data. One time step in the registration algorithm (including 3 linear solves and the computation of the new right hand side and the SSD error) took 1.4 sec and a single solve using a FMG-V(2,1) of our optimized cell-centered multigrid method took 499 ms (FMG-V(2,2) 608 ms, FMG-V(1,1) 390 ms). We start with a FMG-V(2,1), for smaller time steps it is enough to use a FMG-V(1,1) without loosing any accuracy in the solution. In contrast to that a DCT based implementation (described e.g. in [26]) using the FFTW package takes 2107 ms to solve for one of the three components. Here about 65% of the time was spent to compute the forward and backward transforms, the rest by the non-optimized multiplication of the inverse eigenfunctions. Note that in practise sometimes also an additive operator splitting (AOS) scheme is used to solve the registration problem [37, 26]. It is very fast but the time step has to be chosen sufficiently small [26]. We measured 1971 ms to solve for one component in a time step using an unoptimized C implementation.

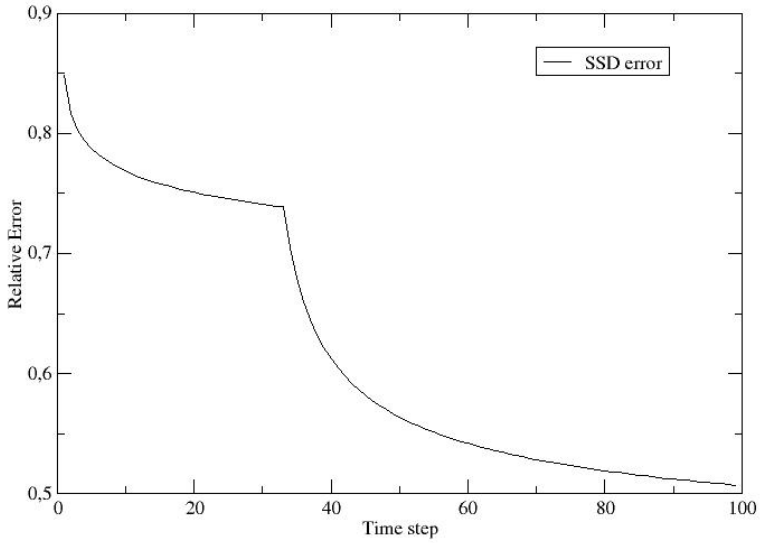

Figure 3.2: Relative SSD error for image registration over time.

4. Conclusions and Further Work. A fast cell-based full multigrid implementation for variational image processing problems is shown to be highly competetive in terms of computing times with alternative techniques such as approaches using FFT-based algorithms. However, this requires a careful machine specific code optimization. Next, this first step has to be extended to an arbitrary number of grid points in each direction, parallelization, and to anisotropic or nonlinear diffusion models.

5. Acknowledgement. This research is being supported in part by the *Deutsche Forschungsgemeinschaft* (German Science Foundation), projects Ru $422/7-1,2,3$.

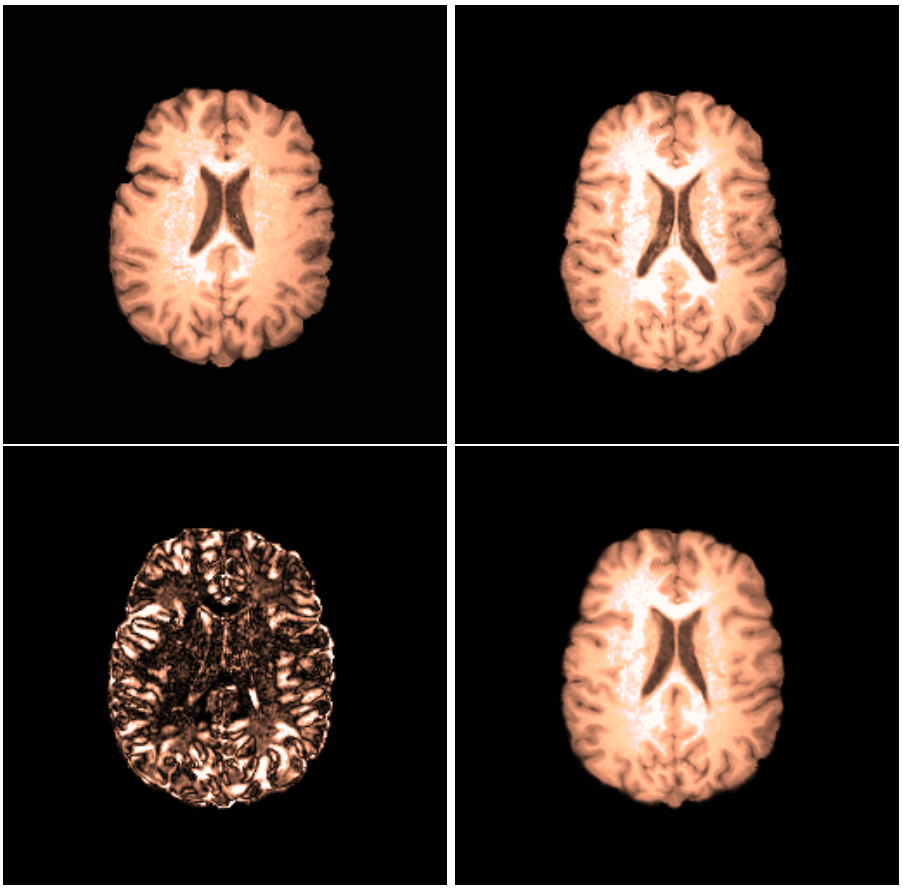

Figure 3.3: Slice of reference image (upper left) template image (upper right), distance image $Tk - R$ (lower left) and registered image (lower right).

REFERENCES

[1] http://www.intel.com/cd/software/products/asmo-na/eng/perflib/mkl/.

[2] http://www.fftw.org.

[3] D. Barkai and A. Brandt, *Vectorized multigrid Poisson solver for the CDC CYBER 205*, Appl. Math. & Comp., 13 (1983), pp. 215–228. (Special Issue, Proceedings of the First Copper Mountain Conference on Multigrid Methods, Copper Mountain, CO, S. McCormick and U. Trottenberg, eds.).

[4] A. Brandt, *Multi-Level Adaptive Solutions to Boundary-Value Problems*, Mathematics of Computation, 31 (1977), pp. 333–390.

[5] W. Briggs, V. Henson, and S. McCormick, *A Multigrid Tutorial*, Society for Industrial and Applied Mathematics, Philadelphia, 2nd ed., 2000.

[6] U. Clarenz, M. Droske, S. Henn, M. Rumpf, and K. Witsch, *Computational methods for nonlinear image registration*, tech. report, Institut fr Mathemtik, Gerhard-Mercator Universität Duisburg, 2006. http://numod.ins.uni-bonn.de/research/papers/public/ClDrHeRuWi04.pdf.

[7] J. W. Cooley and J. W. Tukey, *An algorithm for the machine computation of the complex fourier series*, Mathematics of Computation, 19 (1965), pp. 297–301.

[8] P. Duhamel and M. Vetterli, *Fast fourier transforms: A tutorial review and a state of the art*, Signal Processing, 19 (1990), pp. 259–299.

[9] B. Fischer and J. Modersitzki, *Fast diffusion registration*, Journal of the American Mathematical Society (AMS), (2002), pp. 117–129. http://www.math.uni-luebeck.de/publikationen/pub2002.shtml.

[10] ———, *Curvature based image registration*, Journal of Mathematical Imaging and Vision, 18 (2003), pp. 81 – 85.

[11] M. Frigo and S. G. Johnson, *Fftw: An adaptive software architecture for the fft*, in Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, vol. 3, 1998, pp. 1381–1384.

[12] E. Haber and J. Modersitzki, *A multigrid method for image registration*, tech. report, Institut fr Mathemtik, Universitï¿½ zu Lbeck, Lübeck, 2004. http://www.math.uni-luebeck.de/publikationen/pub2004.shtml.

[13] W. Hackbusch, *Multi-Grid Methods and Applications*, Springer Verlag, 1985.

[14] S. Henn, *A multigrid method for a fourth-order diffusion equation with application to image processing*, SIAM Journal on Scientific Computing, 27 (2005), pp. 831–849.

[15] S. Henn and K. Witsch, *Image registration based on multiscale energy information*, Multiscale Modeling and Simulation, 4 (2005), pp. 584–609.

[16] G. Hermosillo, *Variational Methods for Multi-modal Image Matching*, PhD thesis, Université de Nice, France, 2002.

[17] B. Horn, *Robot vision*, MIT Press, Cambridge, 1986.

[18] F. Jäger, J. Han, J. Hornegger, and T. Kuwert, *A variational approach to spatially dependent non-rigid registration*, in Proc. SPIE, Joseph M. Reinhardt and Josien P. W. Pluim, eds., vol. 6144, February 2006, pp. 860–869.

[19] B. Jähne, *Digitale Bildverarbeitung*, Springer Verlag Berlin, Heidelberg, New York, 6th ed., 2006.

[20] A.K. Jain, *Fundamentals of Digital Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1989.

[21] S. Kabus, A. Franz, and B. Fischer, *On elastic image registration with varying material parameters*, in Proceedings of Bildverarbeitung für die Medizin (BVM), H.-P. Maintzer, H. Handels, A. Horsch, and T. Tolxdorff, eds., Heidelberg, 2005, Springer Verlag, pp. 330–334.

[22] M. Kowarschik, *Data Locality Optimizations for Iterative Numerical Algorithms and Cellular Automata on Hierarchical Memory Architectures*, no. 13 in Advances in Simulation, SCS Publishing House, 2004.

[23] M. Kowarschik, U. Rüde, N. Thürey, and C. Weiss, *Performance Optimization of 3D Multigrid on Hierarchical Memory Architectures*, in Proc. of the 6th Int. Conf. on Applied Parallel Computing (PARA 2002), vol. 2367 of Lecture Notes in Computer Science, Espoo, Finland, Jun 2002, pp. 307–316.

[24] M. Kowarschik, C. Weiss, and U. Rüde, *DiMEPACK — A Cache–Optimized Multigrid Library*, in Proc. of the Int. Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001), H.R. Arabnia, ed., vol. I, Las Vegas, NV, USA, 2001, CSREA, CSREA Press, pp. 425–430.

[25] T. Lehmann, W. Oberschelp, E. Pelikan, and R. Repges, *Bildverarbeitung fr die Medizin*,

Springer, Berlin, 1997.
[26] J. Modersitzki, *Numerical methods for image registration*, Oxford University Press, Oxford, 2004.
[27] M. Mohr and R. Wienands, *Cell-centred multigrid revisited*, Computing and Visualization in Science, 7 (2004), pp. 129–140.
[28] J.-M. Morel and S. Solimini, *Variational methods in image segmentation*, Birkhï¿½ser, Boston, 1995.
[29] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall, 1989.
[30] W. Pennebaker and J. Mitchell, *JPEG: Still Image Data Compression Standard*, Van Nostrand Reinhold, 1993.
[31] C.M. Rader, *Discrete fourier transforms when the number of data samples is prime*, in Proceedings of the IEEE, vol. 56, 1968, pp. 1107–1108.
[32] M. Stürmer, *Optimierung von Mehrgitteralgorithmen auf der IA-64 Rechnerarchitektur*. Lehrstuhl fr Informatik 10 (Systemsimulation), Institut für Informatik, University of Erlangen-Nuremberg, Germany, May 2006. Diplomarbeit.
[33] A.N. Tikhonov and V.Y. Arsenin, *Solution of ill-posed problems*, Winston and Sons, New York, 1977.
[34] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, 2001.
[35] P. Viola and W. M. Wells, *Alignment by maximization of mutual information*, International Journal of Computer Vision, 24 (1997), pp. 137–154.
[36] J. Weickert, *Anisotropic Diffusion in Image Processing*, Teubner, Stuttgart, 1998.
[37] J. Weickert, B. M. ter Haar Romeny, and M. A. Viergever, *Efficient and Reliable Schemes for Nonlinear Diffusion Filtering*, IEEE Transactions on Image Processing, Volume 7, Number 3 (1998), pp. 398–410.
[38] C. Weiss, *Data Locality Optimizations for Multigrid Methods on Structured Grids*, PhD thesis, Lehrstuhl für Rechnertechnik und Rechnerorganisation, Institut für Informatik, Technische Universität München, Munich, Germany, Dec. 2001.
[39] P. Wesseling, *Multigrid Methods*, Edwards, 2004.
[40] R. Wienands and W. Joppich, *Practical Fourier Analysis for Multigrid Methods*, vol. 5 of Numerical Insights, Chapmann and Hall/CRC, 2005. CD included.
[41] Irad Yavneh, *On Red-Black SOR smoothing in multigrid*, SIAM Journal on Scientific Computing, 17 (1996), pp. 180–192.