Image Registration by a Regularized Gradient Flow
A Streaming Implementation in DX9 Graphics Hardware

R. Strzodka, Bonn, **M. Droske**, Duisburg, and **M. Rumpf**, Duisburg

Abstract

The presented image registration method uses a regularized gradient flow to correlate the intensities in two images. Thereby, an energy functional is successively minimized by descending along its regularized gradient. The gradient flow formulation makes use of a robust multi-scale regularization, an efficient multi-grid solver and an effective time-step control.

The data processing is arranged in streams and mapped onto the functionality of a stream processor. This arrangement automatically exploits the high data parallelism of the problem, and local data access helps to maximize throughput and hide memory latency. Although dedicated stream processors exist, we use a DX9 compatible graphics card as a stream architecture because of its ideal price-performance ratio. The new floating point number formats guarantee a sufficient accuracy of the algorithm and eliminate previously present concerns about the use of graphics hardware for medical computing. Therefore, the implementation achieves reliable results at very high performance, registering two 257^2 images in approximately 3sec, such that it could be used as an interactive tool in medical image analysis.

AMS Subject Classification (MSC 2000): 65K10, 65Y10
Key words: image registration, gradient flow, multi-scale, multi-grid, stream processing, graphics hardware computing, DX9 graphics hardware

1 Introduction

Image registration deals with the correlation of the intensities in two images via a usually non-rigid deformation. This deformation may reflect temporal changes in the image source or can compensate for unknown deformation effects of the image acquisition technology. The analysis of temporal changes in anatomic structures in image assisted diagnostics and surgery planning strongly depends on robust registration of images taken at different times.

The optimal correlation between two images depends on the definition of a coherence measure, e.g. the energy $E[u] = \frac{1}{2}\int_\Omega |T \circ \phi - R|^2$, where T, R are the intensity maps of two images and ϕ the deformation, is designed to minimize intensity differences. However, there may be many minimizers to

such a measure. Therefore, many regularizations of the registration problem have been discussed in the literature [3, 5, 8, 12, 16]. The graphics hardware algorithm in this paper follows in its implementation the gradient flow registration presented in [4]. It incorporates the ideas of iterative Tikhonov regularization methods [9], fast multi-grid smoothing [11], and multi-scale use for large displacements [1]. The model will be summarized in the next section. The implementation in this paper focuses on a basic intensity based model. Morphological image matching is to be considered in the future.

Modern graphics hardware can be used for very complex procedural texturing and shading [14, 15] allowing an enormous range of visual effects. But optimization of graphics cards for the processing of large data volumes made them also attractive for scientific computing. We refer to [7] for a comprehensive overview of the literature on general purpose computations on graphics hardware. The most related work in this context is that on multi-grid solvers also discussed in [2, 6], where they have been applied to fluid dynamics. Here, we use the multi-grid hierarchy for both a fast linear equation solver and as an efficient representation of our problem-regularizing multi-scale hierarchy. We also pioneer the graphics hardware based adaptive time-step control governed by Armijo's rule.

Previously the application area of graphics hardware in scientific computing was severly restricted by the low number precision (8 bit) and the limited number of available operations. The DX9 generation has overcome these problems by introducing a floating point number format and a set of the most common mathematical operations. Together with an access from high level languages to these features, most of the code running on common micro-processors could be coded for graphics hardware as well. But this does not mean that every problem can be accelerated in this way, rather the analysis of the problem structure must determine the choice of the appropriate hardware architecture. So while in the past the challenge of graphics hardware accelerated implementations lay in the pure realization of the problem solver in the restricted functionality, today the challenge lies in the construction of problem solvers whose data-flow structure ideally exploits the parallel stream architecture of graphics hardware.

Image processing applications are perfectly suited for parallel streaming implementations on graphics hardware. On the one hand, image sizes are growing with ever higher resolution image acquisition devices, while near real time performance is frequently required, in particular in medical imaging, e.g. in intra surgical image guided diagnosis. On the other hand, in contrast to many other simulation applications numerical accuracy is usually not the ultimate goal in image processing. Instead, one aims for stability of the algorithm and the preservation of the main qualitative effects of the continuous models. This can usually be achieved with lower precision formats, such that the lack of higher internal precision computation and the double float format in graphics hardware is acceptable.

In this paper we prove that complex high level, multi-scale, PDE based methods can be implemented efficiently on recent graphics hardware. Non-rigid image registration is known to be one of the most cost intensive tasks. Basically, all computations in the algorithm, in particular multi-grid V-cycles, computation of energy functionals and their derivatives, time-step control, and image prolongation and restriction are computed on the graphics card, such that we constantly take advantage of its superior memory bandwidth and throughput.

Even considering the newest features in the DX9 graphics hardware, however, graphics cards are still not the ideal platform for the implementation of such stream based algorithms. Reconfigurable computing architectures or dedicated stream processors offer more flexibility in the optimization of the data pipeline for a given application and thus offer higher performance for the same transistor count [10]. However, the price-performance ratio is unrivaled for graphics cards as their mass production drives a constant advancement similar to Moore's Law known for micro-processors but at a more than squared pace, doubling the performance of high-end products in less than 9 months for the same price. Since this development will most likely continue for at least several more years, graphics cards are an ideal basis for inexpensive fast streaming implementations and may even become the main target platform for such tasks, unless reconfigurable computing platforms or dedicated stream processors become more cost efficient.

2 Gradient Flow Registration

2.1 Continuous Model

Given two images, a template and a reference $T, R : \Omega \to \mathbb{R}$, $\Omega \subset \mathbb{R}^2$, we look for a deformation $\phi : \Omega \to \Omega$ which maps the intensities of T via ϕ to the intensities of R such that $T \circ \phi \approx R$. Since ϕ will be small in comparison to $|\Omega|$ it can be suitably expressed as $\phi = \mathbb{1} + u$, with a displacement function u. The displacement u is sought as the minimum of the energy

$$E[u] = \frac{1}{2} \int_\Omega |T \circ (\mathbb{1} + u) - R|^2 \, .$$

A minimizer u in some Banach space $\mathcal{V}$ is characterized by the condition $E'[u] = 0$, where the L^2-representation of E' is given by

$$E'[u] \quad = \quad (T \circ (\mathbb{1} + u) - R) \nabla T \circ (\mathbb{1} + u) \, . \tag{1}$$

This gradient may be used as the descent direction towards a minimum in a gradient descent method. But there may be many minima since any displacements within a level-set of T do not change the energy. Therefore the

descent along the gradient will be regularized by $A(\sigma)^{-1}$, with $A(\sigma) := \mathbb{1} - \frac{\sigma^2}{2}\Delta$ for some $\sigma \in \mathbb{R}^+$. Then the regularized gradient flow

$$\partial_t u = -A(\sigma)^{-1} E'[u]\,,$$

with $u(0) = u_0$, has a unique solution u with $u(t) \in \mathcal{V}$, for some function space $\mathcal{V}$ ([4, Theorem 3.1]).

Despite this uniqueness the gradient descent path may easily get trapped in a local minimum instead of finding the global minimum of E, since the energy E is non-convex. Therefore, a continuous annealing method is used by defining a multi-scale of image pairs

$$T_\epsilon := S(\epsilon)T, \quad R_\epsilon := S(\epsilon)R, \tag{2}$$

for $\epsilon \geq 0$ with a filter operator $S(\cdot)$. The choice $S(\epsilon) = A(\epsilon)^{-1}$ corresponds again to Gaussian filtering. The energy

$$E_\epsilon[u] = \frac{1}{2}\int_\Omega |T_\epsilon \circ (\mathbb{1} + u) - R_\epsilon|^2 \tag{3}$$

induces the corresponding gradient flow on scale ϵ, which has the solution $u_\epsilon(\cdot)$.

2.2 Discretization

Time is discretized by the explicit Euler scheme

$$\frac{u_\epsilon^{n+1} - u_\epsilon^n}{\tau_\epsilon^n} = -A(\sigma)^{-1} E_\epsilon'[u^n]\,,$$

where τ_ϵ^n is determined by Armijo's rule.

$$\frac{E_\epsilon[u_\epsilon^n] - E_\epsilon[u_\epsilon^{n+1}]}{\tau_\epsilon^n \langle E_\epsilon'[u_\epsilon^n], A^{-1} E_\epsilon'[u_\epsilon^n]\rangle} \quad \geq \quad c\,, \tag{4}$$

for $c \in (0, \frac{1}{2})$. This allows an adaptive acceleration of the gradient descent towards a minimum.

Finite-Elements are used for the discretization in space. Let $\{\Psi^i\}_{i \in I_h}$ be the canonical nodal basis of the linear finite element space $\mathcal{V}^h$. Suppose $\bar{U}^n$ is the nodal vector at the n-th time-step. Then we obtain the fully discrete scheme

$$\bar{U}_\epsilon^{n+1} = \bar{U}_\epsilon^n - \tau_\epsilon^n A_h(\sigma)^{-1} \bar{E}_\epsilon'[\bar{U}_\epsilon^n]\,, \tag{5}$$

where the matrix $A_h(\sigma)$ is the discrete counterpart of the operator $A(\sigma)$ in $\mathcal{V}^h$ [4]. We apply this formula to compute an approximate solution $\bar{U}_\epsilon^{N_\epsilon}$ on

scale ϵ by iterating it N_ϵ times until the update is sufficiently small in the L^2 norm:

$$\left\|\tau_\epsilon^n A_h(\sigma)^{-1} \bar{E}'_\epsilon[\bar{U}_\epsilon^{N_\epsilon}]\right\|_2^2 \quad < \quad \delta\,. \tag{6}$$

Since multi-grid solvers are the most efficient tools in solving linear systems of equations, the gradient smoothing $A_h(\sigma)^{-1}\bar{E}'_\epsilon[\bar{U}_\epsilon^n]$ is performed as a multi-grid V-cycle with Jacobi iterations as smoother and standard prolongation and restriction operators. Indeed, to ensure an appropriate regularization it suffices to consider only a single multi-grid V-cycle

$$\mathrm{MGM}(\sigma) \quad \approx \quad A_h(\sigma)^{-1} \tag{7}$$

with few intermediate Jacobi steps on each grid. The same applies to the generation of the multi-scale of images (Eq. 2) with the filter operator $S(\cdot) = A(\epsilon)^{-1}$.

The image scales are chosen exponentially increasing, i. e. we consider scales $(\epsilon_k)_{k=0,\ldots,K}, K \in \mathbb{N}$ with $\epsilon_0 = 0$ being the finest and ϵ_K the coarsest scale. We reduce the number of necessary computations for large scales by defining a pyramid of grids $(\Omega_{h_l})_{l=0,\ldots,L}$, $h_l = 2^{l-L}$ and resolving the images T_{ϵ_k} and R_{ϵ_k} on the coarsest grid Ω_{h_l} for which

$$\epsilon_k \quad \geq \quad \alpha \cdot h_l \tag{8}$$

for some $\alpha \in [\frac{1}{2}, 2]$ still holds (cf. Figure 1), i.e. we have a monotone mapping $l : \epsilon_k \to l(\epsilon_k)$. Thus computations on coarse scales are performed on small grids and only the last few scales represented on the finest grid Ω_{h_0} consume the full processing power. For very coarse scales the function l is bounded from above

$$\forall k : l(\epsilon_k) \leq L_0 \quad < \quad L\,, \tag{9}$$

to avoid the computation of initial deformations on very small grids, e.g. Ω_{h_L} is only a 2x2 grid. Typical choices for the initial grid are $L - L_0 \in \{2, 3\}$. Naturally, the multi-grid V-cycle (Eq. 7) is not affected by this bound and uses all grids in the hierarchy. For further details of the algorithm we refer to [4].

3 Hardware Implementation

3.1 Some fundamentals

In the following we give a simplified view on the DX9 graphics pipeline to facilitate the understanding of the implementation for those unfamiliar with the architecture. The DX9 graphics pipeline contains two main distinct programmable parts, the vertex and the fragment processor (Figure 2). The

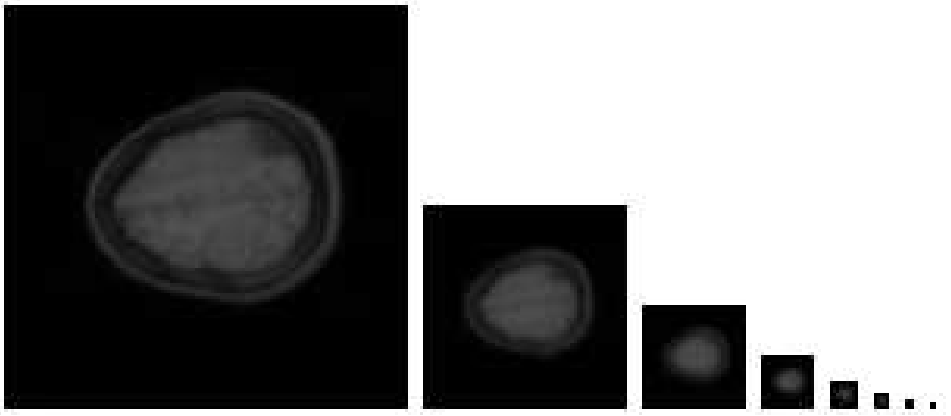

Figure 1: The multi-grid hierarchy encoded in textures of different spatial resolution. Each grid level serves for the representation of several scales of the multi-scale hierarchy.

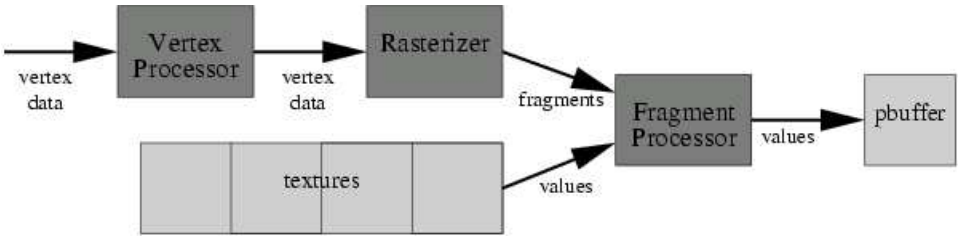

Figure 2: A simple diagram of the DX9 graphics pipeline. Light gray represents data containers, dark gray processing units. In each pass a different texture can serve as the target pbuffer for the output data stream.

vertex processor takes input data associated with a vertex, such as vertex and texture coordinates, normals, colors, lighting coefficients. Each vertex is processed separately with no interaction to other vertices. The vertex program uses the input values to compute the vertex window position, two colors and eight texture coordinates. The hard-wired rasterizer interpolates the above values for each pixel enclosed by the vertices of the figure which is currently being drawn, e.g. a triangle. These interpolated values comprise a fragment and are sent individually to the fragment processor.

The fragment processor can perform many mathematical operations on the values and can also retrieve additional data from arbitrary positions of up to 16 textures. The computations, however, cannot alter the window position but only decide which final color will cover the pixel at that position or choose to discard the fragment.

As the vertices and fragments are processed individually they form streams

of data which may be quickly processed in parallel. The processing of streams achieves highest performance if the data streams are large and the texture accesses in the fragment processor use only neighboring texture values, such that bandwidth-efficient memory burst-modes can be used and latency can be hidden with small local caches. The ingredients of the gradient flow registration impose a data-flow which fulfills these conditions very well.

3.2 Data-flow

The two dimensional input images T and R are represented as 2D textures on the finest grid Ω_{h_0}. The multi-grid hierarchy $(\Omega_{h_l})_{l=0,\dots,L}$ corresponds to textures of successively smaller size (Figure 1). Several such hierarchies are reserved in graphics memory to store any intermediate results, because once the two images T and R are stored in graphics memory all operations are performed on the graphics card. In the end the final result is displayed directly from the graphics memory onto the screen, such that no image transfer between the main memory and the graphics card takes place. This is advantageous, since the memory bandwidth on-board is still much higher than over the AGP bus.

All textures are implemented as floating point pbuffers, i.e. graphics buffers which can be used alternately as a source or destination of a data stream. Computations are performed by loading a computational kernel to the programmable fragment processor, e.g. a prolongation kernel, and streaming the texture operands through that kernel into a target pbuffer. Thereby the vertex processor is used to generate the texture coordinates for the access to neighboring values in textures. The target pbuffer can then be used as a texture operand in the succeeding operation (Figure 2).

3.3 Algorithm

The algorithm consists of up to three nested loops, the scale loop (Eq. 10), the update loop (Alg. 11), and the independent inner loops of the multi-grid V-cycle (Eq. 7) and the energy evaluation by Armijo's rule (Eq. 12).

The scale loop with index k runs from the coarse (ϵ_K) to the fine (ϵ_0) scale representations and uses the prolongation operator to transfer data onto finer grids. It starts by setting the initial displacement $\bar{U}^0_{\epsilon_K}$ on the coarsest scale ϵ_K to zero. Then the gradient flow (Alg. 11) at this scale computes from this vector the approximate solution $\bar{U}^{N_{\epsilon_K}}_{\epsilon_K}$. This solution is used as the initial displacement $\bar{U}^0_{\epsilon_{K-1}}$ at the next finer scale ϵ_{K-1}. This process

$$\begin{aligned} \bar{U}^0_{\epsilon_K} &:= \bar{0}\,, \\ \bar{U}^0_{\epsilon_{k-1}} &:= \bar{U}^{N_{\epsilon_k}}_{\epsilon_k} \end{aligned} \tag{10}$$

continues for $k = K, \ldots, 1$ until the final solution $U_{\epsilon_0}^{N_{\epsilon_0}}$ on the finest scale ϵ_0 is obtained. If the scales $\epsilon_k, \epsilon_{k-1}$ are represented on different grids (cf. Eq. 8), i.e. $l(\epsilon_{k-1}) < l(\epsilon_k)$, then a prolongation operator is used to transfer the vector from the coarser grid $\Omega_{h_{l(\epsilon_k)}}$ to the finer $\Omega_{h_{l(\epsilon_{k-1})}}$.

The update loop with index n performs the gradient descent on a fixed scale ϵ (we therefore omit the index k) until the change in data becomes sufficiently small (Eq. 6). We list the implementation of the discrete scheme (Eq. 5) for the gradient flow problem on scale ϵ in pseudo-code notation:

```
gradient flow at scale ε {                                              (11)
  compute new image scales T̄_ε = MGM(ε)T̄, R̄_ε = MGM(ε)R̄;
  for each n {
    evaluate energy gradient Ē'_ε[Ū^n_ε] (Eq. 1);
    perform smoothing multi-grid V-cycle MGM(σ)Ē'_ε[Ū^n_ε];
    evaluate τ^n_ε by Armijo's rule (Eq. 12);
    compute new solution Ū^{n+1}_ε = Ū^n_ε − τ^n_ε MGM(σ)Ē'_ε[Ū^n_ε];
    break loop if ‖τ^n_ε MGM(σ)Ē'_ε[Ū^n_ε]‖_2^2 < δ;
  }
}
```

The first inner loop performs the multi-grid V-cycle (Eq. 7). Starting on the current grid $\Omega_{h_{l(\epsilon_k)}}$ we apply the Jacobi smoother a few times to the afore computed energy gradient (Eq. 1). Then the remaining residuum is restricted onto the next coarser grid $\Omega_{h_{l(\epsilon_k)+1}}$. This process of Jacobi iterations with succeeding restriction of the remaining residuum to the next coarser level continues until the coarsest grid level Ω_{h_L} is reached. Here we can solve the linear equation system exactly. The solution is prolonged to the next finer level and added to the result of the previous Jacobi iterations on this level. The Jacobi smoother is then applied to the sum several times and the result is again prolonged to the next finer level. We continue in this way until we arrive at the grid $\Omega_{h_{l(\epsilon_k)}}$ from which we started. The result of the Jacobi smoother on this level is the final result of the multi-grid V-cycle.

The second inner loop, the energy loop, determines for each update the maximal time-step width $\tau_{\epsilon_k}^n$ which satisfies Armijo's rule (Eq. 4), i.e. we maximize τ in

$$\begin{aligned} &\bar{E}_{\epsilon_k}[\bar{U}_{\epsilon_k}^n] - \bar{E}_{\epsilon_k}\left[\bar{U}_{\epsilon_k}^n - \tau \mathrm{MGM}_{l(\epsilon_k)}(\sigma_k)\bar{E}'_{\epsilon_k}[\bar{U}_{\epsilon_k}^n]\right] \\ &\quad \geq \quad c\tau \left\langle \bar{E}'_{\epsilon_k}[\bar{U}_{\epsilon_k}^n], \mathrm{MGM}_{l(\epsilon_k)}(\sigma_k)\bar{E}'_{\epsilon_k}[\bar{U}_{\epsilon_k}^n] \right\rangle_{h_{l(\epsilon_k)}} . \end{aligned} \tag{12}$$

In the above formula only the energy $\bar{E}_{\epsilon_k}\left[\bar{U}_{\epsilon_k}^n - \tau \mathrm{MGM}_{l(\epsilon_k)}(\sigma_k)\bar{E}'_{\epsilon_k}[\bar{U}_{\epsilon_k}^n]\right]$ which depends non-linearly on τ needs to be recomputed iteratively.

3.4 Computational Kernels

The different parts of the algorithm are performed by streaming texture operands through different fragment processor kernels:

```
 1 FragOut
 2 jacobiFP( Frag2dIn IN,
 3           uniform sampler2d Tex_B : texunit0,
 4           uniform sampler2d Tex_X : texunit1,
 5           uniform float scale)
 6 {
 7   FragOut OUT;
 8
 9   float2 tex_B= f2texRECT(Tex_B, IN.cCoord.xy);
10   Stencil3x3r2 tex_X; texStar8(tex_X, IN, Tex_X);
11
12   float2 LN= ( + tex_X.mp + tex_X.cp + tex_X.pp
13                + tex_X.mc +          + tex_X.pc
14                + tex_X.mm + tex_X.cm + tex_X.pm )*(1/8.);
15   OUT.col= lerp(LN, tex_B, scale);
16
17   return OUT;
18 }
```

Listing 1: Implementation of the kernel of the Jacobi solver for $A_h X = B$ in the graphics language Cg. Bold keywords belong to the language specification, italic ones are predefined types and functions of a self-made library which facilitates the access to neighboring nodes in a texture. Lines 9,10 assign local variables to data elements of the input data streams (given by the textures `Tex_B,Tex_X`), and the following lines define the actual processing of the data elements with the computation of the convolution and the linear interpolation: `lerp(a,b,c):=(1-c)a+cb`.

- Smoothing with the multi-grid V-cycle $\mathrm{MGM}_\epsilon(\sigma)$:
  - Prolongation operator.
  - Restriction operator.
  - Residuum computation $\bar{U}_\epsilon^n - A_h \bar{X}_\epsilon$.
  - Jacobi iterations with A_h.
- Energy functional:
  - Computation of the error $e_\epsilon := \bar{T}_\epsilon \circ (\mathbb{1} + \bar{U}_\epsilon^n) - \bar{R}_\epsilon$.
  - Computation of the energy gradient $\bar{E}'_\epsilon[\bar{U}_\epsilon^n]$.
- Utilities:
  - Multiply and accumulate.
  - Evaluation of the lumped L^2 scalar product $\langle ., . \rangle_h$.

All these kernels have been programmed in Cg [13], a high level graphics programming language. Listing 1 shows the implementation of the Jacobi kernel for the fragment processor. A different program in the vertex processor generates the texture coordinates in the structure `Frag2dIn IN` for the access to the neighboring nodes. The other parameters of `jacobiFP` are set in the application during the configuration process of the graphics pipeline (Figure 2). Listing 2 shows the pipeline configuration in a C++

```
 1 tex[TEX_N].toTexture(MGlev);
 2
 3 cgGLSetStateMatrixParameter(verVar[VP_NEIGH2D][VV_MVP],
 4   CG_GL_MODELVIEW_PROJECTION_MATRIX, CG_GL_MATRIX_IDENTITY);
 5 cgGLSetParameter4fv(fragVar[FP_JACOBI][FV_SCALE], scale);
 6 cgGLBindProgram(verProg[VP_NEIGH2D]);
 7 cgGLBindProgram(fragProg[FP_JACOBI]);
 8
 9 tex[TEX_B].bind(MGlev, GL_TEXTURE0_ARB);
10 tex[TEX_X].bind(MGlev, GL_TEXTURE1_ARB);
11
12 drawTex(tex[TEX_N].pos[MGlev], tex[TEX_N].size[MGlev],
13          tex[TEX_B].pos[MGlev], tex[TEX_B].size[MGlev],
14          tex[TEX_X].pos[MGlev], tex[TEX_X].size[MGlev]);
15
16 tex[TEX_N].fromTexture(MGlev);
```

Listing 2: Configuration of the graphics pipeline and data streams for the execution of the Jacobi kernel in Listing 1. Bold keywords are functions of the Cg API, italic once are predefined arrays pointing to texture objects, vertex and fragment programs and their variables. The first line sets the target pbuffer for the output data stream. At the end (line 16) we release the pbuffer, such that it can be used as a texture operand in the next pass. Lines 6,7 configure the vertex and fragment processor with the kernel programs. Line 5 sets the `scale` parameter, lines 9,10 bind the textures `TEX_B,TEX_X` as input data streams for `jacobiFP` (Listing 1). Finally, line 12 sends the geometry of the current multi-grid level (`MGlev`) to the vertex processor and thus initiates the execution of the Jacobi iteration in the graphics pipeline (Figure 2).

program for the execution of one iteration of the Jacobi solver. This dual programming model given by 'configware' which configures the processing elements, and 'flowware' which prescribes the flow of the data streams is typical for data stream based architectures [10]. It has the great advantage that the individual elements of the data streams are assembled from memory before the actual processing. This allows the optimization of the memory access patterns, minimizing latencies and maximizing the sustained bandwidth. Unimodal software programs used for instruction stream based architectures, e.g. micro-processors, allow only a limited prefetch of the input data, based on predictions of conditional jumps in the instruction stream.

All kernels perform their task in one pass except for the lumped discrete L^2 scalar product. Usually, it is evaluated by a component-wise multiplication and an iterative addition of local texels, but such a procedure involves a global access to all texels of a texture and would need a global register for accumulation, which is currently not supported in graphics hardware. Hence, we consider a hierarchical implementation with several passes. After the component-wise multiplication we consecutively halve the size of the re-

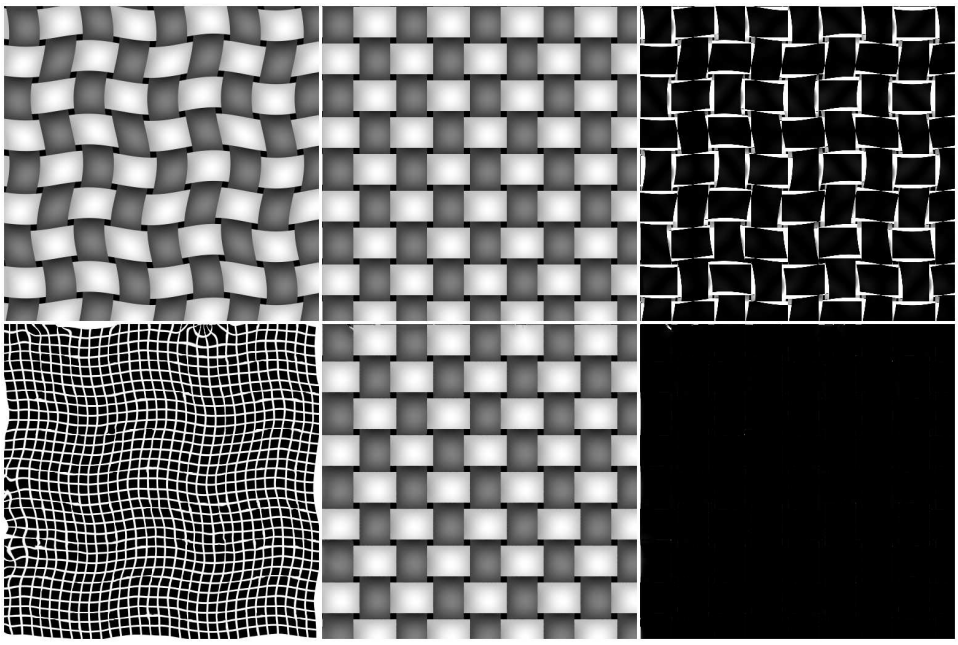

Figure 3: Elimination of low frequency deformations on 513^2 images in 9.8sec. We see that apart from the image boundary where sufficient information is missing, the deformation can be completely eliminated.

sulting texture by applying local filters which sum up the local texel values. This step is repeated from the finest up to the coarsest grid level such that the final result of this hierarchical summation can be retrieved from the coarsest level as a single value for further processing by the CPU.

The energy computation $E_{\epsilon_k}\left[\bar{U}^n_{\epsilon_k} - \tau \mathrm{MGM}_{l(\epsilon_k)}(\sigma_k)\bar{E}'_{\epsilon_k}[\bar{U}^n_{\epsilon_k}]\right]$ required in the evaluation of Armijo's rule (Eq. 12) requires such a lumped discrete L^2 scalar product. Thus we compute

$$\begin{aligned} \bar{V}_\tau &:= \bar{U}^n_{\epsilon_k} - \tau \mathrm{MGM}_{l(\epsilon_k)}(\sigma_k)\bar{E}'_{\epsilon_k}[\bar{U}^n_{\epsilon_k}]\,, \\ E_{\epsilon_k}\left[\bar{V}_\tau\right] &= \frac{1}{2}\langle \bar{V}_\tau, \bar{V}_\tau\rangle_{h_{l(\epsilon_k)}}\,, \end{aligned}$$

where the scalar product is implemented as the hierarchical summation described above.

3.5 Results

Now, we present various results of our registration method. The corresponding figures show six different tiles, which are are arranged in the following way: on the upper left we see the template which should be deformed to fit the reference image to the right of it; on the lower left we see the computed

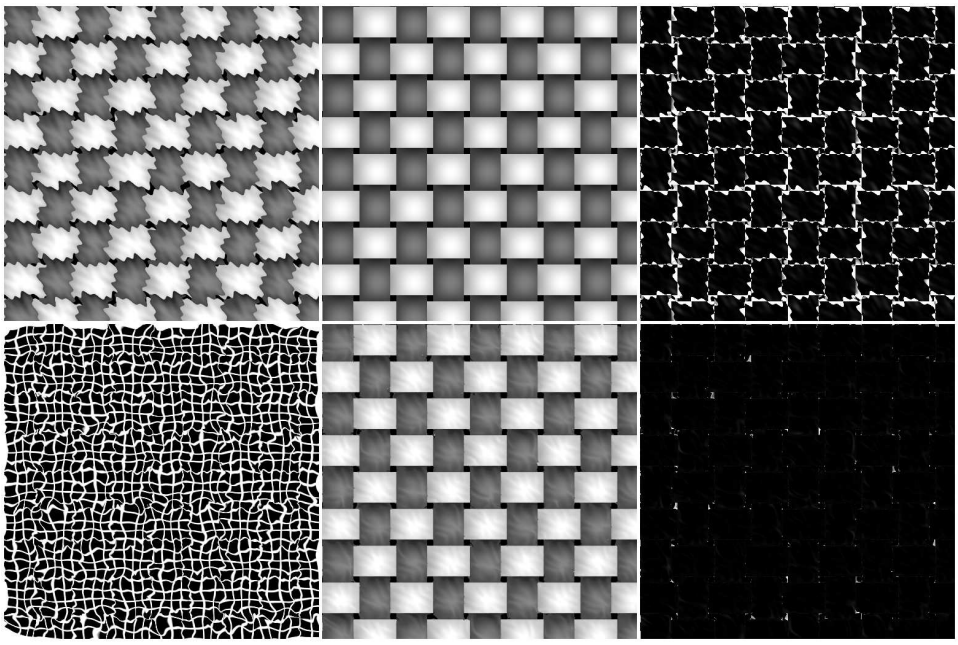

Figure 4: Elimination of strong high frequency distortions on 513^2 images in 9.5sec. The algorithm performs well for high frequencies. Only some ambiguous situations around the strongly deformed small dark squares cannot be resolved.

deformation applied to a uniform grid and to the right the registration result, i.e. the template after the deformation. The rightmost column shows the quadratic difference between the template and the reference image before (upper row) and after (lower row) the registration. With one exception the differences are scaled with a factor 10 and grey values are clamped at black and white, otherwise one would hardly see anything on the error images after registration.

We have tested the algorithm with three different data sets: low and high frequency distortions (Figures 3, 4), large rigid deformations (Figures 5, 6) and medical data sets (Figures 8, 9).

All examples use the same parameter set. Naturally the registration results can be further improved or accelerated by optimizing the parameters for a given problem, but then the time and quality necessary to obtain certain registration results would depend on the hardly quantizable abilities of the user to find the best parameters. The standard parameter set uses 15 different scales, up to 10 iterations of the update loop and up to 10 iterations of the energy loop. We say 'up to' because the loops are aborted if the update is too small. The smoothing multi-grid V-cycle uses 3 Jacobi iterations on each grid both up and down the V-cycle. During the registration process the

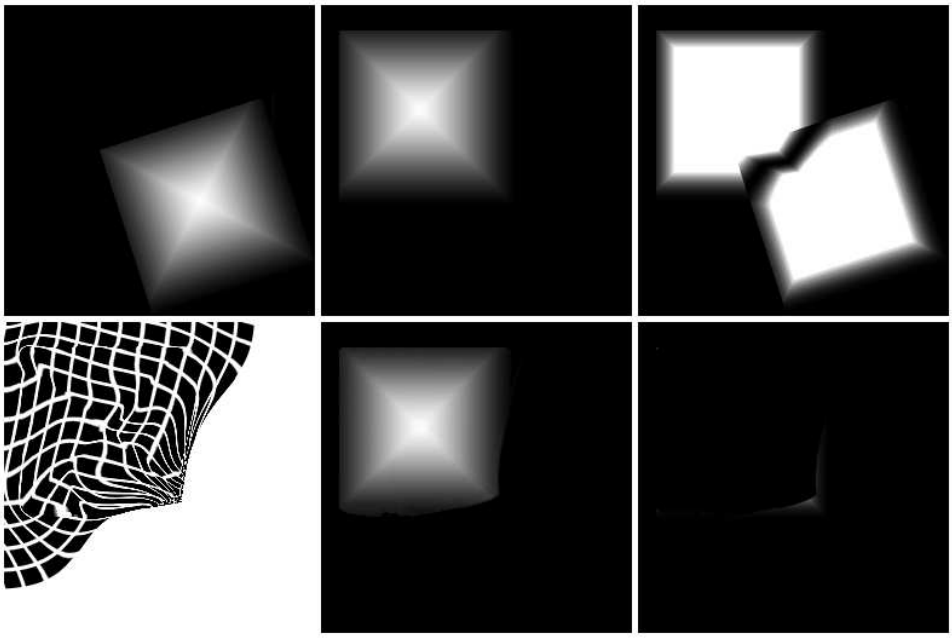

Figure 5: Registration of a large scale rigid deformation on 257^2 images in 3sec. The multi-scale approach allows to reconstruct even large deformations. However, without additional assumptions on the deformation model, the computed deformation might significantly differ from our expectations.

parameter σ_k is exponentially decreasing similar to the scales ϵ_k, such that $\sigma_k^2/h_{l(\epsilon_k)}^2 \in [1,4]$ and thus the condition of $A_{h_{l(\epsilon_k)}}(\sigma_k)$ is uniformly bounded.

Figures 7 and 11 show the decrease of the energy against the overall number of update iterations in the process of registering the examples 6 and 9 respectively. Each point in the graph stands for a gradient descent step (Eq. 5), which includes the computation of the energy gradient, the smoothing with the multi-grid V-cycle, evaluation of Armijo's rule and the update of the solution (cf. Alg. 11). The graph discontinuities indicate scale changes ($\epsilon_k \to \epsilon_{k-1}$), while the X's on the x-axis represent changes of the grid level in the multi-grid hierarchy ($\Omega_{h_l} \to \Omega_{h_{l-1}}$). The number of the X's depends on the choice of the grid used for the computation of the initial deformation (Eq. 9).

Usually each scale change increases the energy, because less smoother data is used in the computation of the differences (Eq. 3). This effect can be sometimes particularly large for the last scale change, because on scale $\epsilon_0 = 0$ no smoothing of the images takes place. Introducing more scales, especially for the finest grid can lessen this effect, such that the energy graph looks nicer, but since the overall energy is hardly reduced in this way, we have not included these additional costly iterations on the finest grid

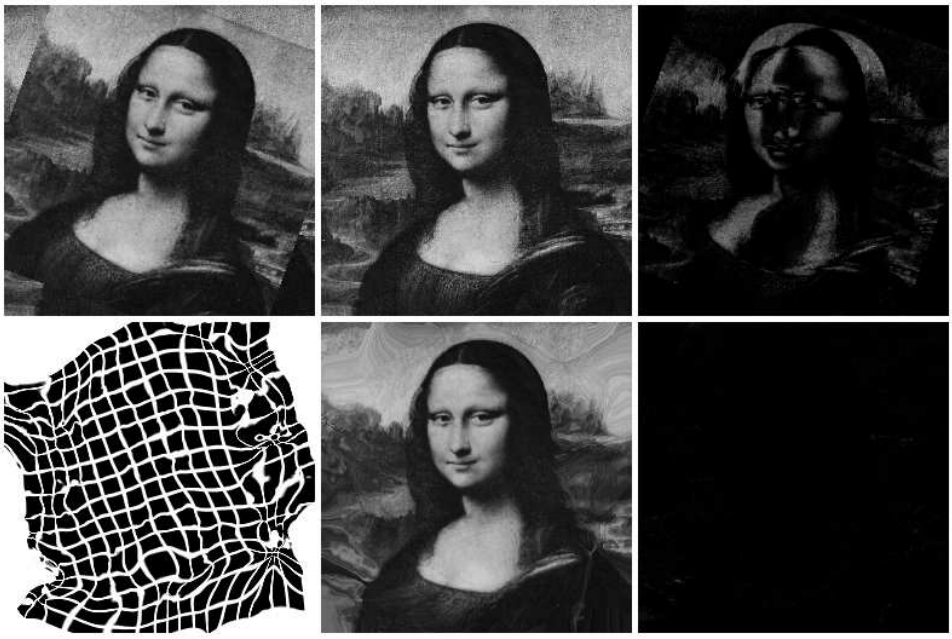

Figure 6: Registration of a rotated 257^2 image in 5.9sec. This is a very hard test for the non-rigid registration, which takes exceptionally long to finish for this image size, since without any a priori knowledge about the underlying rotation there are many possibilities to match the similar grey levels against each other. Obviously in the area of the body the inverse rotation could be identified by the algorithm, whereas the background is rather poorly registered. This outlines that even without the guidance of a concrete deformation model the method performs well for large deformation if the structures to be matched are sufficiently pronounced.

in the standard parameter set. Sometimes we observe energy decreases at the time we change the grid level in the multi-grid hierarchy (Figure 11). We reckon that this effect is due to the additional smoothing caused by the prolongation operator, which decreases the local error in areas of smooth deformations.

3.6 Performance

The implementation depends on the programmability of the computational kernels from Subsection 3.4 and thus requires a DX9 compatible graphics card. We have used a card powered by the GeForceFX 5800 Ultra chip from NVIDIA. The computations were performed either in the standard s23e8 full float or a graphics specific s10e5 half float format. The results for both formats are very similar which is an indicator for the stability of the

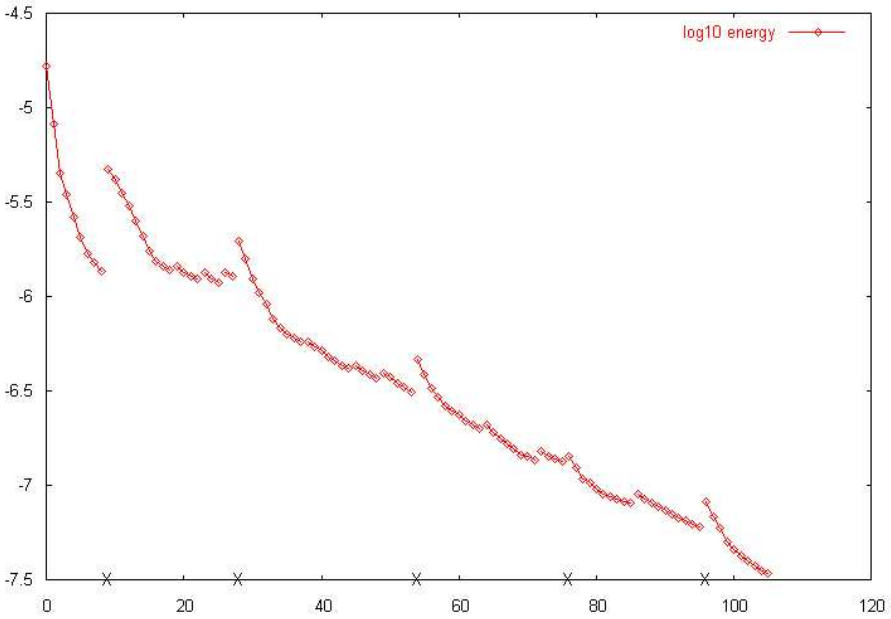

Figure 7: The decrease of the energy against the overall number of update iterations (Eq. 5) in the process of registering the images from Figure 6. The structure of the graph is explained at the end of Subsection 3.5.

algorithm. The performance also varies only slightly in favor of the smaller format as there exists a sufficient bandwidth in comparison to the number of operations in the kernels to transport the full float values.

The complete initialization of the program, including the reading of the images from the hard disk and the configuration of the graphics hardware takes up to 5sec. The duration of the registration itself depends on the size of the images and the number of actual passes, since several loops allow adaptive loop abortion. In general, the registration of 257^2 images takes approx. 3sec and up to 10sec are needed for fully distorted 513^2 images. But for even larger (513×769) medical data, often less time is required (8.5sec) because for such data the deformations are usually not so severe (Figure 9). An estimated software performance for this data set based on the highly optimized implementation in [4], which actually deals with 3D data, would amount to 34sec and thus 4 times more time than the graphics implementation requires.

We expect this factor to grow rapidly in the future, because graphics hardware performance is increasing much faster than the predictions of Moore's Law for micro-processors. Beside higher clock frequencies and wider data buses NVIDIA's GeForce 6800 and ATI's Radeon X800 already execute up to eight times as many operations per clock cycle as the graphics

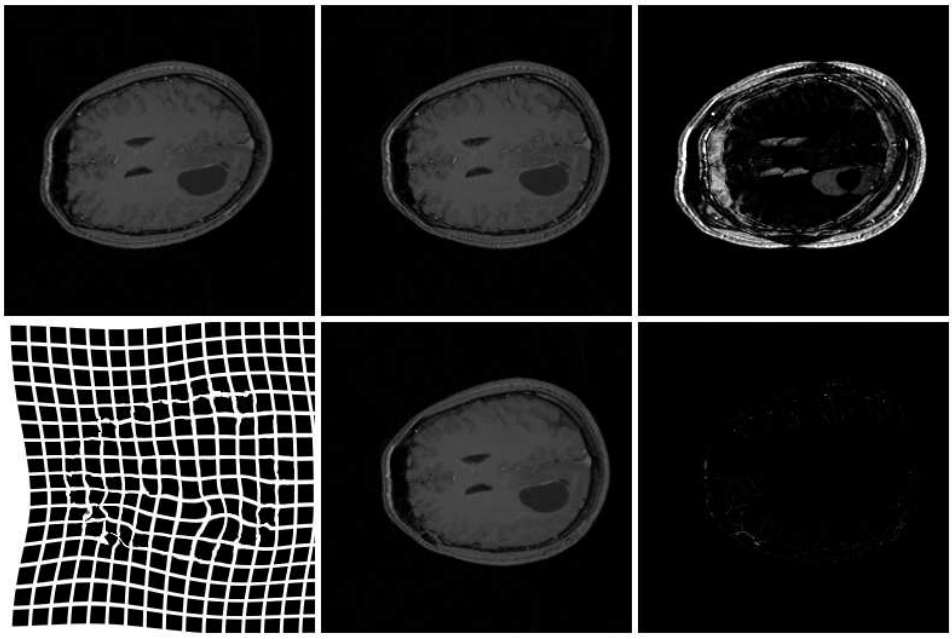

Figure 8: Elimination of a possible acquisition artifact for a medical 257^2 image in 2.2sec. Here we have an example were some areas must expand while others must shrink to fit the reference image. The matching works well apart from the small fluctuation in the lower left part. Such deviations are striking to our perception but have little effect on the overall energy, because they reflect mismatches of the morphology rather than grey level deviations. Therefore additional morphological energy components will be considered in the future.

processor we used. Moreover, the performance of the algorithm is currently also restricted by the lack of an appropriate functionality in the interface. It is an known problem that switching between different pbuffers as destinations requires a general context switch and is therefore unnecessarily slow. This means that our algorithm is often busy switching context data, instead of doing the actual computations, which themselves are very fast. This bottleneck will disappear as soon as the currently developing mechanism for pbuffer switches within a context has been included in the graphics interface.

4 Conclusions

An implementation of the non-rigid gradient flow registration in DX9 graphics hardware has been presented. The performance of such a stream architecture has been exploited by implementing the algorithm in the form of

computational kernels which operate on large streams of data coming from and directed to pbuffers. Moreover, the multi-scale representation and the multi-grid solvers can be realized very naturally in graphics hardware. The use of floating point number formats also offers sufficient precision for the implementation of an adaptive time-step control and early loop abortion. Missing functionality in the graphics interface currently limits performance, but even now large images can be registered in a few seconds, and image assisted diagnostics could already benefit from this tool.

A transfer of the algorithm to 3 dimensional data sets is fairly straight forward, however, in 3 dimensions even the high throughput of current graphics processors cannot compensate the growth in data volume for high resolution images. Hence, we are aiming for an adaptive image processing in graphics hardware. Furthermore, we will consider the efficient matching of image morphologies, which would enable the registration of images of different image modalities, such as CT and MRI.

Acknowledgments

Marc Droske was supported by the DFG within the special research program on time sequence analysis and image processing.

References

[1] L. Alvarez, J. Weickert, and J. Sánchez. Reliable estimation of dense optical flow fields with large displacements. *International Journal of Computer Vision*, 39:41–56, 2000.

[2] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. In *Proceedings of SIGGRAPH 2003*, 2003.

[3] G. E. Christensen, S. C. Joshi, and M. I. Miller. Volumetric transformations of brain anatomy. *IEEE Trans. Medical Imaging*, 16, no. 6:864–877, 1997.

[4] U. Clarenz, M. Droske, and M. Rumpf. Towards fast non-rigid registration. In Z. Nashed and O. Scherzer, editors, *Contemporary Mathematics, Special Issue on Inverse Problems and Image Analysis*. AMS, 2002.

[5] C. A. Davatzikos, R. N. Bryan, and J. L. Prince. Image registration based on boundary mapping. *IEEE Trans. Medical Imaging*, 15, no. 1:112–115, 1996.

[6] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys. A multigrid solver for boundary-value problems using programmable

graphics hardware. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, 2003.

[7] GPGPU. GPGPU - general purpose computation using graphics hardware. http://www.gpgpu.org/. Mark J. Harris (Ed.).

[8] U. Grenander and M. I. Miller. Computational anatomy: An emerging discipline. *Quarterly Appl. Math.*, LVI, no. 4:617–694, 1998.

[9] M. Hanke and C. Groetsch. Nonstationary iterated Tikhonov regularization. *J. Optim. Theory and Applications*, 98:37–53, 1998.

[10] R. Hartenstein. Data-stream-based computing: Models and architectural resources. In *International Conference on Microelectronics, Devices and Materials (MIDEM 2003)*, Ptuj, Slovenia, Oct. 2003.

[11] S. Henn and K. Witsch. Iterative multigrid regularization techniques for image matching. *SIAM J. Sci. Comput. (SISC)*, Vol. 23 no. 4:pp. 1077–1093, 2001.

[12] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens. Multi–modal volume registration by maximization of mutual information. *IEEE Trans. Medical Imaging*, 16, no. 7:187–198, 1997.

[13] NVIDIA. Cg programming language. http://developer.nvidia.com/view.asp?PAGE=cg_main, 2002.

[14] M. S. Peercy, M. Olano, J. Airey, and P. J. Ungar. Interactive multi-pass programmable shading. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings,*, Annual Conference Series, pages 425–432. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[15] K. Proudfoot, W. R. Mark, S. Tzvetkov, and P. Hanrahan. A real-time procedural shading system for programmable graphics. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 159–170. ACM Press / ACM SIGGRAPH, 2001.

[16] J. P. Thirion. Image matching as a diffusion process: An analogy with Maxwell's demon. *Medical Imag. Analysis*, 2:243–260, 1998.

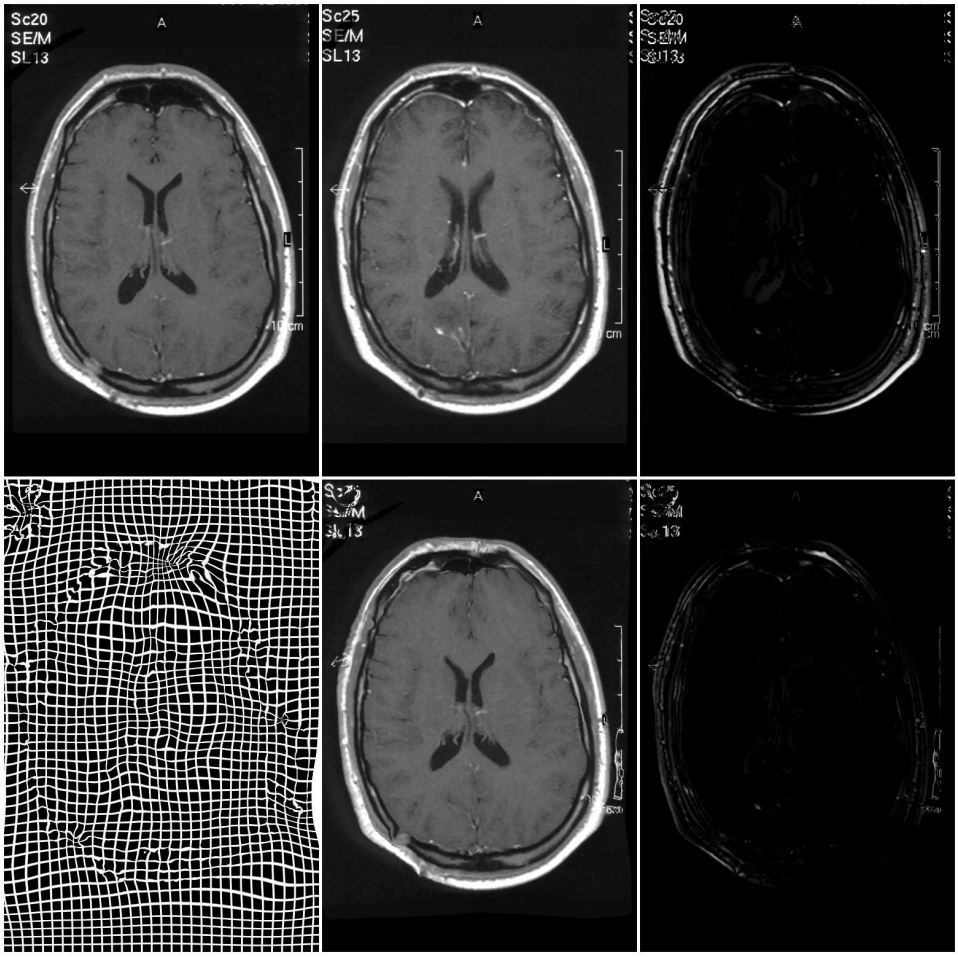

Figure 9: Registration of two brain slices of the same patient taken at different times on 513×769 images in 8.5sec. As the slices were extracted from 3D volumes some anatomical structures are not present in both images and lead necessarily to an error in the registration result, especially in the upper right parts of the images. Also in contrast to the other examples the grey values of the corresponding structures have not exactly the same value such that a small error is present even in the case of perfect fit of image edges (left scull). For this reason, here the error images on the right are not scaled. The algorithm, however, is not distracted by the different grey levels and complements missing data fairly smoothly. In particular the edges are matched nicely. Figure 10 shows the scaled error in the interior of the images.

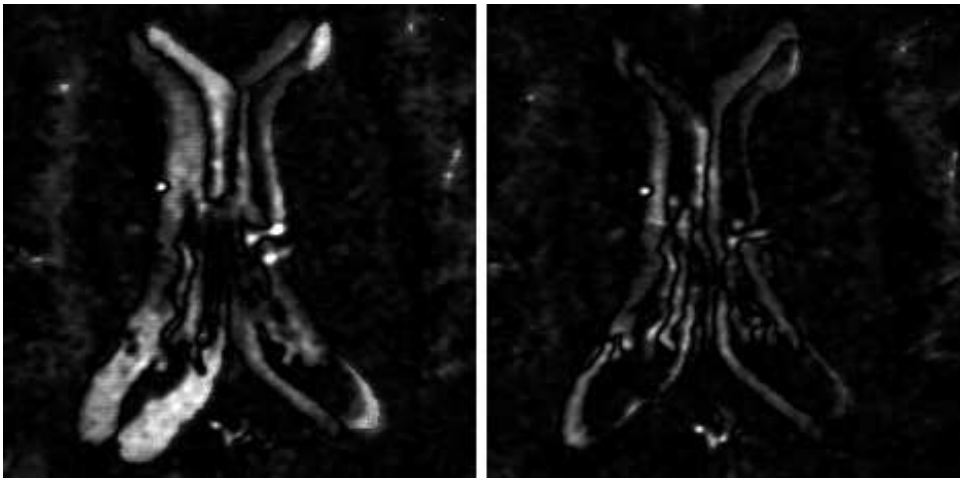

Figure 10: These images show the enlarged central part of the error images from Figure 9. Here the error has been multiplied by 10 again. The images demonstrate that the algorithm does also a good job in matching the central areas.

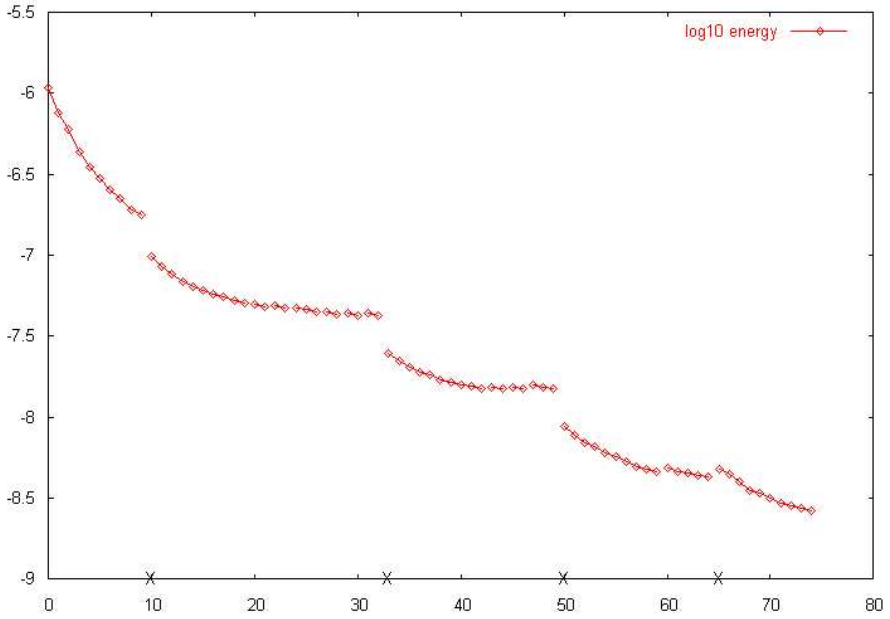

Figure 11: The decrease of the energy against the overall number of update iterations (Eq. 5) in the process of registering the images from Figure 9. The structure of the graph is explained at the end of Subsection 3.5.

Robert Strzodka
caesar research center
Postfach 120 260
53044 Bonn
Germany
strzodka@caesar.de

Marc Droske
Universität Duisburg-Essen
Lotharstr. 65
47048 Duisburg
Germany
droske@math.uni-duisburg.de

Martin Rumpf
Universität Duisburg-Essen
Lotharstr. 65
47048 Duisburg
Germany
rumpf@math.uni-duisburg.de