# EFFICIENT GPU-ACCELERATED ELASTIC IMAGE REGISTRATION

Daniel Ruijters
X-Ray Predevelopment
Philips Medical Systems
Best, the Netherlands
email: danny.ruijters@philips.com

Bart M. ter Haar Romeny
Biomedical Engineering,
Image Analysis and Interpretation
Technische Universiteit Eindhoven
Eindhoven, the Netherlands

Paul Suetens
Medical Image Computing,
ESAT/Radiologie
Katholieke Universiteit Leuven
Leuven, Belgium

**ABSTRACT**
Elastic intra-patient registration can be used to correct for local motion within biomedical images. The application of elastic registration during interventional treatment is seriously hampered by its considerable computation time. The Graphics Processing Units (GPU) can be used to accelerate the calculation of such elastic registrations, without changing the basic registration algorithm. This article discusses how elastic image registration, using cubic B-spline based deformation fields, efficiently can be approached to make use of the vast processing power of the GPU. Our approach employs an efficient GPU-based cubic B-spline deformation field, and also calculates the similarity measure and its derivative on the GPU.

**KEY WORDS**
Medical Image Processing, Elastic Registration, GPU Acceleration

## 1 Introduction

The objective of registration algorithms is to find a spatial mapping between two image datasets. Typically an intensity-based registration algorithm consists of a similarity measure, indicating the quality of a given spatial mapping, and an optimization algorithm, which iteratively searches the optimum (maximum or minimum, depending on the measure) of the similarity measure. The search space consists of the multi-dimensional control variables of the spatial mapping.

The advantage of elastic intra-patient image registration over rigid registration is the fact that it can take local deformation of anatomical structures into account. A cubic B-spline based deformation field is sufficiently smooth to model local elastic displacements of anatomical structures (*e.g.* organs or breast) [1, 2]. However, the application of elastic registration during interventional treatment is still seriously limited by the considerable computation time, which is determined by the very large parameter space of the elastic deformation.

An approach to reduce the computation time, without changing the essential algorithm, is the employment of the vast computation power of modern off-the-shelf Graphical Processing Units (GPU). Though the overall computation power of the GPU nowadays surpasses the power of the CPU, its performance does not scale equally well for any type of algorithm. In the literature there are several publications dealing with GPU-based elastic registration [3, 4, 5], using a piece-wise linear deformation field. We propose a GPU-driven cubic B-spline deformation field, which yields a smoother warping, and therefore can be considered to be a more realistic model for organic deformations.

Further we discuss how the capture range of the elastic registration can be enlarged. It is well known [6] that derivative-based optimizers (*e.g.* quasi-Newton-like optimizers) only evolve to correct solution if the initial position in parameter space is sufficiently close to the optimum. Our approach to elastic registration lends itself very nicely to use derivative information from larger scale-spaces [7]. This allows the optimization process to take information of a larger neighbourhood into account, and therefore is less prone to get stuck in a local optimum.

## 2 Method

### 2.1 Similarity measure

The similarity measure used in intensity based registration algorithms can be expressed as

$$E = E(A, B^\tau) \tag{1}$$

whereby $E$ represents the similarity measure, $A$ the reference image, and $B$ the floating image. $B^\tau$ is the floating image deformed to the coordinate space of the reference image. Let $\vec{i}$ be a position in the reference image space, and the function $\vec{\tau}(\vec{i})$ be the deformation of the reference image coordinate system to the floating image coordinate system. Obviously $B^\tau$ and $B$ are connected as follows: $B^\tau(\vec{i}) = B(\vec{\tau}(\vec{i}))$.

In this article we will restrain ourselves to the class of algorithms, in which the similarity measure can be expressed as a sum of contributions per spatial element (pixel for 2D, voxel for 3D, *etc.*). Sum of Squared Differences (SSD) and Cross-Correlation (CC) are examples of members of this class. This class generally can be written as follows:

$$\begin{aligned} E = \; & \tfrac{1}{\|I\|} \sum_{\vec{i} \in I} e(A(\vec{i}),\, B^{\tau}(\vec{i})) = \\ & \tfrac{1}{\|I\|} \sum_{\vec{i} \in I} e\left(A(\vec{i}),\, B(\vec{\tau}(\vec{i}))\right) \end{aligned} \tag{2}$$

Here $e$ denotes the contribution to the similarity measure per spatial element, and $\vec{i} \in I \subset \mathbb{Z}^N$ represents the set of $N$-dimensional discrete spatial positions (*i.e.* pixel or voxel positions in the image).

The deformation $\vec{\tau}$ is driven by a set of parameters $\vec{c}_j$. It is this set of parameters that is manipulated by the iterative optimization algorithm. In order to obtain a better prediction of parameters used in the next iteration, the Jacobian matrix, containing the partial derivatives of the similarity measure to the parameter space $\delta E/\delta c_{j,m}$ is required [8]. The partial derivative can be decomposed into the following product [2]:

$$\frac{\delta E}{\delta c_{j,m}} = \frac{1}{\|I\|} \sum_{\vec{i} \in I} \frac{\delta e(\vec{i})}{\delta B^{\tau}(\vec{i})} \left. \frac{\delta B(\vec{x})}{\delta x_m} \right|_{\vec{x}=\vec{\tau}(\vec{i})} \frac{\delta \tau_m(\vec{i})}{\delta c_{j,m}} \tag{3}$$

### 2.2 Deformation field

Similar to Kybic and Unser [2], we use a B-spline driven deformation field. The deformation field then can be described by the following equation:

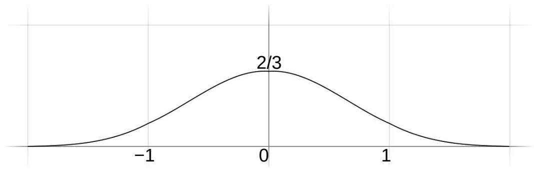
$$\vec{\tau}(\vec{i}) = \vec{i} + \sum_{j \in I_c} \vec{c}_j \cdot \beta_n(\vec{i}/\vec{h} - j) \tag{4}$$

The deformation for position $\vec{i}$ is given by $\vec{\tau}(\vec{i})$. The set of control points $\vec{c}_j$, which drive the deformation, is denoted by $I_c \subset \mathbb{Z}^N$. Vector $\vec{h}$ represents the spacing of the control points, which is required to be integer. Since $\vec{i}$ is added to the sum, the identity deformation corresponds to all control points being zero. $\beta_n(\vec{i})$ is the $N$-dimensional tensor product of an uniform B-spline function, whereby $n$ indicates the degree of the B-spline.

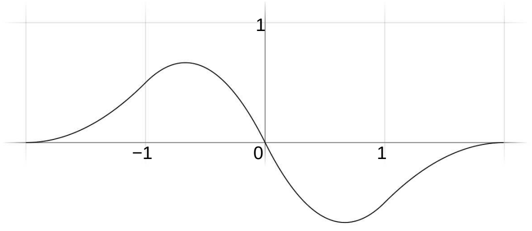
### 2.3 Derivatives

As can be understood from equation 4, the derivative of the deformation field $\delta\tau_m(\vec{i})/\delta c_{j,m}$ simply is a constant term: $\beta_n(\vec{i}/\vec{h} - j)$. Since the control points are evenly spaced, a fixed template of width $n \cdot h$ can be pre-computed to express this derivative. During the calculation of the derivative, the template is then shifted over the image, depending on index $j$.

In contrary to [2], we do not obtain the derivative of the deformed floating image analytically. We rather use an image based approach, employing a convolution with Sobel-like kernels, which approximates the Gaussian derivative. Such a convolution can be very efficiently implemented to run on the GPU.



$$\beta_3(x) = \begin{cases} 0, & |x| \geqq 2 \\ \frac{1}{6} \cdot (2 - |x|)^3, & 1 \leqq |x| < 2 \\ \frac{2}{3} - \frac{1}{2}|x|^2 \cdot (2 - |x|), & |x| < 1 \end{cases}$$



$$\beta_3'(x) = \begin{cases} 0, & |x| \geqq 2 \\ \frac{1}{2}x^2 + 2x + 2, & -2 < x \leqq -1 \\ -\frac{3}{2}x^2 - 2x, & -1 < x \leqq 0 \\ \frac{3}{2}x^2 - 2x, & 0 < x \leqq 1 \\ -\frac{1}{2}x^2 + 2x - 2, & 1 < x < 2 \end{cases}$$

Figure 1. The cubic B-spline, and its 1st order derivative. We use the variant of the B-spline function that is centered around the origin, since this allows us to exploit its symmetry in the GPU programs.

The usage of kernels also allows us to determine the derivative in different scale-spaces, by scaling the B-spline derivative: $\beta_3'(x/m)$. Employing a higher scale-space allows to increase the capture range of the optimization algorithm, since the derivative is based on a wider spatial range [9]. In order to obtain a derivative of the similarity measure that is fully based on a different scale-space, the floating and reference image should be Gaussian blurred. Our first tests show, however, that merely basing $\delta B(\vec{x})/\delta x_m$ on a larger scale-space, by using bigger derivative kernels (see figure 2) already results in an enlarged capture range.

The derivative of the first multiplicand in equation 3 depends on the used similarity measure. In table 1 the derivatives for SSD and CC are given.

## 3 GPU implementation

### 3.1 Two passes

The similarity measure for a given set of transformation parameters, is calculated on the GPU in two passes. These

| Similarity measure | Contribution per pixel | Derivative |
|---|---|---|
| Sum of Squared Differences (SSD) | $e(\vec{i}) = (A(\vec{i}) - B^\tau(\vec{i}))^2$ | $\delta e/\delta B^\tau = 2 \cdot (A(\vec{i}) - B^\tau(\vec{i}))$ |
| Cross-Correlation (CC) | $e(\vec{i}) = A(\vec{i}) \cdot B^\tau(\vec{i})$ | $\delta e/\delta B^\tau = A(\vec{i})$ |

Table 1. Similarity measures, and their derivative with respect to the deformed image.

$$\frac{1}{6\cdot 2} \cdot \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -4 & 0 & 4 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad \frac{4}{81\cdot 9} \cdot \begin{array}{|c|c|c|c|c|}\hline -2 & -6 & 0 & 6 & 2 \\ \hline -15 & -45 & 0 & 45 & 15 \\ \hline -27 & -81 & 0 & 81 & 27 \\ \hline -15 & -45 & 0 & 45 & 15 \\ \hline -2 & -6 & 0 & 6 & 2 \\ \hline \end{array}$$

Figure 2. The $3 \cdot 3$ and $5 \cdot 5$ derivative kernel in x-direction, based on the 1st order derivative of the cubic B-spline (see fig. 1). Larger kernels can be used to obtain derivatives in higher scale-spaces.

two passes also yield the first order derivatives (Jacobian matrix). In the first pass, the floating image is deformed according to the passed parameters. Given this transformed image, in the second pass the contribution of each pixel to the similarity measure $e(\vec{i})$, the derivative of the similarity measure to the deformed floating image $\delta e(\vec{i})/\delta B^\tau(\vec{i})$, and the derivative of the floating image $\delta B(\vec{x})/\delta x_m$ with $\vec{x} = \vec{\tau}(\vec{i})$ are all calculated at once. The derivative of the image space to the deformation control points is constant, and does not need to be computed in every iteration.

### 3.2 Cubic interpolation

The first pass comprises the B-spline driven deformation of the floating image. Sigg and Hadwiger [10] have described how cubic B-spline interpolation can be performed efficiently by the GPU. Their method is based on decomposing the cubic interpolation in $2^N$ weighted linear interpolations, instead of $4^N$ weighted nearest neighbour interpolations. Since linear interpolations are hardwired on the graphics hardware, they can be performed much faster than addressing the set of nearest neighbour lookups, they are composed of.

The basic idea can be understood by considering 1D linear interpolation, which can be expressed as follows:

$$f_{i+\alpha} = (1-\alpha) \cdot f_i + \alpha \cdot f_{i+1} \tag{5}$$

with $i \in \mathbb{N}$ and $\alpha \in [0,1]$. Building on this equation, the weighted addition of two neighbouring samples can be rewritten to be expressed as a weighted linear interpolation:

$$a \cdot f_i + b \cdot f_{i+1} = (a+b) \cdot f_{i+(b/(a+b))} \tag{6}$$

Evaluating the deformation for any given position, using a cubic B-spline, means the weighted addition of $4^N$ adjacent control points, whereby the weights are determined by the cubic B-spline function (see figure 1). In the 1D case this looks like:

$$\begin{aligned} \tilde{f}_{i+\alpha} &= w_0(\alpha) \cdot f_{i-1} + w_1(\alpha) \cdot f_i + \\ &\quad w_2(\alpha) \cdot f_{i+1} + w_3(\alpha) \cdot f_{i+2} \end{aligned} \tag{7}$$

The determination of the weights is further facilitated by the fact that $w_0$ is always located in the first quadrant of the cubic B-spline, $w_1$ always in the second, *etc.* This leads to the following set of weights:

$$\begin{aligned} w_0(\alpha) &= \tfrac{1}{6} \cdot (1-\alpha)^3 \\ w_1(\alpha) &= \tfrac{2}{3} - \tfrac{1}{2}\,\alpha^2 \cdot (2-\alpha) \\ w_2(\alpha) &= \tfrac{2}{3} - \tfrac{1}{2}(1-\alpha)^2 \cdot (2-(1-\alpha)) \\ w_3(\alpha) &= \tfrac{1}{6} \cdot (\alpha)^3 \end{aligned} \tag{8}$$

Using equation 6, we can decompose equation 7 into two weighted linear interpolated lookups.

$$\tilde{f}_{i+\alpha} = g_0 \cdot f_{i+h_0} + g_1 \cdot f_{i+h_1}$$

$$\begin{aligned} g_0 &= w_0 + w_1 \\ g_1 &= w_2 + w_3 \\ h_0 &= (w_1/g_0) - 1 \\ h_1 &= (w_3/g_1) + 1 \end{aligned} \tag{9}$$

Of course this scheme can easily be extrapolated to the $N$-dimensional case, whereby $g_{\vec{j}} = \prod g_{j_k}$, and $\vec{h}_{\vec{j}} = \sum \vec{e}_k \cdot h_{j_k}$, with $k$ denoting the axis and $\vec{e}_k$ the basis vector. In the 3D case this means that 64 nearest neighbour interpolations can be replaced by 8 linear interpolations. On modern GPUs that means a considerable performance gain.

### 3.3 Cubic B-spline deformation

Considering equation 9, it can be observed that the weights $g_0$, $g_1$, $h_0$ and $h_1$ only depend on the B-spline basis function, and not on the pixel/voxel values in the image. This means that they are only dependent on $\alpha$. Since the B-spline, used for deformation, is a function of $\vec{i}/\vec{h}$, with $\vec{h}$ being constant, the weights $g_0$, $g_1$, $h_0$ and $h_1$ depend only on the pixel position, and can be pre-computed for the entire image.

The memory usage can be reduced by considering the fact that the weights are periodic per $\vec{h}$ pixels, and therefore only this segment needs to be computed. Even more can be won by taking advantage of the fact that the B-spline tensor product produces a separable kernel. Therefore, only 1D arrays of width $h_k$ need to be stored, whereby $g_0$, $g_1$, $h_0$ and $h_1$ can be combined in one RGBA lookup table. The difference with the method in [10] is the fact that there is no linear interpolation needed between the entries in the 1D table, since the 1D table is always addressed exactly at an entry index, due to the exact mapping of the table width on the control point interval. The absence of the need to linearly interpolate in the 1D array leads to a higher accuracy, since the fetched parameters $g$ and $h$ are exact (apart from discretization). This means that for $N$-dimensional elastic registration, the cost of the cubic deformation per spatial element (*i.e.* pixel or voxel) is $N$ nearest neighbour lookups in 1D tables, and $2^N$ linear interpolated lookups in the control point texture.

The Cg code [11] below illustrates this process for the 2D case.

```
float2 deformCoordinates(
   float2 coordSource : TEXCOORD0,
   // 1D table with weights in x-direction:
   uniform sampler1D tex_hg_x,
   // 1D table with weights in y-direction:
   uniform sampler1D tex_hg_y,
   // 2D texture with control points:
   uniform sampler2D tex_cp,
   // size of texture tex_cp:
   uniform float2 nrCP,
   // 1 / nrCP:
   uniform float2 rec_nrCP
) : COLOR
{
   // The coord in the 1D table: transform the
   // coordinate from [0,1] to [-0.5, nrCP-0.5]
   float2 coord_hg = coordSource * nrCP - 0.5;

   // lookup the weights in the 1D tables
   float3 hg_x = tex1D(tex_hg_x, coord_hg.x).xyz;
   float3 hg_y = tex1D(tex_hg_y, coord_hg.y).xyz;

   // determine the coordinates for linear
   // interpolation
   float2 coord00 = coordSource;
   float2 coord10 = coordSource;
   coord00.x += hg_x.x * rec_nrCP.x;
   coord10.x += hg_x.y * rec_nrCP.x;

   float2 coord01 = coord00;
   float2 coord11 = coord10;
   coord01.y += hg_y.x * rec_nrCP.y;
   coord11.y += hg_y.y * rec_nrCP.y;

   // fetch the four linear interpolations
   float2 tex_cp00 = tex2D(tex_cp, coord00).xy;
   float2 tex_cp10 = tex2D(tex_cp, coord10).xy;
   float2 tex_cp01 = tex2D(tex_cp, coord01).xy;
   float2 tex_cp11 = tex2D(tex_cp, coord11).xy;

   // weigh along the y-direction
   tex_cp00 = lerp(tex_cp00, tex_cp01, hg_y.z);
   tex_cp10 = lerp(tex_cp10, tex_cp11, hg_y.z);

   // weigh along the x-direction
   tex_cp00 = lerp(tex_cp00, tex_cp10, hg_x.z);

   return coordSource + tex_cp00;
}
```

The outcome of this procedure is a $N$-dimensional offset, which should be added to the current location. This delivers the coordinate in the original floating image. After this coordinate has been obtained the corresponding intensity of the floating image at this location has to be found. This can be done by simple linear interpolation (hardwired on the GPU), or by cubic interpolation as described by Sigg and Hadwiger [10].

## 3.4 Similarity measure & derivatives

In the second pass we bind the deformed floating image and the reference image as textures, and iterate over all spatial elements.

The derivative of the floating image has to be calculated at $\vec{\tau}(\vec{i})$ for all $\vec{i} \in I$, whereby $I$ is the set of reference image grid points. This derivative is approximated by calculating the gradient of the deformed floating image. In order to obtain the gradient image at a given spatial element, its $3^N$ neighbourhood has to be evaluated. This neighbourhood is then multiplied with the derivative kernels for every axis direction. For the multi-scale approach the $3^N$ kernels are replaced by larger kernels of size $(2 \cdot n + 1)^N$ (see figure 2), and the neighbourhood that has to be sampled, should be enlarged correspondingly. The sampling of the neighbourhood, especially for large kernels, can be further optimized using the principle described in equation 6.

After sampling the reference image at the corresponding position, it is rather straightforward to determine the contribution to the similarity measure, and the derivative of the similarity measure with respect to the floating image (see table 1).

Summarizing, the gradient vector is determined for each spatial element, and the contribution to the similarity measure and its derivative is established for the respective spatial element. The results are written to an output texture. For 2D registration only one RGBA output texture is enough, for 3D registration two output textures have to be used. In OpenGL this can be done by using the `GL_EXT_framebuffer_object` and `GL_ARB_draw_buffers` extensions.

The operations that are performed per pixel are described in the following pseudo Cg code [11] for the 2D case:

```
float3 secondPass(
   float2 coord : TEXCOORD0,
   uniform samplerRECT tex_ref,
   uniform samplerRECT tex_float,
   uniform samplerRECT sobel_x,
   uniform samplerRECT sobel_y
) : COLOR
{
   // Sample the 3*3 neighbourhood, and process
   // the samples into the gradient
```

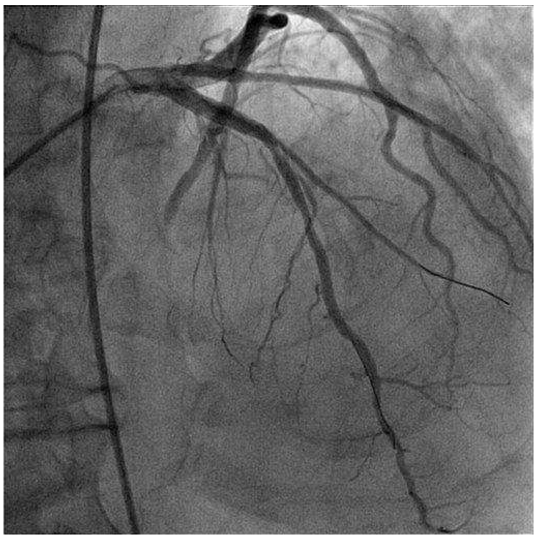


Figure 3. A frame from an angiographic x-ray sequence, showing the left coronary arteries.

```
   float2 gradient;
   gradient.x = sobel_x * neighbourhood;
   gradient.y = sobel_y * neighbourhood;

   // sample the current pixel position
   float ImgFloat = texRECT(tex_float, coord).x;
   float ImgRef = texRECT(tex_ref, coord).x;

   // Similarity measure: SSD
   float difference = ImgRef - ImgFloat;
   float sim_measure = difference * difference;
   float derivative = 2.0f * difference;

   // Pre-multiply the gradient with the
   // derivative of the sim. measure
   gradient *= derivative;

   return float3(gradient.x, gradient.y,
      sim_measure);
}
```

## 4 Results

Using the described approach, it took 402 ms to perform an elastic 2D registration of two images consisting of $1024^2$ pixels, using $8^2$ control points. A single iteration in the optimization process, which consists of deforming the floating image and calculating the similarity measure and Jacobian matrix, took on average 36.9 ms. We used a 2.33 GHz Pentium 4 system, with 2 GB RAM memory, and nVidia QuadroFX 3500 graphics card with 256 MB on board memory. Except for the GPU, we did not use any further optimizations, like using SSE2 instructions or multi-threading.

We used the described algorithm to register all the images in an angiographic x-ray sequence, depicting the coronary arteries. The sequence consisted of 83 images. An example of such an image is shown in figure 3. To visualize the vessels in x-ray images, iodene contrast medium is injected. In the sequence it is visible how the contrast medium enters the vessel tree, and finally washes out. The purpose of the elastic registration is to correct for the motion due to the beating of the heart. As reference image we chose the image with most contrast medium visible. The other frames are then registered to the reference image, starting with the frame that is temporal adjacent to the reference frame. The deformation parameters are always initialized with the parameter set that was found with the registration of the previous frame. Since the movement between adjacent frames is small, the algorithm can converge quicker and more robustly to the optimal deformation.

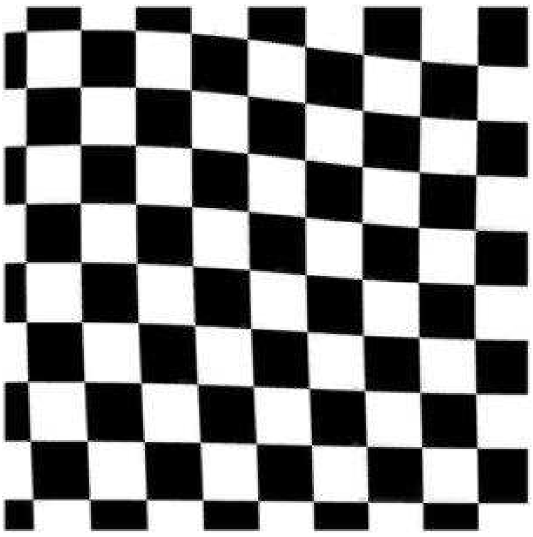


Figure 4. A checkerboard representation of a fragment of the deformation field, used to register a frame from an angiographic x-ray sequence to a reference frame in the same sequence.

The movement of the main bifurcations is corrected very nicely when using $8^2$ control points. The smaller vessels, however, still show quite some rest motion. A multi-resolution control point approach, as described by Schnabel *et al*. [12] could help to reduce the rest motion, without exploding the size of the parameter space.

When inspecting the deformations of the images in the sequence using a checkerboard image, like shown in figure 4, the periodic movement of the heart was very nicely visible. This provides an unintended application of the elastic registration, when it is employed to a cardiac angiographic image sequence; the amplitude of the control point that shows the largest movement, could serve as indication of the phase of the cardiac cycle, *e.g.* when no actual ECG signal is available.

## 5 Conclusions

In this article we have described how intensity based elastic registration algorithms, using a B-spline deformation model, efficiently can be implemented to run on the GPU. We have discussed the various aspects of an efficient and accurate approach to cubic B-spline deformation on the GPU. Further we demonstrated how the similarity measure, as well as its derivative, can be calculated by the GPU, using a two-pass solution. Also we have indicated how a multi-scale approach of the derivative can help to enlarge the capture range, when employing quasi-Newton like optimizers.

In future work we would like to further explore the possibilities of a multi-scale approach to the derivative of the similarity measure. Also we would like to incorporate a multi-resolution approach [12], concerning the control points, in order to obtain finer deformations, where necessary.

## References

[1] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes, "Nonrigid registration using free-form deformations: Application to breast mr images," *IEEE Trans. Medical Imaging*, vol. 18, pp. 712–721, Aug 1999.

[2] J. Kybic and M. Unser, "Fast parametric elastic image registration," *IEEE Trans. Image Processing*, vol. 12, pp. 1427–1442, Nov 2003.

[3] G. Soza, M. Bauer, P. Hastreiter, C. Nimsky, and G. Greiner, "Non-rigid registration with use of hardware-based 3D Bézier functions," in *MICCAI'02*, pp. 549–556, 2002.

[4] R. Strzodka, M. Droske, and M. Rumpf, "Image registration by a regularized gradient flow. a streaming implementation in DX9 graphics hardware," *J. Computing*, vol. 73, pp. 373–389, Nov 2004.

[5] A. Köhn, J. Drexl, F. Ritter, M. König, and H.-O. Peitgen, "GPU accelerated registration in two and three dimensions," in *Proc. Bildverarbeitung für die Medizin*, pp. 261–265, Mar 2006.

[6] D. L. G. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, "Medical image registration," *Phys. Med. Biol.*, vol. 46, pp. R1–R45, 2001.

[7] U. Clarenz, M. Droske, and M. Rumpf, "Towards fast non–rigid registration," in *Inverse Problems, Image Analysis and Medical Imaging, AMS Special Session Interaction of Inverse Problems and Image Analysis*, vol. 313, pp. 67–84, 2002.

[8] S. Kabus, T. Netsch, B. Fischer, and J. Modersitzki, "B-spline registration of 3D images with Levenberg-Marquardt optimization," in *SPIE Medical Imaging'04*, pp. 304–313, 2004.

[9] P. Thévenaz, U. E. Ruttimann, and M. Unser, "Iterative multi-scale registration without landmarks," in *Int. Conf. Image Processing'95*, vol. 3, pp. 228–231, Oct 1995.

[10] C. Sigg and M. Hadwiger, "Fast third-order texture filtering," in *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation* (M. Pharr, ed.), pp. 313–329, 2005.

[11] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard, "Cg: A system for programming graphics hardware in a C-like language," *ACM Trans. Graphics*, vol. 22, no. 3, pp. 896–907, 2003.

[12] J. A. Schnabel, D. Rueckert, M. Quist, J. M. Blackall, A. D. Castellano-Smith, T. Hartkens, G. P. Penney, W. A. Hall, H. Liu, C. L. Truwit, F. A. Gerritsen, D. L. G. Hill, and D. J. Hawkes, "A generic framework for non-rigid registration based on non-uniform multi-level free-form deformations," in *MICCAI'01*, pp. 573–581, Oct 2001.