Abstract

Computational Strategies for Meshfree Nonrigid Registration

Eliezer Gershon Kahn

2006

Biological shapes such as the brain are difficult to register due to their complicated geometry. To deal with this, registration methods often rely on a transformation model consisting of a dense regular grid such as a free form deformation or B-spline grid. However, very dense grids or meshes are usually needed to register images with convoluted shapes, and a regular mesh structure is not well suited for the irregular structure of the brain. What is therefore needed is a meshfree approach such as a radial basis function transformation model. Unfortunately, because radial basis functions are typically non-compact, using them with large numbers of points is fraught with numerical difficulties and, as a result, their use in image registration is not prevalent.

The goal of this work is to overcome these computational difficulties so that radial basis function transformations can be used efficiently, even with large numbers of points. To achieve this, a new registration framework was developed based on automatic differentiation and the fast multipole method. Automatic differentiation is useful since an important component of registration is computing the gradient of the similarity metric which is to be optimized. Automatic differentiation allows one to efficiently calculate gradients without having to write any gradient code explicitly. Although the technique of automatic differentiation is well established, it does not appear to be used for image registration. The fast multipole method was developed to efficiently evaluate large sums

such as radial basis functions but its use in image registration is still minimal. With the integration of these algorithms within a complete registration framework, it should be possible to obtain a truly meshfree registration.

Computational Strategies for Meshfree Nonrigid Registration

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Eliezer Gershon Kahn

Dissertation Director: Lawrence Hamilton Staib

December 2006

Acknowledgments

I would like to express my deepest appreciation to my advisor, Professor Lawrence Staib, for his teaching and guidance throughout my years at Yale. His support, patience, and kindness motivated me to do my best work.

I am grateful to Professors James Duncan and Hemant Tagare for useful discussions and for sitting on my thesis committee.

I would also like to thank Dr. Bill Crum of the University College London for serving as the external reader and for providing help with some of the experiments in this work.

The members of the IPAG lab contributed to an exciting and intellectually stimulating environment. Many thanks to Xenois Papademetris, Marcel Jackowski, Sudha Chelikani, James Beaty, Debayan Datta, Bill Greene, Chrissy Delorenzo, Deepti Bathula, Xaining Qian, Jing Yang, Hirohito Okuda, Gang Liu, Ping Yan, Qian Yang, Yun Zhu, Stathis Hadjidemetriou, Ning Lin, Reshma Munbodh, Zhong Tao, Weichuan Yu, Gary Ho, Yongmei Wang, Oskar Skrinjar, Choukri Mekkaoui, Mario Rodriguez-Bosquez, and Carolyn Meloling.

This work was supported in part by grant EB0311 from the National Institute of

Biomedical Imaging and BioEngineering.

Finally, I would like to thank my wife Idith for all her love and support. Without her help, I could not have finished this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Biological shapes such as the brain are difficult to register due to their complicated geometry. As a result, registration methods often rely on a transformation model consisting of a dense regular grid such as a free form deformation, or a B-spline grid. However, because of the convoluted nature of biological shapes, a regular mesh is not well suited for the irregular structure of the brain. What is desired is a *meshfree* approach so as to entirely bypass the problem of choosing a mesh. This work focuses on developing such a registration system for meshfree registration.

1.1 Radial Basis Functions

Radial basis functions (RBFs) are an alternative to grid based strategies. RBFs are constructed by placing control points at arbitrary locations in space rather than on a regular grid. As a result, they are entirely meshfree. Mathematically, RBFs have the form,

$$f(\vec{x}) = \sum_{i=0}^{N} d_i \phi_i(|\vec{x} - \vec{x}_i|) \tag{1.1}$$

where d_i are scalar coefficients, $\vec{x}_i$ are the locations of the control points, and $\phi(r)$, $r \geq 0$, is known as the *kernel*. Popular kernels include the thin plate spline, $r^2 \log r$, in 2D or r in 3D, and the multiquadrics, $(r^2 + \tau^2)^{k/2}$.

These functions are very smooth and provide very visually pleasing interpolating functions. They also have interesting energy minimization properties. For instance, it can be shown that the 2D thin plate spline minimizes the bending energy:

$$\int\int \left(\frac{\partial^2 f}{\partial x^2}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x\, \partial y}\right)^2 + \left(\frac{\partial^2 f}{\partial y}\right)^2 dx\, dy. \tag{1.2}$$

Although counterintuitive, studies have shown that despite the fact that RBFs do not have compact support, they make excellent interpolants (Franke, 1982). They are infinitely differentiable unlike B-splines or compact support RBFs. Furthermore, having an analytical function is useful in deformation analysis and morphometric studies as various differential operators on the transformation can also be modeled analytically.

Additionally, RBFs are global since the basis functions typically are not compactly supported. Although some might view the global nature of these functions as a disadvantage, it can be argued that global functions are the ideal way to model brain variability, since, biologically, the growth and change of biological structures is, to a certain extent, a global process, not a local one. Hence global functions would be more appropriate. As the main goal of nonrigid registration is to further understand the structure, function, development, and degeneration of the brain—a very important goal—we need to use the best available mathematical functions.

Another advantage of radial basis functions is that they are mesh free since the control points can be placed anywhere in space. Other types of transforms such as free form deformations or B-splines rely on a regular mesh of control points placed in space. This is usually a rectangular grid of points. Other grid schemes such as cylindrical grids are also possible (Coquillart, 1990). Such control grids are suited for only certain classes of problems, namely when the shapes in the image are similar to the shape of the grid. For example, if the image contained box or cube shaped structures, then a rectangular grid would be appropriate. Similarly, if the image contained spherical shaped structures, than a spherical grid would be more suited.

However, in medical imaging, many biological shapes are very complicated and deciding on a grid that best matches the shapes in the images is difficult, if not impossible. For example, although the brain is roughly ellipsoidal, it contains numerous folds that have no specific shape. Hence an ellipsoidal grid, though perhaps better than a rectangular grid, is still not an adequate match for the geometry of the brain. Of course, we can decrease the spacing of the grid to that of the voxel spacing, but then we would have far too many points to deal with leading to numerical instabilities in the transformation. Many control points would also be unnecessary, having been placed in homogeneous regions.

1.2 Computational Difficulties when Using Radial Basis Functions

Unfortunately, although, in theory, RBFs would appear to be the ideal interpolation functions to use, in practice several numerical difficulties must be dealt with if they are to be used efficiently. First, it is necessary to solve a linear system in order to compute the coefficients d_i in Eq. (1.1). Solving such systems directly requires $O(N^3)$ operations which is prohibitive for large numbers of points.

Second, it is necessary to evaluate the transformation function at every voxel in the image. This would require $O(N)$ operations for each voxel. If M is the number of voxels in the image, then the total number of operations is $O(NM)$ which is also extremely large.

Third, computation of the gradient for optimization is also computationally expensive. Finite differences are clearly too expensive for such a task since each gradient component would require the solution of a large linear system as well as summing over the entire image. For example, suppose we use the sum of squared differences as the metric (to be discussed more fully later),

$$f(\vec{q}) = \sum_i (M(u(\vec{q}), v(\vec{q}), w(\vec{q})) - F_i)^2, \tag{1.3}$$

where F is the fixed image, M is the moving image, and $\vec{q}$ is the vector of independent variables. Then, a single component of the gradient is

$$\frac{\partial f}{\partial q_j} = \sum_i 2(M(u(\vec{q}), v(\vec{q}), w(\vec{q})) - F_i)\left(\frac{\partial M}{\partial u}\frac{\partial u}{\partial q_j} + \frac{\partial M}{\partial v}\frac{\partial v}{\partial q_j} + \frac{\partial M}{\partial w}\frac{\partial w}{\partial q_j}\right). \tag{1.4}$$

Here $(\partial M/\partial u, \partial M/\partial v, \partial M/\partial w)$ is the image gradient at (u, v, w) which can be computed easily. The remaining partials, $\partial u/\partial q_i$, $\partial v/\partial q_i$, and $\partial w/\partial q_i$, depend on the type of transformation being used. However, for an RBF transformation such as given in Eq. (4.1) which first requires the solution of a linear system and does not have local support, it is unlikely that an analytical expression that is computationally feasible can be found. To our knowledge, none has been published yet in the literature. The same problem occurs with other metrics such as mutual information.

The remainder of this work presents our proposal to solve these problems by drawing upon two techniques which have not seen widespread use in the medical imaging literature: fast multipole methods (Greengard and Rokhlin, 1987) and automatic differentiation (Griewank, 2000).

1.3 Main Contributions

To overcome these difficulties and achieve a truly meshfree registration, this dissertation presents a new registration framework based on automatic differentiation (AD) and the fast multipole method (FMM).

Automatic differentiation is useful for computing gradients of complicated functions. As we will discuss later, optimization is a major component of registration. In order to optimize a function it is usually necessary to compute gradients. This, however, is often difficult and sometimes even impossible. Furthermore, the need to compute gradients makes it difficult to implement and design new objective functions. Fortunately, the technique of automatic differentiation removes the burden of computing gradients from the programmer to the computer (Griewank, 2000). Unfortunately, it has not been exploited for use in image registration. Our new registration framework uses AD for computing gradients.

The fast multipole method was developed to efficiently evaluate large sums such as (1.1) but their use in image registration is still minimal. The fast multipole method is important because it can be shown that the complexity of evaluating Eq. (1.1) for a single $\vec{x}$ is reduced from $O(N)$ to $O(\log N)$ or even $O(1)$. With the integration of these algorithms within a complete registration framework it is possible to obtain a true meshfree registration.

1.4 Outline of the Dissertation

In the next chapter, chapter 2, we review the literature of medical image registration. We discuss the four components of any registration algorithm as well as applications of registration. Although we are mainly concerned in this dissertation with the nonrigid case, we also describe the rigid case as well.

Next, chapters 3 and 4 contain an overview of automatic differentiation and the fast multipole method. These algorithms have not been of much use in image processing and, therefore, we discuss them in these chapters.

In the next two chapters, chapters 5 and 6, we describe how AD and the FMM can be used in combination for doing intensity based nonrigid registration. In chapter 5 we discuss general issues that arise when integrating AD with image registration and how to overcome them. We describe the use of various similarity metrics such as the sum of squared differences and mutual information and several types of transformations. Chapter 6 continues the discussion of the previous chapter and describes one specific transformation used in registration, namely, radial basis functions such as thin plate splines.

Chapter 7 presents results to validate the methodology of this work and chapter 8 concludes with a summary and future work.

Chapter 2

Review of Image Registration

In this chapter, we provide the relevant background on medical image registration. For a more thorough review, we refer the reader to other review papers and books (Bankman, 2000; Brown, 1992; Crum et al., 2004a; Fitzpatrick et al., 2000; Hajnal et al., 2001; Hill et al., 2001; Lester and Arridge, 1999; Maintz and Viergever, 1998; Maurer and Fitzpatrick, 1993; Pluim et al., 2003; Toga, 1999; Zitová and Flusser, 2003). Although we are mainly concerned in this dissertation with the nonrigid case, for completeness we also describe the rigid case.

2.1 Applications of Registration

Image registration is typically divided into two areas: rigid and nonrigid registration. Rigid registration is a very mature area and is considered by many to be a solved problem (Hajnal et al., 2001). This is no doubt due to the low dimensionality of the problem. As a result, rigid registration algorithms have found their way into commercial products. Nonrigid registration, however, is still an active field of research, and it is to this topic that this dissertation is addressed. This section provides a partial list of some applications where the need for rigid and nonrigid image registration arises in medicine.

2.1.1 Rigid Registration

1. Registration of preoperative images with intra-operative planning.

2. Fusion of images of different modalities (e.g. MRI and CT) so that radiologists can compare them more easily.

3. Registration of time series data. In such situations, images are taken immediately after each other, so no deformation needs to be accounted for.

2.1.2 Nonrigid Registration

There are two broad areas where nonrigid registration is used: intersubject registration and intrasubject registration.

Intersubject Registration

1. Spatial normalization of images for statistical parametric mapping.

2. Brain mapping and the creation of digital anatomic atlases.

Intrasubject Registration

1. Comparing images taken over long periods of time since nonrigid deformations may have taken place.

2. Brain-shift compensation during neurosurgery. When the skull is opened during neurosurgery, the brain shifts slightly. Hence, the images taken prior to the surgery may no longer be valid.

2.2 How is the Registration Problem Solved?

The goal of registration is to find an optimal *transformation* that best matches the moving image with the fixed image. Such a transformation can be described by a set of N parameters. To solve the registration problem, it is typical to design an objective function or *similarity metric* as a function of the transformation parameters. The similarity metric measures how well the moving and fixed images correspond, and such a correspondence is based on comparing various *features* chosen from the images. Our goal is to *search* through the possible candidate transformations until we have found the optimal one. When these transformation parameters are found, our similarity metric is optimized.

More formally, following the work of Brown (1992), any registration algorithm can be understood as consisting of four basic components. This applies to both rigid and nonrigid registration.

1. Feature space: First, we need to choose the features that will drive the registration. Nonrigid registration algorithms have typically been divided into two classes depending on whether they use structural features such as points, curves, or surfaces to drive the registration or simply gray level values. Most algorithms have usually incorporated either structural features or gray level values but not both, but recently there have been efforts to combine both types of information (Collins et al., 1998; Papademetris et al., 2004; Wang and Staib, 2000).

2. Space of transformations: Next, we need to choose the space of transformations. The goal of any registration algorithm is to find a transformation that best maps the moving image to the fixed image. We prefer to somehow limit the class of transformations in which we will search in order to make it feasible to find the best transformation. For instance, in affine registration the space of transformations would be limited to those only consisting of translation, rotation, scale, and shear. Nonrigid transformations transformation such as perspective, polynomial, piecewise polynomial, basis functions, and regularization transformations are more general (Wolberg, 1990). In general, the more parameters we allow in the transformation, the harder the problem will be. Additionally, in some applications the class of transformations used consists of those that are one-to-one (invertible). Methods have been developed for making a transformation one to one (Fujimura and Makarov, 1998; Lee et al., 1996; Tiddeman et al., 2001).

3. Similarity metric: Next, we need to choose the metric used to determine how well the transformed moving image matches the fixed image. Some popular similarity metrics are sum of squared differences (SSD) and mutual information

(MI). Mutual information is particularly good in multi-modal image registration.

4. Search strategy: The final component is the search strategy. Most registration algorithms search for the best transformation by optimizing the similarity metric whose independent variables correspond to the transformation parameters and whose minimum corresponds to the best transformation. Powell's method and the conjugate gradient method are two popular optimization methods that are often used. Other methods do not use optimization but instead formulate a system of partial differential equations where the correct transformation is the solution of the PDE.

2.3 The Four Components in Detail

Once we have selected something for each of these components, we can then use them to solve the registration problem. This section discusses each of the components one by one in more detail.

2.3.1 Feature Space

As mentioned above, features normally chosen are the voxel intensities, or structural features such as points, curves or surfaces. Thus, when registering the brain, points such as the anterior commissure (AC) or posterior commissure (PC) cab be used. Curve features can be sulcal or gyral lines. Surfaces can include the outer cortical surface, the surfaces of the ventricles, and the gray-matter white-matter boundary. It is also possible that instead of considering curves and surfaces one can simply sample them at several points and just consider the points. Thus, points can be said to include both curves and surfaces (Chui, 2001).

Curvature is another feature that can be used. Curvature is a property of curves and surfaces. For curves, the curvature is

$$|\alpha''(s)| \tag{2.1}$$

where α is a curve parameterized by arc length (do Carmo, 1976). For surfaces, there is more than one type of curvature. The two principal curvatures at a point $\vec{p}$ can be found by intersecting the surface with a plane that is normal to the tangent plane at $\vec{p}$. The minimum and maximum, k_1 and k_2 respectively, of the resulting curve for all possible orientations of this normal plane are the two principal curvatures. The mean curvature is then defined as $(k_1 + k_2)/2$ and the Gaussian curvature as $k_1 k_2$. Curvature extrema can be used to identify points (or curves) for registration.

2.3.2 Space of Transformations

Transformations used in registration range from simple rigid or affine transformations with few parameters to complex deformable transforms with thousands or even millions of parameters that can warp the space in quite arbitrary ways. Out of the four components of registration, it is the space of transformations which distinguishes nonrigid registration from affine or rigid registration, since unlike the transformation, the other three components can often be used in both rigid and nonrigid registration. When doing nonrigid registration, one usually performs an affine registration first and then initializes the nonrigid registration with the result of the affine registration.

As mentioned above, the goal of nonrigid registration is to find a transformation that best maps the moving image to the fixed image. Therefore, in 3D we can describe the most general transformation function as the mapping $F : R^3 \rightarrow R^3$ or

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} F_1(x_1, x_2, x_3) \\ F_2(x_1, x_2, x_3) \\ F_3(x_1, x_2, x_3) \end{bmatrix} \tag{2.2}$$

where the point (x_1, x_2, x_3) is mapped to (y_1, y_2, y_3).

Low Dimensional Transformations

The linear transformation can be represented as a simple 3×3 matrix:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} . \tag{2.3}$$

The affine transformation is almost the same as the linear, but with an extra translational term:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} . \tag{2.4}$$

If one is interested in a purely rigid transformation, for instance, when registering two images of the same person at the same time of two different modalities (no deformation), then the transformation would be of the form:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} c_{x_2}c_{x_3} + s_{x_1}s_{x_2}s_{x_3} & c_{x_2}s_{x_3} + s_{x_1}s_{x_2}c_{x_3} & c_{x_1}s_{x_2} \\ -c_{x_1}s_{x_3} & c_{x_1}c_{x_3} & s_{x_1} \\ s_{x_1}c_{x_2}s_{x_3} - s_{x_2}c_{x_3} & -s_{x_1}c_{x_2}c_{x_3} - s_{x_2}s_{x_3} & c_{x_1}c_{x_2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{2.5}$$

where $c_{x_1} = \cos\phi_{x_1}$, $c_{x_2} = \cos\phi_{x_2}$, $c_{x_3} = \cos\phi_{x_3}$, $s_{x_2} = \sin\phi_{x_2}$, $s_{x_3} = \sin\phi_{x_3}$, $s_{x_1} = \sin\phi_{x_1}$, and ϕ_{x_1}, ϕ_{x_2}, and ϕ_{x_3} are the rotation angles around the x_1, x_2, and x_3 axes, respectively.

High Dimensional Transformations

For higher dimensional nonrigid transformations, one popular class of transformation functions uses free form deformations (Coquillart, 1990; Sederberg and Parry, 1986) such as B-splines (Rueckert et al., 1999). This involves placing a rectangular grid of points on the image domain and interpolating, using B-splines, to other points.

Mathematically, this transformation can be expressed as

$$y_1(x_1,x_2,x_3) = x_1 + \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l(r)B_m(s)B_n(t)a_{i+l,j+m,k+n} \tag{2.6}$$

$$y_2(x_1,x_2,x_3) = x_2 + \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l(r)B_m(s)B_n(t)b_{i+l,j+m,k+n} \tag{2.7}$$

$$y_3(x_1,x_2,x_3) = x_3 + \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l(r)B_m(s)B_n(t)c_{i+l,j+m,k+n} \tag{2.8}$$

where the B-spline basis functions are

$$B_0(u) = (1-u)^3/6 \tag{2.9}$$

$$B_1(u) = (3u^3 - 6u^2 + 4)/6 \tag{2.10}$$

$$B_2(u) = (-3u^3 + 3u^2 + 3u + 1)/6 \tag{2.11}$$

$$B_3(u) = u^3/6 \tag{2.12}$$

and $i = \lfloor x_1 \rfloor - 1$, $j = \lfloor x_2 \rfloor - 1$, $k = \lfloor x_3 \rfloor - 1$, $r = x_1 - \lfloor x_1 \rfloor$, $s = x_2 - \lfloor x_2 \rfloor$, $t = x_3 - \lfloor x_3 \rfloor$, and $a_{i+l,j+m,k+n}$, $b_{i+l,j+m,k+n}$, and $c_{i+l,j+m,k+n}$ are the displacements of the control points.

Another type of transformation uses radial basis functions (RBF) such as thin plate splines (Bookstein, 1989) or multiquadrics (Hardy, 1990). Mathematically, these can be expressed as

$$y_1(x_1,x_2,x_3) = x_1 + \sum_{i=1}^{N} a_i\phi(|(x_1,x_2,x_3)-(x_{1i},x_{2i},x_{3i})|) + \sum_{i=1}^{m} u_i p_i(x_1,x_2,x_3) \tag{2.13}$$

$$y_2(x_1,x_2,x_3) = x_2 + \sum_{i=1}^{N} b_i\phi(|(x_1,x_2,x_3)-(x_{1i},x_{2i},x_{3i})|) + \sum_{i=1}^{m} v_i p_i(x_1,x_2,x_3) \tag{2.14}$$

$$y_3(x_1,x_2,x_3) = x_3 + \sum_{i=1}^{N} c_i\phi(|(x_1,x_2,x_3)-(x_{1i},x_{2i},x_{3i})|) + \sum_{i=1}^{m} w_i p_i(x_1,x_2,x_3) \tag{2.15}$$

where a_i, b_i, c_i, u_i, v_i, and w_i are coefficients which are computed by solving three linear systems, the third term on the right hand side of each of these three equations are optional low order polynomial, and $\phi(r)$, $r \geq 0$, is a basis kernel which can have several forms. Popular kernels include the 2D thin plate spline, $r^2 \log r$, the 3D thin plate spline, r, and the multiquadrics, $(r^2+\tau^2)^{k/2}$, where τ is any real number and k is an odd integer. Such radial basis function transformations are often preferred because they are very smooth, and experiments have shown them to be better than most other interpolants (Franke, 1982). Unfortunately, when the number of points is large, computing the coefficients of the basis functions becomes costly due to the fact that in order to find the coefficients it is necessary to solve an $N \times N$ linear system where N is the number of points. Solving such systems using a direct linear solver requires $O(N^3)$ operations. Clearly, it is desirable to use as many points as possible since the more points used to define the mapping, the more flexible the transformation allowing finer accuracy in the registration. This problem has led many to abandon radial basis functions in favor of

B-splines. One main goal of this work is to show that we can, in fact, develop fast algorithms even for large number of points. We will return to this later.

How do we Actually Warp an Image?

Supposing we have a transformation that we would like to apply to an input image, how do we actually go ahead and warp the image? This is a very tricky issue which is often not addressed in the literature.

In general when we try to warp an image we have two options: forward mapping or inverse mapping (Wolberg, 1990). In the forward mapping approach, our mapping function is such that given any point $\vec{P}$ in the input image, an output point $\vec{Q}$ will be computed. The problem with this approach, however, is that the points from the input image may get mapped to points that are in between pixels in the output image. This results in a complicated scattered data interpolation problem which is non-trivial.

Inverse mapping solves this problem. In inverse mapping, our mapping function is such that given a point $\vec{Q}$ in the output image, the point $\vec{P}$ in the input image which was mapped to $\vec{Q}$ will be computed. Therefore, to compute the pixel value at point $\vec{Q}$ in the output image we need to interpolate the input image at point $\vec{P}$. This is easier since this time we are interpolating a regular grid, not scattered points.

If we only have a forward mapping function, we can still apply inverse mapping to warp the image by inverting the transformation. This can be done by solving the system of nonlinear equations, Eq. (2.2), for x_1, x_2, x_3 using, for example, Newton's method. Such a method needs to be performed for each pixel in the output image. Convergence to a solution is possible if and only if the Jacobian of the transformation (2.2) is nonzero.

In the context of image registration we, therefore, have two options for warping. One option is to optimize for the best forward mapping and then use Newton's method to solve for the inverse mapping and warp the moving image to the fixed image. Unfortunately using Newton's method is expensive, even if it converges quickly. The second option is to directly optimize for the inverse transform so that we do not need to use Newton's method since we already have the inverse. Obviously, the second approach is better. Thus, when we say that the goal of image registration is to "find a transformation that best maps the moving image to the fixed image," what we really mean is "find *the inverse of* a transformation that best maps the moving image to the fixed image."

Combining Transformations: Serial vs. Additive

When one does an affine registration prior to the nonrigid one, how do we actually integrate it into the nonrigid registration? One simple way to combine two transformations would be to resample the moving image and treat this resampled image as the new moving image. Hence we do not directly deal with the affine transformation when doing the nonrigid registration since the affine transformation is accounted for when performing the resampling. The problem with this method, of course, is that the process of resampling may introduce undesirable approximations and aliasing artifacts in the resampled image. There remain two other ways which we consider: serial and additive. These two approaches are applicable whenever combining more than one transformation.

The serial way, is as follows. Suppose the first transformation is of the form $\vec{y} = g(\vec{x})$ and the second is of the form $\vec{z} = f(\vec{y})$ where $\vec{x}$, $\vec{y}$, and $\vec{z}$ are points in R^3, then the combined transformation, F, is

$$\vec{z} = F(\vec{x}) = f(g(\vec{x})). \tag{2.16}$$

The additive way is as follows. Suppose the two transformations are again f and g and we wish to combine them into a single transformation, F. Then this combined transformation is

$$\vec{z} = F(\vec{x}) = \vec{x} + (f(\vec{x}) - \vec{x}) + (g(\vec{x}) - \vec{x}). \tag{2.17}$$

Note that this time both f and g are evaluated at the same point $\vec{x}$ (rather than f being evaluated at $\vec{y}$ and g at $\vec{x}$). In other words, we view both f and g as independent displacements which are added to $\vec{x}$ to produce the transformed point.

The serial way is probably what most people intuitively have in mind when we talk about combining transformations: We apply the first transformation to point $\vec{P}$ to get $\vec{P'}$ and then apply the second transformation to $\vec{P'}$ to get $\vec{Q}$. The additive way can be slightly confusing since both transformations are applied to the same input point $\vec{P}$. However, the additive way has the advantage of being more efficient to compute since only addition is involved whereas the serial way usually requires a (matrix) multiplication.

2.3.3 Similarity Metric

Perhaps the two most popular similarity metrics used for registration are sum of squared differences (SSD) and mutual information (MI).

SSD

The sum of square differences (SSD) in 3D is (Fitzpatrick et al., 2000)

$$f(\vec{q}) = \sum_i (M(u(\vec{q}), v(\vec{q}), w(\vec{q})) - F_i)^2 \tag{2.18}$$

where F is the fixed image, M is the moving image, and $\vec{q}$ is the vector of independent variables. Notice that each of three components of the transformation depends on the independent variables $\vec{q}$.

Mutual Information

One of the most important developments in the 1990's was the application of information theory to image registration (Collignon et al., 1995; Viola and Wells, 1995). One result was a similarity metric known as the mutual information (MI) which can be expressed as (Mattes et al., 2003)

$$f(\vec{q}) = \sum_l \sum_k p(l, k; \vec{q}) \log(\frac{p(l, k; \vec{q})}{p_M(l; \vec{q}) p_F(k)}) \tag{2.19}$$

where $\vec{q}$ is the vector of independent variables, $p(l,k;\vec{q})$ is the joint histogram of the fixed image and the transformed moving image, $p_M(l;\vec{q})$ is the histogram of transformed moving image, and $p_F(k)$ is the histogram of the fixed image. The double sum is over all bins in the histograms which are indexed by l and k.

2.3.4 Search strategy

Finding the transformation is usually accomplished by optimizing the similarity metric over the transformation parameters. The values of the independent variables at the minimum correspond to the desired transformation. Common types of optimization strategies used are non-gradient and gradient based methods, global methods such as simulated annealing and genetic algorithms, and PDE based methods. For gradient based method, the question is, of course, how do we compute the gradient? We return to this in the next chapter. In the remainder of this chapter, we discuss each of these methods in greater detail.

Non-gradient Based Method

Two popular non-gradient based methods for multivariable optimization are the simplex method and Powell's method Press et al. (1992). Such methods are not considered in this work due to their poor convergence properties for large size problems.

Steepest Descent Method

The simplest type of gradient descent method is to simply move in the direction of the gradient until we cannot decrease the function any longer. This is known as steepest descent. Unfortunately, this algorithm does not converge very quickly and often gets slowed down due to zigzagging through the solution space which results from the fact that we end up revisiting the same directions over and over again.

Conjugate Gradient Method

The conjugate gradient method overcomes the problems of the steepest descent method. In the conjugate gradient, at each iteration the next direction to move in is computed based on all the previous directions. This prevents us from revisiting a direction already used. The conjugate gradient method has excellent convergence properties and, as a result, we use it often in this work.

Newton and Quasi Newton Based Methods

Newton based methods compute the Hessian of the objective function and thereby create a quadratic approximation to the objective function. Quadratic functions can be easily minimized by solving a linear system.

One problem with Newton based methods is that computing the full Hessian is very expensive. Therefore, quasi-Newton methods have been developed which approximate the Hessian such as the Limited Broyden Fletcher Glajl Shjofd (L-BFGS) method (Liu and Nocedal, 1989). An extension to this method is known as L-BFGS-B which adds simple bound constraints to the objective function (Byrd et al., 1995; Zhu et al., 1997).

Simulated and Deterministic Annealing

The optimization methods discussed so far are all local and hence have the problem of being trapped in local minima. Clearly it is desirable to find the global minimum of the similarity metric. Simulated annealing is such a method. Unlike gradient based methods where we only go in the downhill direction, with simulated annealing, we sometimes allow moves in the uphill direction. Hopefully, this will enable us to get out of local minima. Simulated annealing is modeled on the natural process of annealing where a system which is in a high energy state is gradually lowered to a low energy state. An important parameter is the temperature which is gradually lowered at each iteration. When the temperature parameter is high, more random direction movements are possible and there is a higher probability of getting out of local minima. But as the temperature gets lower, such movements become more unlikely.

Deterministic annealing is similar to simulated annealing except that a deterministic rule is used to determine whether or not we make a move, rather than a random one.

Genetic Algorithms

Genetic algorithms model the evolutionary process of natural selection. As in evolution, genetic algorithms involve *chromosomes*, *mutations*, and *crossovers* (or *recombination*). We start out with a random population of chromosomes. Each chromosome is a string of bits which corresponds to the bits of the independent variables. The goal is to find the chromosome which best optimizes the objective. Each chromosome is assigned a *fitness* value depending on how well it solves the problem. Then, pairs of genes are repeatedly selected from this population and crossover and mutations occur until a new population or generation is obtained. If this new population contains a solution, we stop. Otherwise the entire process is repeated until the next generation is created.

PDE Based Methods

The well-known Euler-Lagrange theorem establishes an important relationship between optimization and partial differential equations. As a result there is an entire class of algorithms that attempt to solve the registration problem by converting the objective function to a PDE (Bajcsy and Kovacic, 1989; Christensen, 1994; Christensen et al., 1996, 1993). Included in these methods are those that view the process of registration as a fluid deformation and borrow concepts from fluid dynamics.

Multiresolution Strategies

One popular method to help avoid being trapped in local minima is a multiresolution approach. Such methods can be applied to many types of problems in image analysis other than registration and are based on the idea of first trying to solve the problem at a lower resolution and then using this answer to initialize the algorithm at a higher resolution. Thus, to implement such a strategy, we would create a hierarchy of images where each image higher up in the hierarchy is at a lower resolution than the one below it. Then the algorithm is first applied to the coarsest image in the hierarchy. The solution obtained at this level is used to initialize the next level of the hierarchy. This process is repeated until the image with the highest resolution in the hierarchy is reached.

Chapter 3

Automatic Differentiation

3.1 Introduction

As mentioned in the previous section, a major component of registration is the optimization strategy used. Many optimization algorithms are iterative methods based on the concept of gradient descent (Press et al., 1992). In such algorithms, we start out with an initial estimate of the solution and then use the gradient of the objective function to compute the next candidate solution. We repeat this until convergence. However, often the objective function to be optimized is extremely complicated and does not have a gradient that is expressible in analytic form. For example, if the objective function involves complicated algorithms such as solutions of differential equations, solutions of linear systems, fast Fourier transforms, numerical quadratures and the like, then there may be no analytical gradient. In such cases, the only available option for computing such derivatives is finite differencing. However, if there are n independent variables then computing such finite differences requires at least n function evaluations. For many functions, this computation is totally unrealistic. This is especially true in image registration where each function evaluation typically requires summing over an entire image.

To solve this problem, a technique known as automatic differentiation (AD) has been developed (Griewank, 2000; Griewank et al., 1996) which has seen important applications in many numerical problems especially in relation to the solution of partial differential equations (Hart et al., 2005; Kim et al., 2006; Lea et al., 2002; Scholze et al., 2002; Shapiro and Tsukanov, 1999; Tsukanov and Shapiro, 2002; Wang et al., 1995). Although registration is primarily an optimization problem, AD does not appear in the registration literature. Automatic differentiation is based on the principle that any computer algorithm, no matter how complicated, is, nevertheless, just a series of simple computations such as addition or multiplication whose derivatives are easy to compute. Since any function can be viewed as the composition of many smaller elementary functions, then, in theory, using the chain rule, it should be straightforward to compute the derivative of almost any scalar function, no matter how complicated. Is such a scheme worthwhile? Yes! A result known as the “cheap gradient rule” says that the time to compute the gradient of a scalar function of n variables (where n can be very large) is only four or five times the time to compute the original function itself (Griewank, 2000). In this chapter we review the basic ideas of AD which are necessary for image registration.

3.2 Matrix Approach

There are at least two approaches we can use to explain the theory of automatic differentiation: a matrix approach and a graph theoretic approach. In this section we present the matrix approach and in the next we present the graph theoretic approach. Suppose we are given the scalar function

$$z = F(\vec{x}), \qquad \vec{x} \in \mathbf{R}^n,\ z \in \mathbf{R},\ F : \mathbf{R}^n \to \mathbf{R}, \tag{3.1}$$

and we would like to compute its gradient. Since, as already mentioned, any mathematical function evaluated on a computer is really a composition of elementary functions, Eq. (3.1) can be expressed as the sequence (following the notation similar to the work by Griewank (1992))

$$g(\vec{x}) \to \vec{s}_0, \quad f_i(\vec{s}_i) \to \vec{s}_{i+1}, \quad h(\vec{s}_m) \to z, \qquad i = 0, \ldots, m-1 \tag{3.2}$$

In this formula, $\vec{s}_i$ represents the set of all variables which includes the independent variables, any variables that depend on them, as well as the dependent variable. Let q be the number of elements in this set. Each f_i is an elementary function that maps the set $\vec{s}_i$ to itself, g is the very first operation which maps the independent variables to the set of all variables, and h is the final elementary operation that maps the set of all variables to a single scalar z. An equivalent way to express this is

$$z = F(\vec{x}) = h(f_{m-1}(\cdots(f_2(f_1(f_0(g(\vec{x}))))) \cdots)). \tag{3.3}$$

Differentiating this formula using the chain rule we have

$$F'(\vec{x}) = g'(\vec{x}) f_0'(\vec{s}_0) f_1'(\vec{s}_1) f_2'(\vec{s}_2) \cdots f_{m-1}'(\vec{s}_{m-1}) h'(\vec{s}_m). \tag{3.4}$$

Here $F'(\vec{x})$, which is the gradient of $F(\vec{x})$, is an $n \times 1$ column vector. Each f_i' is a $q \times q$ Jacobian matrix of the elementary transformations f_i. g' is the $n \times q$ Jacobian matrix of g, and h', an $n \times 1$ column vector, is the gradient of h. Since the final factor h' is a column vector while all the other factors are square or rectangular matrices, and since matrix multiplication is associative, it follows that for better performance, the above product should be computed from right-to-left rather than from left-to-right. This way we have a sequence of matrix-vector products rather than a sequence of matrix-matrix products. However, doing a right-to-left product requires that we somehow keep all the previous elementary operations in memory and then apply the chain rule "in reverse". The leads us to what is known as the "reverse mode" of automatic differentiation: we evaluate the function $F(\vec{x})$ while simultaneously recording every single operation in memory. Such a recording is called a *trace* of the function. Then, we apply the chain rule in reverse to compute the gradient. If we were to compute the product from left-to-right, this would be the "forward mode" of automatic differentiation which is only efficient when the number of dependent variables is greater than the number of independent variables. Since in registration the similarity metrics we encounter are usually scalar, the forward mode will not be needed.

3.3 Directed Acyclic Graph Approach

The previous matrix based approach was used to present automatic differentiation in an abstract manner. It allows one to clearly see the distinction between the forward and reverse mode, and will be useful later on when explaining the checkpointing algorithm. It is not, however, how AD would actually be implemented on a digital computer. Therefore, an equivalent approach which is more amenable to actual implementation is based on graph theory. The type of graph relevant to AD is known as a directed acyclic graph (DAG). Recall that a graph is a set of nodes and arcs. Each arc connects two nodes. By *directed* we mean that each arc has associated with it a direction which can be represented as an "arrow". It is *acyclic*, in that it is impossible to construct a path which starts from a given node and ends at that same node while passing through at least one other node.

3.3.1 Example

We give a simple example to demonstrate the use of directed acyclic graphs and the reverse mode of AD similar to an example given by Griewank (2000, Chap. 1). We will use automatic differentiation to compute the gradient of the scalar function

$$z = x \sin(y \exp(x)) \tag{3.5}$$

for $x = 1.2$ and $y = 0.7$. The diagram in figure 3.1 shows this function as a DAG. If

$$\begin{tikzcd}
x \arrow[r] & v_{-1} \arrow[dr] \arrow[ddrrrr] & & & & & \\
 & & v_1 \arrow[dr] & & & & \\
y \arrow[r] & v_0 \arrow[rr] & & v_2 \arrow[r] & v_3 \arrow[r] & v_4 \arrow[r] & z
\end{tikzcd}$$

Figure 3.1: Directed acyclic graph for Eq. (3.5).

this function is expanded out, we have

$$\begin{aligned} v_{-1} &= x \\ v_0 &= y \\ v_1 &= \exp(v_{-1}) \\ v_2 &= v_1 * v_0 \\ v_3 &= \sin(v_2) \\ v_4 &= v_{-1} * v_3 \\ z &= v_4. \end{aligned} \tag{3.6}$$

Each of the v_i is a node in the DAG and the edges correspond to elementary operations. The reverse mode of automatic differentiation is applied to this function for $x = 1.2$ and

$$\begin{aligned}
v_{-1} &= x = 1.2\\
v_0 &= y = 0.7\\
v_1 &= \exp(v_{-1}) = 3.3201\\
v_2 &= v_1 * v_0 = 2.3241\\
v_3 &= \sin(v_2) = 0.72945\\
v_4 &= v_{-1} * v_3 = 0.87533\\
z &= v_4 = 0.87533\\
\bar{v}_4 &= \bar{z} = 1.0\\
\bar{v}_3 &= \bar{v}_4 * v_{-1} = 1.2\\
\bar{v}_{-1} &= \bar{v}_4 * v_3 = 0.72945\\
\bar{v}_2 &= \bar{v}_3 * \cos(v_2) = -0.82086\\
\bar{v}_1 &= \bar{v}_2 * v_0 = -0.57460\\
\bar{v}_0 &= \bar{v}_2 * v_1 = -2.7253\\
\bar{v}_{-1} &= \bar{v}_{-1} + \bar{v}_1 * \exp(v_{-1}) = -1.1783\\
\bar{y} &= \bar{v}_0 = -2.7253\\
\bar{x} &= \bar{v}_{-1} = -1.1783
\end{aligned}$$

Figure 3.2: Example of reverse mode of automatic differentiation applied to Eq. (3.5).

$y = 0.7$ as shown in figure 3.2. Differentiating analytically, we get

$$\begin{aligned}
\frac{\partial z}{\partial x} &= x\cos(y\exp(x))y\exp(x) + \sin(y\exp(x))\\
\frac{\partial z}{\partial y} &= x\cos(y\exp(x))\exp(x)
\end{aligned} \tag{3.7}$$

Evaluating at $x = 1.2$, $y = 0.7$, we have $\frac{\partial z(1.2,0.7)}{\partial x} = -1.1783$, $\frac{\partial z(1.2,0.7)}{\partial y} = -2.7253$ as expected.

3.4 “The Cheap Gradient Rule”

We mentioned earlier that according to the “cheap gradient rule” it is possible to compute gradients very quickly such that the time to compute the gradient is only a small multiple of the time to compute the function. This is in contrast to finite differences where the multiple is very large, typically at least $O(N)$ for N independent variables. The “cheap gradient rule” can be explained intuitively using the following simplified example. Suppose the time to compute the function alone is 1 second. Suppose further that the function consists of 1000000 binary operations of the form $c = a@b$ where $@$ is a binary operator such as multiplication or addition (the variables a, b, and c will of course usually be different in each operation). Thus, each operation consists of 2 memory reads for variables a and b, one binary instruction $@$ (which probably involves 3 registers in the CPU), and 1 memory write for variable c, for a total of 4 instructions. To simplify things, assume that all instructions take the same amount of time. Thus, there are a total of 4000000 instructions in the entire 1 second function evaluation.

Now computing gradients is a two step process: a forward sweep with tracing on followed by a reverse sweep. Let’s therefore first discuss the time to do the forward sweep with tracing. At least 4000000 instructions are needed to compute the function. In addition, we must *record* all these instructions so that we can process the function in reverse. How do we do this? Well, we need to record the fact that variables a, b, c were involved. In other words, we need to record the 3 *memory locations* involved in the binary operation. This takes 3 writes. In addition we need to record the specific *values* of a, b, and c. This requires another 3 writes. In addition we need to record which binary operation @ was performed, hence requiring another write. Thus to record each elementary operation requires at least 7 more instructions per operation for a total of 11 instructions per operation. Thus a total of 11000000 instructions are needed for the entire forward sweep.

Let's now discuss the reverse sweep. In the reverse sweep we take the derivative of each of the 1000000 elementary operations done in the forward sweep. Thus the reverse sweep can be viewed as a sequence of 1000000 elementary derivative operations. For simplicity assume that each of these elementary derivative operations also takes 4 instructions. Thus we need at least 4000000 instructions for the reverse sweep. In addition, there is overhead due to the fact that we must read all the recorded instructions from memory. Let's assume that this costs another 7 instructions per operation. Then the total number of operations for the reverse sweep is also about 11000000 instructions.

Thus the total number of instructions is about 22000000 which is 5.5 times the original function evaluation. Although this example is overly simplified and there is more overhead as well as speed-ups which we have not discussed, the actual implementation will be very similar to the way we just described.

3.5 Branches in the Objective Function

Once one has the entire function trace in memory, it is actually possible to reuse it for different values of the independent variables without having to go through the recording a second time. However, this is only possible if the function does not contain any "branches". By a branch we mean the familiar *if* or *switch* statements that are common in programming languages. For instance, we may have an *if* statement

```
**if** $x > 0$ **then**
    do something
**else**
    do something else
**end if**
```

where x depends on the independent variables. Clearly, this *if* statement may not be differentiable when x equals zero. Therefore, when tracing the function, if x happens to be positive, only the first part of the *if* statement will be recorded in memory but not the second. In other words the trace of the function will only be valid when $x > 0$. Thus if we attempt to reuse the trace with a different set of independent variables, x may turn out to be negative and the trace stored in memory will no longer be valid. It will therefore be necessary to redo the entire trace all over again.

As we will see later, all functions we deal with in image registration have these branches, so it will usually never be possible to reuse the trace for a different set of independent variables.

3.6 Checkpointing

One problem with the reverse mode of automatic differentiation is that the trace of the entire function must be recorded during the forward sweep. For large functions, however, this can quickly consume all the memory in the computer. This is certainly the case in image registration which has complicated objective functions. One solution to this problem is known as the *checkpointing* strategy (Griewank, 1992; Griewank and Walther, 2000).

To understand the checkpointing algorithm, we must view the sequence of elementary operations of the objective function as a type of discrete-time process in which during each time interval, the “state” of system changes or is transformed to another “state”. Using the notation above (section 3.2), a state of the system is fully specified by the values of all its variables, $\vec{s}_i$, and one can view each elementary mapping $f_i(\vec{s}_i) \rightarrow \vec{s}_{i+1}$ as a single time interval or time step. (Actually, treating each individual elementary operation as a single time step would be overkill for the checkpointing algorithm. It is better to treat a consecutive sequence of, say, M elementary operations as a single time step where M can be many thousands of operations.)

It now follows that if we know the state of the system at time t, then we can simply jump to time t by loading each variable with the appropriate value. The computation of the remainder of the function can then continue in exactly the same way as if it had been started from the very beginning.

With this new view of the objective function, the reverse mode can be improved by noting that to reverse the function it is not necessary to have the *entire* function trace stored in memory. Instead, it suffices to have only the part of it which is currently being differentiated. This can be accomplished be saving the state of the system at several points in time during the forward sweep. These special saved states are called *snapshots*

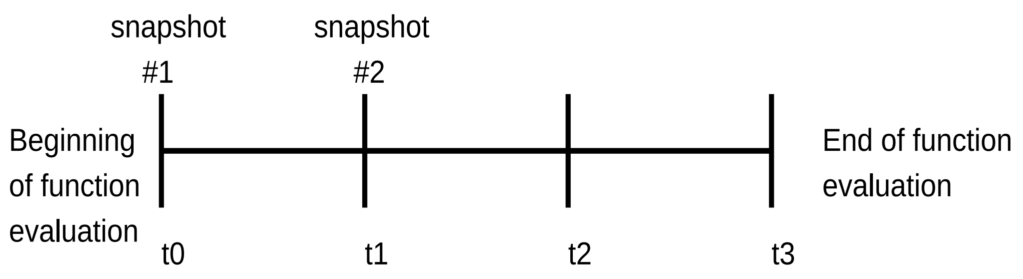

Figure 3.3: Checkpointing example.

or *checkpoints*.

By saving snapshots of the state of the system, we can then jump to any point in time of the function evaluation for which we saved a state of the system and continue the forward computation of the function from there. With this ability to easily move to any point in time, it is possible to perform the reverse sweep in pieces, one time step at a time.

An example will help clarify how this can be done. Suppose we have a function which can be divided into 3 time intervals as shown in figure 3.3. One possible way to compute the derivative is as follows. We take a snapshot at t_0 and then begin the function evaluation at t_0 without tracing. As we proceed, we take another snapshot at t_1. When we reach t_2 (note we do not take a snapshot at t_2), we turn on tracing and complete the function evaluation thus arriving at t_3. Then we begin the reverse sweep starting from t_3 going in reverse and stopping when we reach t_2. Then we reload the snapshot from t_1, evaluate the function with *tracing turned on* from t_1 until t_2, and continue computing the derivative in reverse from where we left off, namely from t_2 to t_1. We repeat this step one more time: reload the snapshot from t_0, evaluate the function with tracing turned on from t_0 to t_1, and compute the derivative in reverse from where we left off, namely from t_1 to t_0.

The functions we deal with in image registration, however, can have thousands of time steps, and storing a snapshot for every single time step simultaneously is impractical. Therefore, it is necessary to set a limit to the number of checkpoints stored at any give moment in time and decide on a suitable checkpointing schedule. Differences in the checkpointing schedule can dramatically effect the overall computational requirements, so it is important to choose the best schedule. This is essentially a combinatorial optimization problem. Fortunately, Griewank (1992) has derived an optimal schedule for the reverse mode of automatic differentiation.

3.7 Implementation

3.7.1 Implementing Automatic Differentiation

The discussion until now presented the mathematics of AD. We now discuss briefly how to actually implement it on a computer. There are several known methods used to implement AD. We discuss here operator overloading and source transformations.

Operator Overloading

The technique used in the ADOLC package (Griewank et al., 1996), which we used in our experiments, is based on operator and function overloading, which is a language feature of some programming languages such as C++. Operator overloading means that the standard arithmetic operations (such as +, -, *, and /) can be redefined for different datatypes provided that they remain binary (or unary in the case of unary operators) functions. Function overloading is the same idea applied to standard functions such as the square root or power function. To overload the operators and functions, we define a new special datatype that behaves just like the standard floating point datatypes[1]. All our overloaded operators and functions will act on this special datatype rather than on the usual floating point datatypes. These new operators and functions will redefine the standard functions of the same name to both (a) do the actual calculation and (b) record a trace of the computation so we can reverse it later. The advantage of overloading is that we can write the code in exactly the same way as if we were doing normal arithmetic. The tracing that takes place is completely transparent to the coder.

Source Transformation

Another way to implement AD is known as source transformation. In this way, the source code of the objective function is processed by a program which then generates code to compute the derivative. This is how AD was implemented in the ADIFOR package (Bischof et al., 1992, 1994).

[1]In ADOLC this datatype is called *adouble*

3.7.2 Implementing Checkpointing

Fork and Join Approach

There are various ways to implement the checkpointing algorithm. One way, discussed by Griewank (1992), uses UNIX's fork and pipe commands (Stevens, 1993). This approach exploits the fact that checkpointing is an inherent part of modern multitasking operating systems: for a serial processor to give the illusion of doing many things as once requires that the state of a process be saved. Therefore, to use the operating system to help us in this method, when a new checkpoint is created, a new process is forked off. The parent process then blocks until the child process returns. Since forks are implemented using copy-on-write (in more modern systems), forking off another process does not copy all the data from the parent process (which is crucial when dealing with large images). Communication between processes is achieved with pipes (or shared memory regions, as was done by Mauer-Oats (1997)).

Goto Approach

Implementing the fork and pipe approach requires experience with UNIX systems programming and may not be portable to all architectures. In order to avoid detailed UNIX systems programming, the entire algorithm can be coded in a single address space without any forks and pipes. To achieve this, we developed a procedure using *goto*s. *Goto* statements are used in order to jump to an arbitrary time step. This makes moving data around easier at the expense of more complicated code. The fork and pipe approach is more generic, however, and also facilitates parallelization, making it a useful possible future improvement.

Chapter 4

Radial Basis Functions and the Fast Multipole Method

4.1 Fast Evaluation of Radial Basis Function Sums

As mentioned in chapter 2, computing the sum

$$f(\vec{x}) = \sum_{i=1}^{N} d_i \phi_i(|\vec{x} - \vec{x}_i|) \tag{4.1}$$

is expensive for large numbers of points, especially if we need to evaluate it over an image grid. The fast multipole method (FMM) (Greengard, 1987, 1988; Greengard and Rokhlin, 1987, 1988) was developed to deal with this problem. Although it was originally developed to solve the N-body problem in potential theory, it has been extended to the interpolation problem which is our primary concern here (Beatson and

Greengard, 1997; Beatson et al., 2001a; Beatson and Light, 1997; Beatson and Newsam, 1998; Cherrie et al., 2002). The main result is that the cost of computing an approximate sum of the form (4.1) at all points $\vec{x}_i$ can be reduced from $O(N^2)$ to $O(N \log N)$ or even $O(N)$. Because of its importance, the FMM has been ranked as one of the top ten algorithms of the twentieth century (along with the fast Fourier transform) (Dongarra and Sullivan, 2000).

The fast multipole method is based on the principle that points that are far away (in the far field) do not have as much influence as points that are close. Therefore, the influence of all far field points can be approximated using a Laurent-type expansion while the influence of nearby points can be summed directly. The form of the far field (as well as the near field and their translations) depends on the form of the basis function.

Such expansions are computed by first hierarchically partitioning the entire space into a set of disjoint panels or boxes as shown in figure 4.1. Each level of the hierarchy has 2^{d^l} boxes where d is the dimension of the space. Thus in 3D there are 8^l boxes and in 2D there are 4^l boxes, where l is a given level. The top or zeroth level of the hierarchy contains the entire space, the first level contains 8 boxes (in 3D), the second level contains 64 boxes (in 3D) and so on. The number of levels chosen is usually about $\log_{2^d} N$.

Once we have partitioned our space to the desired depth, the various expansions can be computed in two stages: the upward pass and the downward pass. In the upward pass, the far field sums are computed in the lowest level of the hierarchy for each of the boxes. Then we work our way up the hierarchy and convert the sums of the 2^d

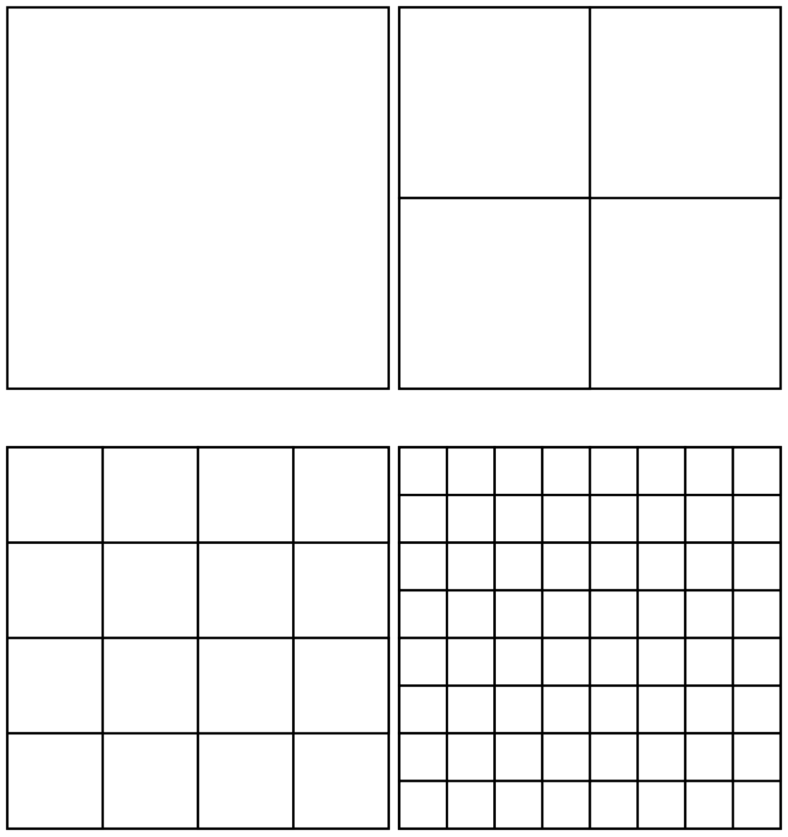

Figure 4.1: Top four levels of the FMM hierarchy.

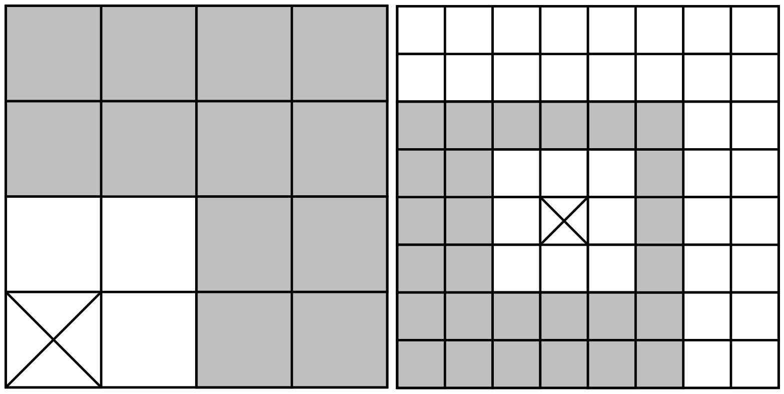

Figure 4.2: Interaction lists (in gray) of boxes marked with an X at levels 2 and 3.

children into a single sum for the parent by shifting and summing each of the 2^d children expansions. We stop when we complete the second level since above that all boxes are nearest neighbors of every other box. Formulas for expansions and proofs for the shifting and convergence have been derived by Cherrie et al. (2002) for multiquadrics, which includes 3D thin plate splines as a special case (when $\tau = 0$ and $k = 1$). In the downward pass, we work our way down the hierarchy and for the ith box, we convert the far field expansions of all other boxes that are (a) well separated from i and (b) children of the nearest neighbors of ith's parent (also known as the *interaction list* (Greengard and Rokhlin, 1987)) to local Taylor expansions. We call this local Taylor expansion the near field. Examples of interaction lists at levels 2 and 3 are shown in figure 4.2. Since there are at most $3^d - 1$ nearest neighbors for a given box and since each box has 2^d children, the size of the interaction list is at most $3^d \cdot (2^d - 1)$. Thus in 2D there at most 27 boxes in the interaction list and in 3D there are at most 189. Again, proofs for the conversion and shifting of the near field have been derived by Cherrie et al. (2002). The final local near field has the form

$$Near(\vec{x}) = \sum_{|\alpha|\le r} c_\alpha \vec{x}^\alpha \tag{4.2}$$

which is simply a 3D polynomial of degree r, where $\vec{x} = (x_1, x_2, x_3)$ is a point in $\mathbf{R}^3$ and α is a multi-index [1].

Finally, to evaluate the original sum (4.1) we add the influence of all points that are either in the current box or one of its nearest neighbors as well as the near field which represents the contribution from all far away points:

$$f(\vec{x}) = \sum_{i=1}^{N} d_i \phi_i(|\vec{x} - \vec{x}_i|) \approx Near(\vec{x}) + Direct(\vec{x}) \tag{4.3}$$

where

$$Direct(\vec{x}) = \sum_{i=1}^{M} d_i \phi_i(|\vec{x} - \vec{x}_i|). \tag{4.4}$$

is the sum over the points within the box containing $\vec{x}$ and its nearest neighbors, where we are assuming they contain a total of M points. Figure 4.3 shows pseudocode of the complete algorithm.

Now, using Eq. 4.3, the transformation $[u(\vec{x}), v(\vec{x}), w(\vec{x})]$ in (2.18) can be expressed as

$$u(\vec{x}) = x_1 + \sum_{\alpha\le r} c^{(1)}_{\alpha,j} \vec{x}^\alpha + \sum_{i=1}^{M} d^{(1)}_i \phi^{(1)}_i (|\vec{x} - \vec{x}_i|) \tag{4.5}$$

[1] A multi-index is a list of n integers where $|\alpha| \le r$ is defined as $\alpha_1 + \alpha_2 + \cdots + \alpha_n \le r$ and $\vec{x}^\alpha := x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$.

```
// Upward pass
**for** each box at level $i$ **do**
    compute the far field expansion for each box
**end for**
**for** $i$ from second to bottom level to level 2 **do**
    **for** each box at level $i$ **do**
        compute the far field expansion by shifting the 8 children
    **end for**
**end for**

// Downward pass
**for** $i$ from level 2 to bottom level **do**
    **for** each box at level $i$ **do**
        convert far field to near field
    **end for**
    **for** each box at level $i$ **do**
        shift near field
    **end for**
**end for**
To evaluate sum: add near field and direct points
```

Figure 4.3: FMM algorithm pseudocode.

$$v(\vec{x}) = x_2 + \sum_{\alpha \leq r} c^{(2)}_{\alpha,j} \vec{x}^{\alpha} + \sum_{i=1}^{M} d^{(2)}_i \phi^{(2)}_i (|\vec{x} - \vec{x}_i|) \tag{4.6}$$

$$w(\vec{x}) = x_3 + \sum_{\alpha \leq r} c^{(3)}_{\alpha,j} \vec{x}^{\alpha} + \sum_{i=1}^{M} d^{(3)}_i \phi^{(3)}_i (|\vec{x} - \vec{x}_i|) \tag{4.7}$$

where j is the index of the jth box at the finest level of the hierarchy. Note that each box at the finest level has its own set of coefficients $c^{(1)}_{\alpha,j}, c^{(2)}_{\alpha,j}$, and $c^{(3)}_{\alpha,j}$, while the coefficients $d^{(1)}_i, d^{(2)}_i$, and $d^{(3)}_i$ are common to all the boxes.

4.1.1 Analytical Form of the Expansions and Translations

In this section, we state the form of the expansions for the near and far fields of multiquadric RBFs, their translations, and the conversion of the far field to a local near field. We omit the proofs which can be found in the paper by Cherrie et al. (2002).

- The Far Field: This polynomial is computed in step 1 for each box in the bottom level of the hierarchy. As derived by Cherrie et al. (2002, Lemma 3.1), the expansion for a single center is

$$\Phi(\vec{x} - \vec{t}) = (|\vec{x} - \vec{t}|^2 + \tau^2)^{k/2} = \sum_{\ell=0}^{\infty} \frac{P^{(k)}_{\ell}(|\vec{t}|^2 + \tau^2, -2\langle \vec{t}, \vec{x} \rangle, |\vec{x}|^2)}{|\vec{x}|^{2\ell - k}} \tag{4.8}$$

  where $\langle \vec{t}, \vec{x} \rangle$ is an inner product, and

$$P^{(k)}_{\ell}(a,b,c) = \sum_{j=\lfloor \frac{\ell+1}{2} \rfloor}^{\ell} \binom{k/2}{j} \binom{j}{\ell - j} b^{2j-\ell}(ac)^{\ell-j}, \qquad \ell \geq 0. \tag{4.9}$$

For N centers we therefore have

$$\sum_{i=1}^{N} d_i \Phi(\vec{x}-\vec{t}_i) = \sum_{i=1}^{N} d_i (|\vec{x}-\vec{t}_i|^2+\tau^2)^{k/2} = \sum_{\ell=0}^{\infty}\sum_{i=1}^{N} d_i \frac{P_\ell^{(k)}(|\vec{t}|^2+\tau^2, -2\langle\vec{t},\vec{x}\rangle, |\vec{x}|^2)}{|\vec{x}|^{2\ell-k}}. \tag{4.10}$$

Now, how do we know that these polynomials converge, and, even if they do converge, do they converge quickly or slowly? To answer this, Cherrie et al. (2002) proved that the absolute difference between the truncated expansion of (4.10) (i.e. summing to $p+k$ rather than to infinity) and the infinite expansion is

$$|s(\vec{x}) - s_p(\vec{x})| \leq \begin{cases} 2^k M R^k (\frac{1}{c})^{p+1} \frac{1}{1-1/c}, & k>0 \\ \binom{p}{p+k+1} M R^k (\frac{1}{c})^{p+1} (\frac{1}{1-1/c})^{-k}, & k<0 \end{cases} \tag{4.11}$$

where $|\vec{x}| > R = \sqrt{r^2+\tau^2}$, r is the radius of the expansion, $c = |\vec{x}|/R$, and $M = \sum_{i=1}^{N} |d_i|$. Additionally, Cherrie et al. (2002) has proved that this expansion is unique.

To actually form these expansions, one should not use these formulas directly. Instead there is a recurrence relation which is more efficient. Letting

$$G_\ell(\vec{x}) = P_\ell^{(k)}(|\vec{t}|^2+\tau^2, -2\langle\vec{t},\vec{x}\rangle, |\vec{x}|^2), \tag{4.12}$$

the recurrence is

$$G_\ell(\vec{x}) = \begin{cases} 1, & \ell = 0, \\ -k\langle \vec{t}, \vec{x} \rangle, & \ell = 1 \\ A_\ell \langle \vec{t}, \vec{x} \rangle G_{\ell-1}(\vec{x}) + B_\ell |\vec{x}|^2 (|\vec{t}|^2 + \tau^2) G_{\ell-2}(\vec{x}), & \ell \geq 2 \end{cases} \tag{4.13}$$

where

$$A_\ell = -2\frac{k/2 - \ell + 1}{\ell} \tag{4.14}$$

and

$$B_\ell = -\frac{\ell - k - 2}{\ell}. \tag{4.15}$$

Cherrie et al. (2002) presents pseudocode for the 2D case only. Therefore, we present it here for the 3D case as shown in figure 4.4

- Far Field Translation: During the upward pass it is necessary to translate each of the far field expansions at a given level to the center of its parent. This calculation is quite involved and we therefore refer the reader to the paper by Cherrie et al. (2002) for the details. Basically, if the original polynomial is $s_p(\vec{x})$ and we then shift it by $\vec{u}$, the shifted polynomial is of the form

$$s_p(\vec{x} - \vec{u}) = \sum_{\ell=0}^{p+k} \frac{\widehat{Q}_\ell(\vec{x})}{|\vec{x}|^{2\ell-k}} \tag{4.16}$$

```
int $i, j, \ell$
double $A_\ell, B_\ell, a, b, tmp$
$G(0,0,0) = d$
$G(1,0,0) = -d * k * t_0$
$G(1,1,0) = -d * k * t_1$
$G(1,0,1) = -d * k * t_2$
**for** $\ell$ from 2 to $p + k$ **do**
  $A_\ell = -2 * (k/2 - \ell + 1)/\ell$
  $B_\ell = (k - \ell + 2)/\ell$
  $a = A_\ell$
  $b = B_\ell * (t_0^2 + t_1^2 + t_2^2 + \tau^2)$
  **for** $i$ from 0 to $\ell - 1$ **do**
    **for** $j$ from 0 to $\ell - 1 - i$ **do**
      **if** $i + j \leq \ell - 2$ **then**
        $tmp = G(\ell - 2, i, j) * b$
        $G(\ell, i, j)$ += $tmp$
        $G(\ell, i + 2, j)$ += $tmp$
        $G(\ell, i, j + 2)$ += $tmp$
      **end if**
      $tmp = G(\ell - 1, i, j) * a$
      $G(\ell, i, j)$ += $tmp * t_0$
      $G(\ell, i + 1, j)$ += $tmp * t_1$
      $G(\ell, i, j + 1)$ += $tmp * t_2$
    **end for**
  **end for**
**end for**
```

Figure 4.4: Polynomial generator in 3D.

where

$$\widehat{Q}_\ell(\vec{x}) = \sum_{j=0}^{\ell} q_j(\vec{x}) P_{\ell-j}^{(-2p+k)}(|\vec{u}|^2, -2\langle\vec{x},\vec{u}\rangle, |\vec{x}|^2), \quad 0 \le \ell \le p+k, \tag{4.17}$$

and $q_j(\vec{x})$ is a homogeneous polynomial as defined by Cherrie et al. (2002).

- Near Field: The near field is

$$\Phi(\vec{x}-\vec{u}) = (|\vec{x}-\vec{u}|^2+\tau^2)^{k/2} = \sum_{\ell=0}^{\infty} \frac{P_\ell^{(k)}(|\vec{x}|^2, -2\langle\vec{u},\vec{x}\rangle, |\vec{u}|^2+\tau^2)}{(\sqrt{|\vec{u}|^2+\tau^2})^{2\ell-k}} \tag{4.18}$$

which is just a regular polynomial.

- Conversion of Far Field to Near Field: Converting the far field to the near field is very similar to shifting the far field. Again, Cherrie et al. (2002) derived error bounds for this approximation:

$$\left|\Phi(\vec{x}-\vec{u}) - \sum_{\ell=0}^{q} \frac{P_\ell^{(k)}(|\vec{x}|^2, -2\langle\vec{u},\vec{x}\rangle, |\vec{u}|^2+\tau^2)}{(\sqrt{|\vec{u}|^2+\tau^2})^{2\ell-k}}\right| \le \begin{cases} (\sqrt{|\vec{u}|^2+\tau^2})^k \left(\frac{|\vec{x}|}{\sqrt{|\vec{u}|^2+\tau^2}}\right)^{q+1} \frac{\sqrt{|\vec{u}|^2+\tau^2}}{\sqrt{|\vec{u}|^2+\tau^2}-|\vec{x}|}, & k>0 \\ \binom{q-k}{q+1}(\sqrt{|\vec{u}|^2+\tau^2})^k \left(\frac{|\vec{x}|}{\sqrt{|\vec{u}|^2+\tau^2}}\right)^{q+1} \left(\frac{\sqrt{|\vec{u}|^2+\tau^2}}{\sqrt{|\vec{u}|^2+\tau^2}-|\vec{x}|}\right)^{-k}, & k<0 \end{cases}. \tag{4.19}$$

- Near Field Translation: This is just a polynomial shift which is equivalent to a convolution. Cherrie et al. (2002) therefore suggests using the fast Fourier transform to do this calculation. However, unless the degree of the polynomial is very large, computing the convolution using the FFT is totally impractical. Therefore, it would seem that a direct polynomial multiplication is more efficient.

Thus, suppose we have a polynomial of the form

$$p(x_1, x_2, x_3) = \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} \sum_{k=0}^{n_k} s_{ijk} x_1^i x_2^j x_3^k. \tag{4.20}$$

Then using the binomial formula

$$(y+z)^n = \sum_{k=0}^{n} \binom{n}{k} y^k z^{n-k} \tag{4.21}$$

(where y and z are scalars) we have that the shifted polynomial is

$$p(x_1 + a, x_2 + b, x_3 + c) = \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} \sum_{k=0}^{n_k} s_{ijk} \sum_{k_i=0}^{i} \sum_{k_j=0}^{j} \sum_{k_k=0}^{k} \binom{i}{k_i} \binom{j}{k_j} \binom{k}{k_k} \times x_1^i x_2^j x_3^k a^{n_i-k_i} b^{n_j-k_j} c^{n_k-k_k}. \tag{4.22}$$

Unfortunately, this is the most intensive part of the FMM calculation as can be seen from the six sums in the formula, and better ways are needed to speed it up.

4.1.2 The $O(N \log N)$ and $O(N)$ Algorithms

The above description is the original formulation of the FMM and has $O(N)$ complexity. This can be seen from the fact that only on the bottom level of the upward pass are all the points accessed but not at the remaining levels of the upward pass or any of the levels of the downward pass. Therefore, it is not too difficult to show that the bottom level has complexity $O(N)$ while the remaining levels of the upward pass and the entire downward pass are independent of N. Furthermore, a single evaluation of the near field and far field is also independent of N. Hence, the total complexity of evaluating the sum at all N points is $O(N)$.

However, when dealing with points in 3D, although the complexity is proportional to N, the constant of proportionality is very large and implementing the full $O(N)$ FMM is very time-consuming and usually not worth the effort unless there are literally hundreds of thousands of points. This is because in 3D the size of the interaction list is at most 189, as explained in section 4.1, hence requiring that many conversions of far field expansions to local Taylor expansions for each box at each level of the downward pass. Therefore, a simpler approach is to stop after the upward pass (without doing the downward pass) and evaluate the far fields (i.e. Eq. (4.10)) directly. Additionally, in the upper levels of the upward pass it is simpler to form the far fields directly using Eq. (4.10) rather than using the translation formula, Eq. (4.16). Because all points are now accessed at each level, each level has complexity $O(N)$, and because the number of levels is about $\log N$, the total complexity is $O(N \log N)$ (Beatson and Greengard, 1997). In this work, we only use the $O(N \log N)$ algorithm since all experiments were done in 3D.

4.2 Solving for the RBF Coefficients

In the original fast multipole algorithm developed for the N-body problem, the coefficients in the sum (4.1) were known. The difficulty was actually computing the sums. However, in the interpolation problem we have the additional problem of not knowing the coefficients d_i. Initially, all we know are the values of the function at scattered points. It is then necessary to solve a large linear system to compute the coefficients d_i. Clearly, solving such systems directly has complexity $O(N^3)$ and is too expensive for large point sets. We therefore review in this section other approaches in the literature to this problem.

4.2.1 The Linear System

The linear system for solving for the coefficients d_i can be written as (Rohr et al., 2001)

$$\mathbf{A} \begin{bmatrix} \vec{d} \\ \vec{a} \end{bmatrix} = \begin{bmatrix} \vec{p} \\ 0 \end{bmatrix} \tag{4.23}$$

where

$$\mathbf{A} = \begin{bmatrix} (\mathbf{\Phi} + N\lambda\mathbf{W}^{-1}) & \mathbf{L} \\ \mathbf{L}^T & 0 \end{bmatrix}, \tag{4.24}$$

$$\mathbf{W}^{-1} = \begin{bmatrix} \sigma_1^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_n^2 \end{bmatrix}, \tag{4.25}$$

σ_i are weights, $\vec{d}$ are the coefficients from Eq. (4.1), $\vec{a}$ are the coefficients of the optional polynomial terms, Φ_{ij} are the basis functions $\phi(|\vec{x}_i - \vec{x}_j|)$ from Eq. (4.1), and L_{ij} are the optional polynomial terms.

The parameter λ controls whether or not the RBF approximates or interpolates the true function values. In the interpolating case $\lambda = 0$ while for the approximating case $\lambda > 0$. The higher λ is, the more approximate the resultant interpolant.

4.2.2 Fast Iterative Solutions of Radial basis functions with Preconditioned GMRES

As mentioned previously, solving the linear system (4.23) directly requires $O(N^3)$ operations, which is prohibitive for large systems. Therefore, fast iterative methods have been developed to solve such systems which reduce the complexity significantly even for large numbers of points (Beatson et al., 1999, 2001b). We focus here on one discussed by (Beatson et al., 1999) which uses the Generalized Minimum Residual method (GMRES) (Barrett et al., 1994; Saad and Schultz, 1986).

The GMRES belongs to class of solvers known as Krylov subspace methods. In such methods, at each iteration the next candidate solution vector is computed based on all the previous solution vectors.

One problem is that the matrix A in Eq. (4.23) is typically ill-conditioned and hence convergence is very slow. The solution is to use a preconditioning matrix M. For simplicity, we focus here on left preconditioning only and assume there is no polynomial part, i.e. $A = \Phi$.

A good left preconditioner M when multiplied by the original matrix system (4.23) makes it well-conditioned. The closer MA is to the identity matrix, the better conditioned the new system. Thus, it follows that our goal is find an *approximate* inverse M such that

$$MA \approx I \quad or \quad M \approx A^{-1}. \tag{4.26}$$

A standard method for finding the *exact* inverse of any $N \times N$ invertible matrix is to

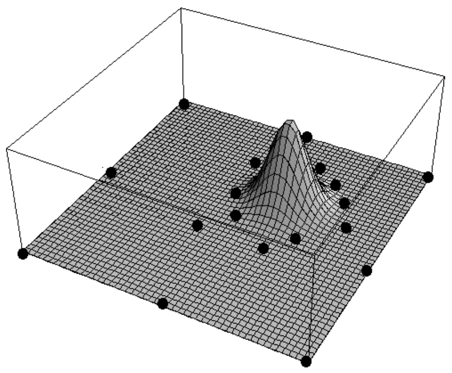

Figure 4.5: Cardinal Function in 2D. Eight points are located far away along the perimeter of the region while the remaining points are close to the center point. The cardinal function is constrained to zero at all the black points and equals unity at its maximum.

solve the following N linear systems:

$$A\vec{v}_i = \vec{e}_i, \qquad i = 1 \ldots N \tag{4.27}$$

where $\vec{e}_i$ is the ith unit vector. The inverse A^{-1} is then formed by placing $\vec{v}_i$ into the ith column of A^{-1} (Strang, 1988).

Clearly, solving (4.27) for every $\vec{v}_i$ is more difficult than solving our original problem (4.23). However, there is a fast way to find an approximate inverse (Beatson et al., 1999; Brown et al., 2005). Each of the N linear systems of Eq. (4.27) is equivalent to solving the following problem. Suppose we have a *cardinal function* centered at one of the ith control points. A cardinal function is equal to one at the ith point and is zero at all the other $N-1$ points. (See figure 4.5). Suppose we wish to express this function in the form of Eq. (4.1). Then the system we need to solve is precisely the ith system of Eq. (4.27). The ith solution vector of all these N systems becomes the ith column of A^{-1}.

While we seem to have N $N \times N$ systems of equations (which is impractical), we now show that this can reduced to N $R \times R$ systems where $R \ll N$. We do this by solving for an *approximate* cardinal function. An approximate cardinal function is equal to one at the point it is centered on, like the regular cardinal function, but it is constrained to zero at only $R-1$ of the N points where $R \ll N$, rather than at all the remaining $N-1$ points. These $R-1$ points should be chosen such that most of them are very close to the center point while a few of them should be far away to make sure that the approximate cardinal function does not blow up far away from the center point.

To express each of these approximate cardinal functions as a sum of radial basis functions requires solving N $R \times R$ system. Since such systems are small, they can be solved easily using a standard algorithm such as LU decomposition. Thus the cost of solving all these systems is $O(NR^3)$. With these solutions, we can then form the *approximate* inverse M by placing the ith solution in the ith column of M where each component of the solution is placed in its corresponding row. M will be mostly sparse since all remaining $N-R$ components are set to zero.

4.3 Other Approaches: Compact Support Radial Basis Functions

In order to deal with the computational difficulties associated with RBFs, researchers have developed what are known as *compact support radial basis functions* (Buhmann, 2000; Wendland, 1995; Wu, 1995). Some have used them for nonrigid registration

(Fornefett et al., 2001; Rohde et al., 2003). These are like radial basis functions except that they are truncated to zero after a certain point. For instance, one possibility is

$$\phi(r) = (\max(1-r,0))^4(3r^3+12r^2+16r+4), \qquad r \geq 0. \tag{4.28}$$

Solving the interpolation matrix is therefore a lot simpler. However, compact RBFs appear to be inferior to and less accurate than standard non-compact RBFs (Zhang et al., 2000). Given that there *are*, in fact, ways to efficiently solve the interpolation matrix for non-compact RBFs, as discussed in this chapter, we do not use compact RBFs in this work.

Chapter 5

Using Automatic Differentiation with Registration

We now have all the pieces we need to build a registration algorithm based on automatic differentiation. In this chapter we discuss general issues that arise when implementing such an algorithm using an SSD or MI metric regardless of the type of transformation used. In the next chapter we discuss the particular case of RBF transformations.

5.1 When to Use Checkpointing

In certain situations checkpointing can be avoided depending on the form of the objective function and the transformation function. If the objective function F can be expressed as a sum over the image:

$$F = \sum_{V} f_i = \sum_{V_1} f_i + \sum_{V_2} f_i + \cdots + \sum_{V_N} f_i \tag{5.1}$$

where the image space V is broken up into pieces $V_1, V_2, \ldots, V_N$, then, from the linearity of this formula, the gradient ∇F is the sum

$$\nabla F = \nabla \sum_{V} f_i = \nabla \sum_{V_1} f_i + \nabla \sum_{V_2} f_i + \cdots + \nabla \sum_{V_N} f_i. \tag{5.2}$$

Hence, we can divide the image into pieces small enough so that trace of the individual pieces can fit into memory.

In practice, this property holds for SSD metrics that use an affine or B-spline transformation. However, for information theoretic objectives which require the computation of a histogram and do not sum over the image voxels directly, this property is usually not valid, hence requiring the need for checkpointing.

Also, when using a radial basis function transformation which first requires the solution of a linear system prior to the sum, this strategy will not work. In order to compute the gradient, the objective function must depend directly on the independent variables. Hence if we were to apply this strategy directly, then we would need to solve the entire linear system for each block. Thus, checkpointing would be necessary in this situation as well. Note that for rigid, affine, or B-spline transformations there is no setup stage and dividing the image in pieces is therefore feasible.

5.2 Interpolation

For best results in automatic differentiation, it is important that the function be continuous. When one uses AD on functions the depend on image values, we must properly handle image interpolation. After transforming a voxel in the fixed image to the space of the moving image, it is necessary to interpolate at that point using, perhaps, trilinear interpolation since this point is usually at a non-integer location. However, it follows from here that these transformed points depend directly on the independent variables. In other words, if we view the moving image as a look-up table, then the mapped point from the transformation is basically a non-integer index into this lookup table. Now, since there are millions of voxels in this image, and since in many AD implementations any variable that depends on the independent variables must be declared as a special datatype, it would appear to be necessary to have such an *index* declared as a special datatype.[1] This would create a huge computational burden. Furthermore the function would not be continuous.

Instead, a better way is that whenever we interpolate within the image we use a first-order Taylor approximation. Suppose we wish to compute the intensity value at the (non-integer) point (x_1, x_2, x_3) where the intensity at that point is I and the gradient is $(g_{x_1}, g_{x_2}, g_{x_3})$. Then using a first-order Taylor approximation, the intensity at that point can be expressed as

$$I = f + x_1 g_{x_1} + x_2 g_{x_2} + x_3 g_{x_3} \tag{5.3}$$

where f is a constant. In Eq. (5.3) x_1, x_2, x_3, and I are special AD variables since they

[1]For example, using ADOLC's active indices.

depend on the independent variables whereas f, g_{x_1}, g_{x_2}, and g_{x_3} are not.

Thus, whenever we wish to compute the intensity of an image at a given location, we

1. Compute the intensity I and gradient $(g_{x_1}, g_{x_2}, g_{x_3})$ at (x_1, x_2, x_3) using trilinear interpolation without using any special variables.

2. Use Eq. (5.3) to set the intensity (*with* special variables for x_1, x_2, x_3, and I).

3. Finally, use the result in the objective function. In this way, the intensity of the interpolated point depends continuously on (x_1, x_2, x_3).

This method generalizes easily to higher order interpolation. Instead of Eq. (5.3), there will be more terms depending on the type of interpolation used.

We mentioned in section 3.5 that the trace of objective functions we deal with in image registration are unique to a particular set of independent variables and cannot be reused with a different set of variables. From the discussion in the last two paragraphs, we can now understand why this is so. The entire moving image is not actually traced. Instead we only interpolate within the moving image when necessary. If we were to change the independent variables, then we may need to interpolate at other locations in the moving image thus rendering the original trace invalid.

5.3 Histograms

An issue that arises when using information based metrics is the need to compute histograms. Unfortunately, the same problem that we had with interpolating the image also exists with creating the histogram. Namely, the bins of the histogram are usually expressed as a raw array of values, and the moving image value (which depends directly on the independent variables of the objective function) needs to be cast to an integer index value in order to increment the frequency of an individual bin. One way around this problem is to use Parzen windows (Duda et al., 2000). Parzen windows is a method to construct a histogram so that it is smooth. In the usual way to create a histogram each bin can only have discrete, integer values. With Parzen windows, each bin can take on any real value, and instead of incrementing a single bin at a time, we increment all the bins in a neighborhood or "window" of a specific bin by the value of a windowing function centered on the bin. Popular windowing functions include Gaussian and B-splines. If we use a cubic B-spline as a windowing function, then a one dimensional histogram (in practice the histogram will be 2D but the result is the same) can be written as (cf. Mattes et al., 2003, Eq. (6))

$$p(l) = \sum_{l} \beta^{(3)}(l - f) \tag{5.4}$$

where f is the (scaled) moving image value and $\beta^{(3)}(u)$ is the cubic B-spline basis function

$$\beta^{(3)}(u) = \begin{cases} \frac{1}{6}(4 - 6u^2 + 3|u|^3), & 0 \leq |u| < 1 \\ \frac{1}{6}(8 - 12|u| + 6u^2 - |u|^3), & 1 \leq |u| < 2 \cdot \\ 0, & 2 \leq |u| \end{cases} \tag{5.5}$$

Using such Parzen windows, using histograms with AD is straightforward. However, as with interpolation, a given trace is unique for a particular set of independent variables, and if they change, the function must be recomputed.

Chapter 6

Nonrigid Registration with Radial Basis Function Transformations

In the previous chapter, we discussed general issues that arise when using AD to compute gradients of registration metrics. In this chapter, we discuss the particular case of RBF transformations. This chapter brings together all the ideas of the previous chapters and presents the complete nonrigid registration algorithm based on automatic differentiation, checkpointing, radial basis functions, and the fast multipole method—the primary goal of this work.

6.1 Preliminaries

Before the similarity metric can be optimized, several preliminaries need to be taken care of. First, as mentioned earlier, unlike B-spline grids, RBFs allow for arbitrary placement of control points throughout the fixed image. Since few control points are needed in those parts of the image which are very uniform, we choose to place control points in non-uniform areas. We use an edge detector such as the gradient magnitude or Canny edge detection filter (Canny, 1986) to finds points located along boundaries (figure 6.1) and then choose N nonzero points randomly from this edge image such that the minimum distance between points is above a certain threshold. Next, it is necessary to choose a level of refinement for the fast multipole method. This number is chosen so that there is a certain number of points per box at the lowest level (cf. Greengard, 1988). Next, we compute the preconditioning matrix using the algorithm of Beatson et al. (1999) described in section 4.2.2.

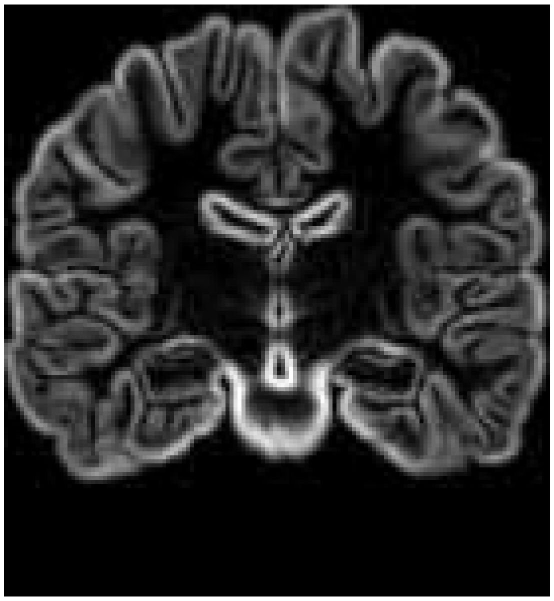

Figure 6.1: Shows a slice of the gradient magnitude of a brain image. Control points are placed on voxels with nonzero gradient magnitude.

6.2 The Similarity Metric

We are now ready to present the similarity metric which uses an RBF transformation and AD with checkpointing for gradient calculation. The C-Style pseudocode in figure 6.2 illustrates how we divided up the sum of squared differences metric into time steps and it consists of three sections.[1] This figure should be interpreted identically to figure 3.3, except that here there are many more time steps, and the direction of time is shown “vertically” rather than “horizontally”. To see this, assume that all loops in the pseudocode are fully unrolled; then the first occurrence of “[time step]” corresponds to time t_0, the second occurrence of “[time step]” corresponds to time t_1, and so on. Sometimes we write “[time steps]” in the plural to indicate that there are several time steps in that part. We only include the major loops in the GMRES and FMM sections. The contents of these loops can be found in the references referred to earlier.

Part 1: GMRES

The very beginning of the objective function evaluation is obviously the assignment of values to the independent variables, which, in this case, is the displacements of the control points. This point in time is the beginning of the very first time step, as shown in the pseudocode. The reason we optimize over the displacements rather than the coefficients d_i directly is twofold. The first reason has to do with stability: the function with respect to the displacements is much more stable than the function with respect to the coefficients d_i. Because of the non-local nature of the basis functions, a small change

[1]The mutual information metric is the same except for the third section.

```
/* Part 1. Do computation of radial basis function coefficients using GMRES for each
dimension */

[time step]
for dimensions 1 to 3 do
  [time steps]
  for $i$ from 0 to $maxIter$ do
    [time step]
    for $j$ from 0 to $m$ do
      [time steps]
    end for
    [time steps]
  end for
end for


/* Part 2. Do FMM upward pass */

[time step]
for $i$ from bottom level to level 2 do
  for each box at $i$th level do
    [time step]
  end for
end for


/* Part 3. Do sum of squared differences */

[time step]
$sum$=0
for $i$ from 0 to total voxels in image do
  if ($i$%10000 == 0) then
    [time step]
  end if
  $diff = interpolatedMovingValue - fixedValue$
  $sum = sum + diff^2$
end for
return $sum$
```

Figure 6.2: SSD Metric Pseudocode. If all loops are unrolled, then this pseudocode can be understood in the same way as figure 3.3, with the ith occurrence of “[time step]” corresponding to time t_i.

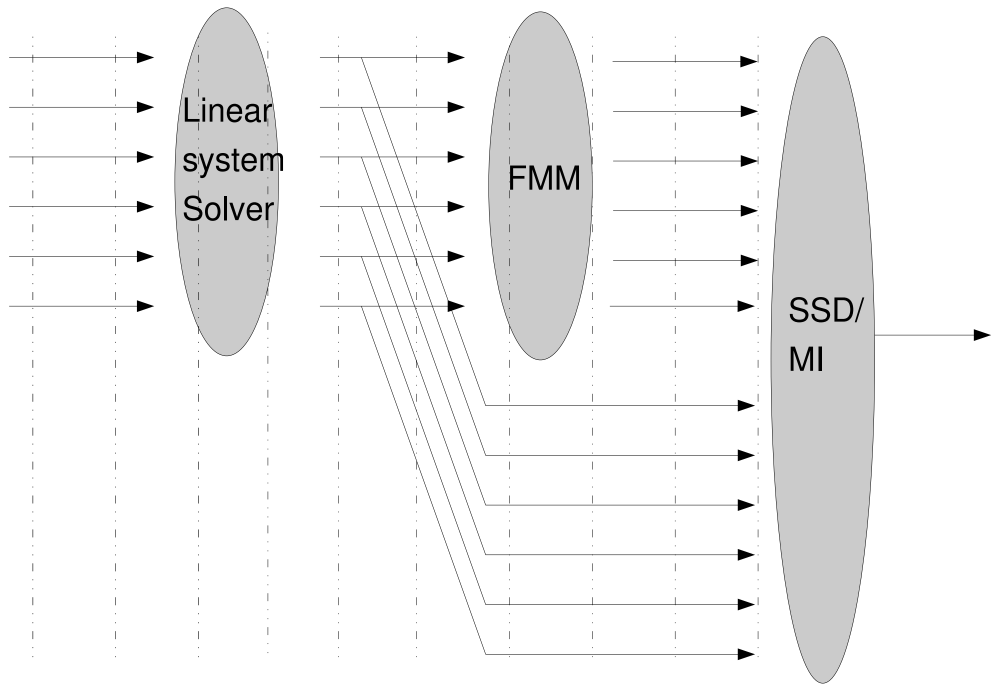

Figure 6.3: Shows the similarity metric as a directed acyclic graph. The dotted lines represent time steps. There are many more time steps than shown.

in any of the coefficients can drastically change the entire transformation. It is very hard to optimize such an unstable function. However, a small change in the displacements of the control points will not result in a major change in the transformation. The second reason is that optimizing over the displacements allows us to easily add point constraints, if desired, at arbitrary locations by placing simple bounds on the components of the displacement.

Because we are optimizing over the displacements rather than the coefficients, we need to solve for the coefficients using GMRES for each component of the transformation. GMRES, as described by Barrett et al. (1994), works by attempting to update the next solution vector based on all the previous solution vectors. All these previous vectors must therefore be stored in memory. However, due to memory constraints, we may be limited in the number of vectors we can store. One possible strategy is to restart after m iterations. However, there is no simple strategy for choosing a value for m (Barrett et al., 1994). Since performing preconditioning results in fewer necessary iterations for convergence, we chose not to restart the iteration and simply set m to the maximum number of iterations.

Another important point concerns the tolerance stopping condition. Normally, when one does checkpointing, the total number of time steps is specified beforehand. However, if the stopping criteria is based on the residual vector, then we obviously do not have this information. However, as pointed out by Griewank and Walther (2000), we are free to modify the total number of time steps prior to the start of the reverse sweep. Thus, if each iteration is composed of T time steps then initially one should allow for a maximum of $T * maxIter$ time steps. If the algorithm converges in, say, M iterations, then the number of total time steps is reduced by $T * (maxIter - M)$. A simpler approach is to eliminate the tolerance as a stopping criteria altogether and simply run the algorithm for a fixed number of iterations.

Part 2: FMM

Once the GMRES stage is completed, we then need to use the FMM to generate the necessary polynomials. As explained earlier in section 4.2.2, FMM consists of an upward pass and a downward pass where at each level of the hierarchy we iterate through each box. We do not show the downward pass, though, since, as explained in section 4.1.2, doing the full FMM in 3D is usually not worth the effort. We place a time step at each box as shown. In addition, we note that the pseudocode shown assumes that the matrix multiplication is done the standard way (with $O(N^2)$ complexity). Therefore, following the solution of the linear systems, it is necessary to compute the FMM polynomials. However, if the FMM itself is used to compute the matrix multiplication, then Part 2 will be unnecessary as it will already have been performed in Part 1 (in the last iteration of each of the three linear systems).

Part 3: Sum of Squared Difference

With the FMM complete, we can now actually compute the value of the similarity metric. We evaluate the sum of the objective function (2.18) with time steps placed every $numberOfVoxelsPerTimeStep$ voxels. The value of $numberOfVoxelsPerTimeStep$ depends on the amount of available memory. Note that if the similarity metric was mutual information, then Part 3 would be replaced with code for MI, but parts 1 and 2 would remain the same.

With the function divided up as explained, and with the ability to jump to the beginning of any arbitrary time step, one can then optimize the objective function using a gradient based method such as conjugate gradient (Press et al., 1992).

Chapter 7

Results

Having described the theory behind the approach of this work in previous chapters, we are now ready to test our algorithms on real images. However, validating image registration algorithms is not easy, and various approaches can be found in the literature (Crum et al., 2004b; Hellier et al., 2003; Pennec and Thirion, 1995; Schnabel et al., 2001; Strother et al., 1994; Warfield et al., 2001; West et al., 1997). Therefore, in this work, we made use of a variety of validation strategies, depending on the type of images registered, as not all approaches were appropriate in all situations. These techniques included:

- visual inspection.
- using an overlap measure. Crum et al. (2004b) proposed validating registration based on how well the registration aligns various structures. To measure accuracy,

the following overlap measure is used:

$$\frac{N(F \cap M)}{N(F \cup M)} \tag{7.1}$$

where $N(F \cap M)$ is the number of voxels of a specific structure in the fixed and moving images that overlap, and $N(F \cup M)$ is the number of voxels in their union. This measure varies from 0, corresponding to no overlap, to 1, corresponding to complete overlap.

- measuring the number of misclassified voxels (NMV).
- comparing registration with and without automatic differentiation.

The algorithms were coded in C++ and were mostly based on code from the ITK Insight Segmentation and Registration Toolkit (Ibanez and Schroeder, 2003). In addition, we made use of some Numerical Recipes code for implementing the conjugate gradient optimization algorithm (Press et al., 1992), code from the book *Templates for the Solution of Linear Systems* for the GMRES method (Barrett et al., 1994), the ADOLC package for automatic differentiation (Griewank et al., 1996), the Revolve package for checkpointing (Griewank and Walther, 2000), and the ANN package for nearest neighbor search (necessary for computing the preconditioning matrix—see section 4.2.2) (Mount and Arya, 2005). The code was compiled with optimization and run on a dual Intel Xeon processor workstation running the GNU/Linux operating system.

In this chapter, we begin with simple experiments and then proceed to more complicated cases.

7.1 Experiments to Evaluate Accuracy of Automatic Differentiation with Registration

It is important to make sure that using automatic differentiation with registration gives us accurate gradients. A good way to test this is to compare it to a case when analytical gradients are available. If we get equivalent results using both methods, then we can be more confident that the AD approach will work when we have no analytical gradient. In this section, we discuss implementing the AD approach on functions with analytical gradients and show that AD gives the same results as the analytical gradients. In later sections, we discuss metrics which use an RBF transformation and are therefore not amenable to analytical gradients.

To verify that AD is feasible for registration, we implemented AD on registration problems using rigid (Eq. (2.5)), affine (Eq. (2.4)), and B-spline (Eq. (2.6)) transformations with both sum of squared differences and mutual information. The code used to implement these registrations was based on the Insight Segmentation and Registration Toolkit (Ibanez and Schroeder, 2003). Tables 7.1 and 7.2 show the results of these types of transformations for both SSD and MI. In each one, we show the three transformations. The rigid transformation is a 6 parameter transformation defined by Eq. (2.5), The affine transformation is a 12 parameter transformation defined by Eq. (2.4), and the B-spline transformation consists of a $17\times18\times20$ grid with a spacing of about 10 mm for a total of 6120 control points or 18360 parameters. The images used were 2 images from the IBSR database (IBSR, 2004).[1] No checkpointing was used

[1]See section 7.3 for more discussion about this database.

for the SSD case when using AD, but checkpointing was used for MI when using AD. The first row shows the number of independent variables in each case. The second row shows the running time for computing the function alone. The third row shows the time for computing the gradient using AD. The fourth row shows the ratio of the time to compute the gradient using AD to the time to compute the function alone. The fifth row shows the time for computing the gradient analytically. The sixth row shows the ratio of the time to compute the gradient analytically to the time to compute the function alone. The seventh row shows the number of iterations in the conjugate gradient optimization. Finally, the last row shows the average relative difference between the final values of the independent variables, i.e.

$$\frac{1}{N}\sum_{i=1}^{N}\frac{|a_i - b_i|}{|a_i|} \tag{7.2}$$

where a_i are the final independent variables when using AD and b_i are the final independent variables when differentiating analytically.

We see that for the SSD metric, after several iterations, there is practically no difference between the transformation parameters in the AD and analytical optimizations. However, for MI, we see that after several iterations, differences accumulate between the analytical and AD cases. This is perhaps due to the fact that our analytical gradient was computed based on the method described by Thevenaz and Unser (2000) and Mattes et al. (2003). A close reading of those papers reveals that a Taylor approximation was made and hence it is not the true gradient. Another possibility is that the MI metric is more sensitive to tiny changes in the independent variables so that after several iterations the solution changes by a sizable amount.

You will also notice that the running times for the B-splines gradient is very long. This is simply because our particular implementation of the analytical B-spline gradient was not optimal, not because doing B-splines analytically is slower than with AD. In an optimal B-spline implementation, the analytical B-spline gradient would be faster than the AD version.

	rigid	affine	B-spline
Number indep vars.	6	12	18360
Time (function) (sec)	1.95	1.99	30.1
Time-AD (gradient) (sec)	8	8.63	181.038
Ratio-AD	4.1	4.3	6.01
Time-dir (gradient) (sec)	5.4	3.45	1548.96
Ratio-dir	2.8	1.7	51.46
Iterations	4	10	25
Norm	1.05906e-11	4.91091e-10	4.36718e-10

Table 7.1: Running times and others values for registration with an SSD metric. See text for explanation of values.

	rigid	affine	B-spline
Number indep vars.	6	12	18360
Time (function) (sec)	3.29	3.2	30.59
Time-AD (gradient) (sec)	58.8	65.4	454.15
Ratio-AD	17.87	20.4	14.846
Time-dir (gradient) (sec)	15	8	4301.6
Ratio-dir	4.97	2.7	140.6
Iterations	4	16	25
Norm	0.279176	0.106872	0.2065

Table 7.2: Running times and others values for registration with an MI metric. See text for explanation of values.

7.2 Experiments with Synthetic Shapes

7.2.1 Registration of a Sphere and a "Planet"

In this experiment, we registered two synthetic shapes, a "planet" and a sphere. Our planet is like a sphere except that it has seven "mountains" and seven "craters". A slice and a surface rendering are shown in figure 7.1. Both images are 128×128×128 voxels with a spacing of 1. We registered the sphere to the "planet" using a B-spline transformation with grid sizes of 8×8×8, 12×12×12, 16×16×16, and 20×20×20 as well as an RBF transformation with 512, 1728, 4096, and 8000 points. The points were randomly placed on the nonzero voxels of the gradient magnitude image. An SSD metric was used. For the RBF, we used a thin plate spline (multiquadric with $\tau = 0$, $k = 1$), and the degree of the polynomials used in the fast multipole calculations was 5 (i.e. $p = 4$ so that $p + k = 5$). The number of iterations used in the GMRES solver was 40. For the optimization, we used the conjugate gradient method (Press et al., 1992).

Figures 7.2-7.9 and tables 7.3-7.4 show results for these experiments. In tables 7.3-7.4, the first row shows the size of the B-spline grid. The second, third, and fourth rows are the same as in tables 7.1-7.2. The fifth row is the number of iterations taken in the optimization algorithm, and the final row is the number of misclassified voxels (NMV) following the completion of the registration. Note that prior to the registration, the number of misclassified (NMV) voxels was 51304. In table 7.4, the last two rows are the number of time steps and the maximum number of checkpoints used in the checkpointing algorithm. We see from figures 7.2-7.9 that both RBFs and B-splines

Grid Size	8^3	12^3	16^3	20^3
Time (function) (sec)	28	16	16	16
Time (gradient) (sec)	148	92	96	99
Ratio	5.28	5.75	6	6.19
Iterations	10	10	10	10
NMV	35831	16055	4681	2632

Table 7.3: Running times and other values for sphere registered to "planet" with a B-spline transformation. See text for explanation of values.

give comparable results. However, we point out that with this example, the difference between B-splines and RBFs is very noticeable. The B-spline warp is very local in nature. The only parts of the image that get warped are in the vicinity of the "mountains" and "craters". The RBF. however, is global and thus the *entire* space gets warped. Clearly, for this case, it cannot be argued that one warping is more "correct" than the other. There is no way to decide what the true transformation is without additional information.

However, suppose that this sphere is a young, "undeveloped" brain and the "planet" represents the brain at a later stage in its development. If we were to now ask what the correct transformation is, it would appear that the B-spline transformation is too simplistic. We would surely expect the change in the brain to result from the growth and development of the *entire* brain, not just along the surface. Hence, it seems to us that a global transformation such as RBFs is a more appropriate model of such brain variability since it takes into account the global nature of the brain.

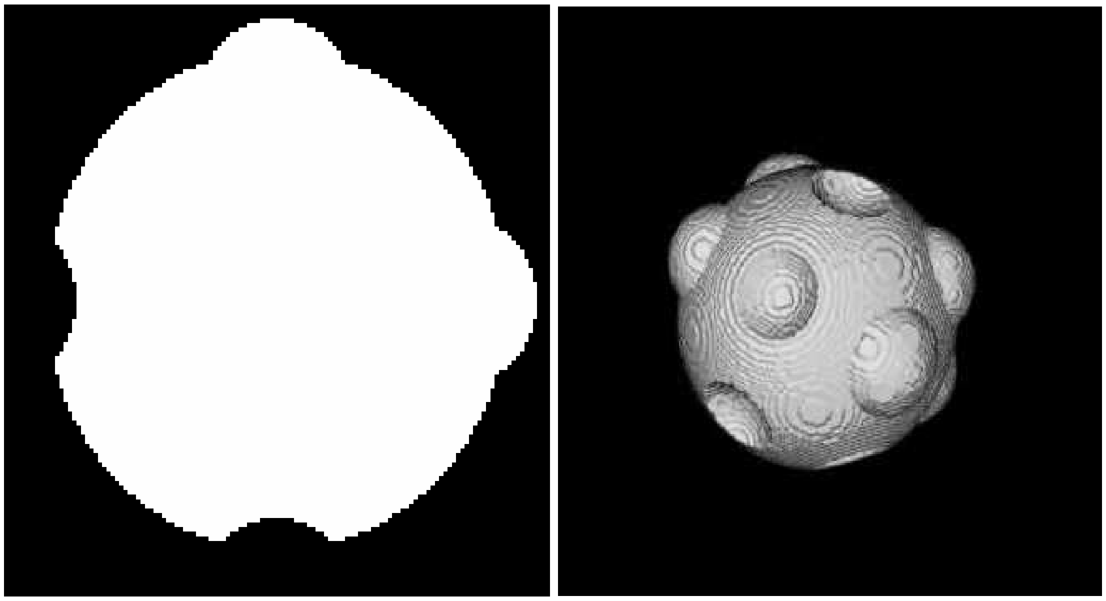

Figure 7.1: Slice and surface rendering of “planet” with “mountains” and “craters”.

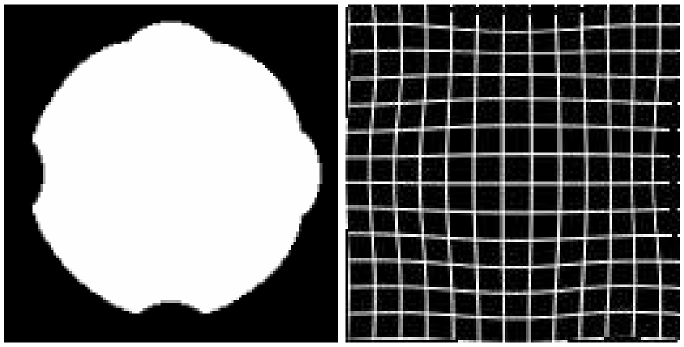

Figure 7.2: Result of sphere registered to “planet” with corresponding warped grid image for a B-spline grid of 8×8×8 (512 points).

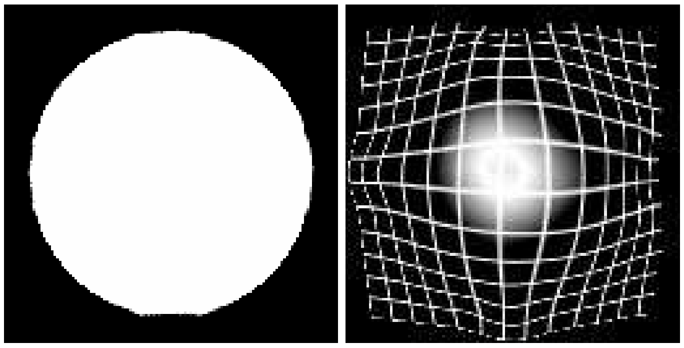

Figure 7.3: Result of sphere registered to “planet” with corresponding warped grid image for an RBF of 512 points.

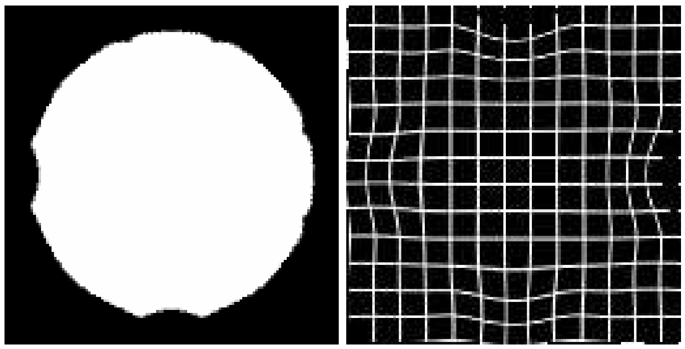

Figure 7.4: Result of sphere registered to “planet” with corresponding warped grid image for a B-spline grid of 12×12×12 (1728 points).

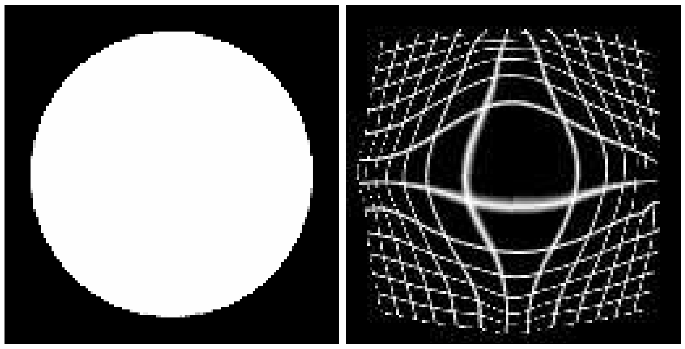

Figure 7.5: Result of sphere registered to “planet” with corresponding warped grid image for an RBF of 1728 points.

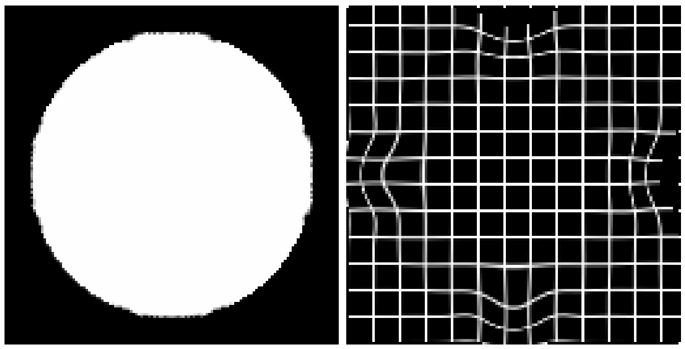

Figure 7.6: Result of sphere registered to “planet”with corresponding warped grid image for a B-spline grid of 16×16×16 (4096 points).

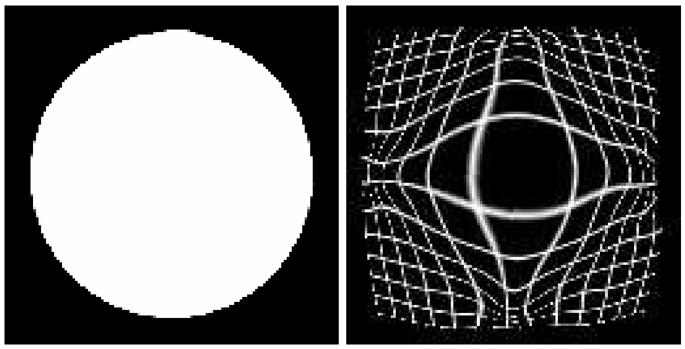

Figure 7.7: Result of sphere registered to “planet” with corresponding warped grid image for an RBF of 4096 points.

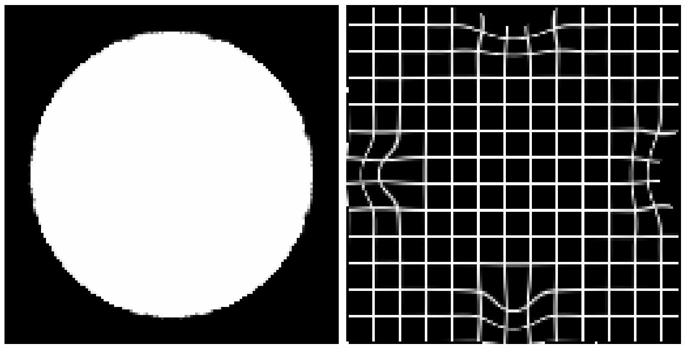

Figure 7.8: Result of sphere registered to “planet” with corresponding warped grid image for a B-spline grid of 20×20×20 (8000 points).

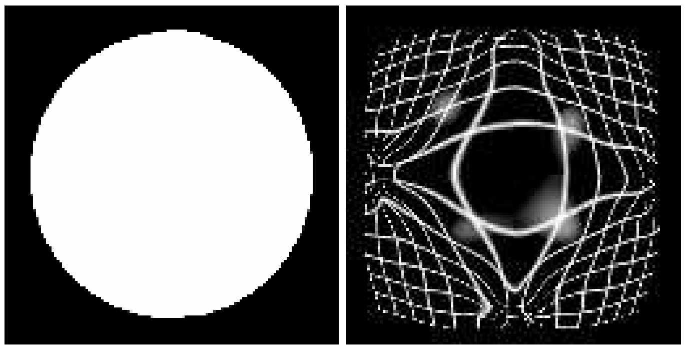

Figure 7.9: Result of sphere registered to “planet” with corresponding warped grid image for an RBF of 8000 points.

Number of Points	512	1728	4096	8000
Time (function) (sec)	190	235	297	410
Time (gradient) (sec)	1544	1870	2296	3221
Ratio	8.1	7.96	7.73	7.86
Iterations	10	10	10	10
NMV	5843	1606	1497	1891
Number of Time steps	4704	4956	5586	6468
Max number of checkpoints	20	20	20	20

Table 7.4: Running times and other values for sphere registered to “planet” with an RBF transformation. See text for explanation of values.

7.2.2 Registration of a Sphere and a 3D “C”

Christensen (1994) recommends testing nonrigid registration algorithms using a “C” shaped image. Since all experiments in this work are in 3D, we instead used a 3D “C” shaped image that is very similar to Christensen’s 2D “C” image. The dimensions are 128×128×128 and have isotropic spacing of 1. The inner radius of the “C” is 21 voxels, the outer radius is 41 voxels, and a cylinder of radius 10 is used to make the opening in the “C”. The intensity value of the “C” is 100 and the background has value 0. Orthogonal slices of the 3D “C” are shown in the figure 7.10.

We registered a sphere with a radius of 31 voxels to the “C” image using a B-spline transformation with grid sizes of 8×8×8, 10×10×10, 12×12×12, 16×16×16, and 20×20×20 as well as RBF transformation with 512, 1000, 1728, 4096, and 8000 points. The points were randomly placed on the nonzero voxels of the gradient magnitude image. An SSD metric was used. For the RBF, we used a thin plate spline (multiquadric with $\tau = 0$, $k = 1$), and the degree of the polynomials used in the fast multipole calculations was 5 (i.e. $p = 4$ so that $p + k = 5$). The number of iterations used in the

GMRES solver was 40. For the optimization, we used the conjugate gradient method (Press et al., 1992). Figures 7.11-7.20 and tables 7.5 and 7.6 show results and timing information for these experiments. The explanation for the values of these two tables are identical to those of tables 7.3-7.4. Note that prior to the registration, the number of misclassified voxels (NMV) was 203220. We see that the B-spline transformation is unable to correctly warp the sphere into the "C", whereas the radial basis function transformation can (although there is still a remaining little dot in the center which the radial basis function was unable to register).

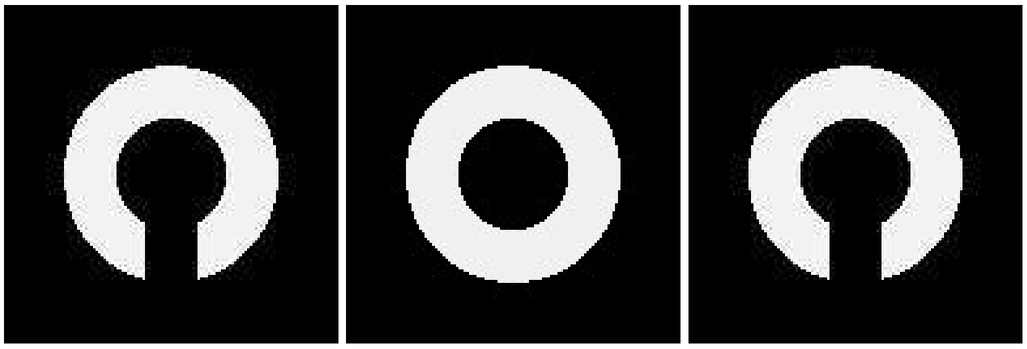

Figure 7.10: Shows 3 orthogonal slices of 3D "C" image.

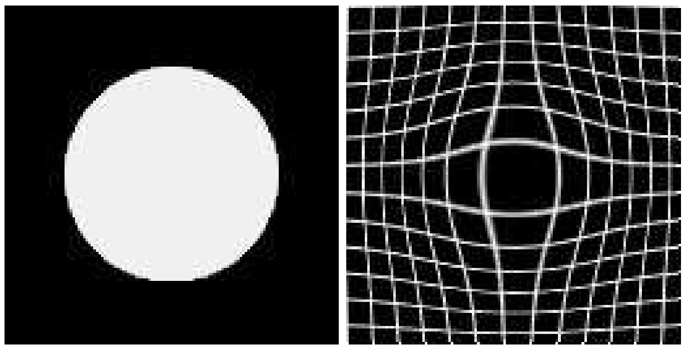

Figure 7.11: Result of sphere registered to "C" with corresponding warped grid image for a B-spline grid of 8×8×8 (total number of control points: 512).

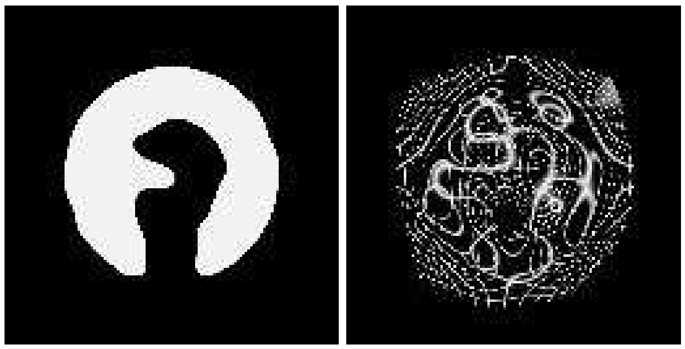

Figure 7.12: Result of sphere registered to “C” with corresponding warped grid image for an RBF transformation consisting of 512 points.

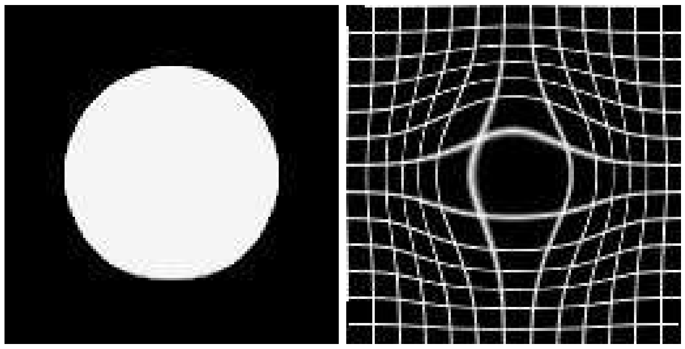

Figure 7.13: Result of sphere registered to “C” with corresponding warped grid image for a B-spline grid of 10×10×10 (total number of control points: 1000).

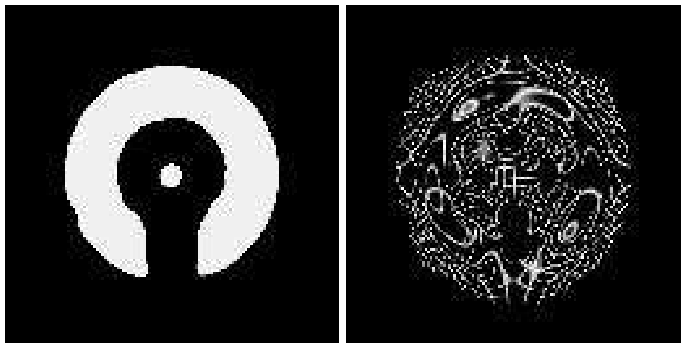

Figure 7.14: Result of sphere registered to “C” with corresponding warped grid image for an RBF transformation consisting of 1000 points.

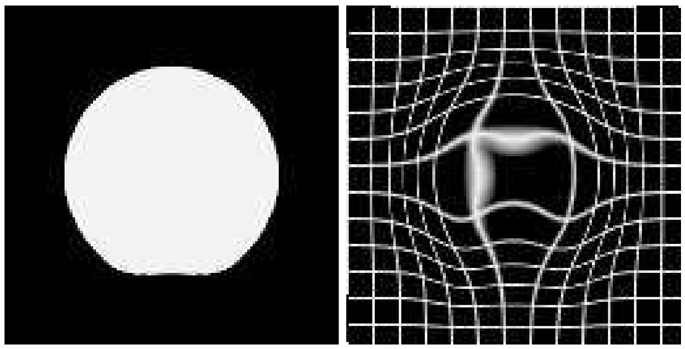

Figure 7.15: Result of sphere registered to “C” with corresponding warped grid image for a B-spline grid of $12\times12\times12$ (total number of control points: 1728).

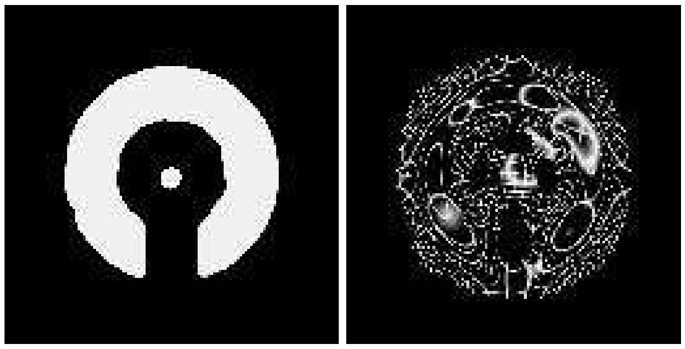

Figure 7.16: Result of sphere registered to “C” with corresponding warped grid image for an RBF transformation consisting of 1728 points.

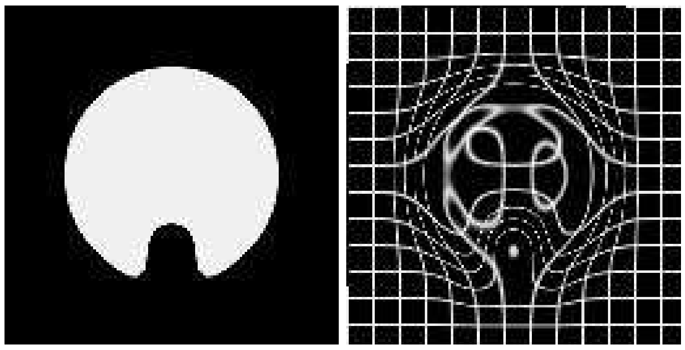

Figure 7.17: Result of sphere registered to “C” with corresponding warped grid image for a B-spline grid of $16\times16\times16$ (total number of control points: 4096).

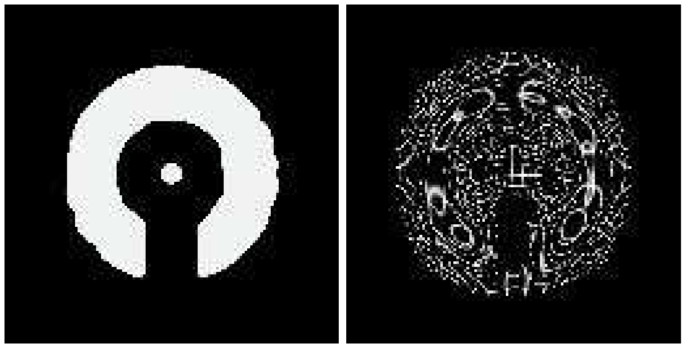

Figure 7.18: Result of sphere registered to “C” image with corresponding warped grid image for an RBF transformation consisting of 4096 points.

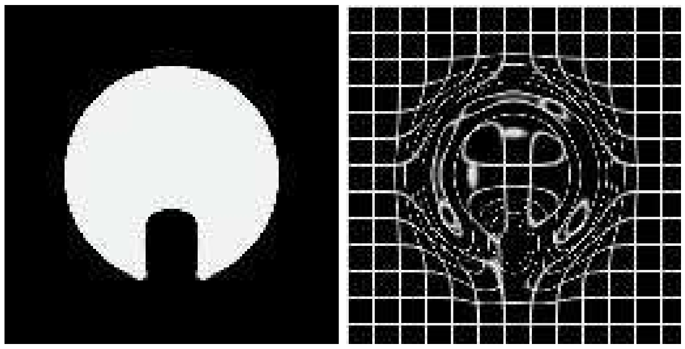

Figure 7.19: Result of sphere registered to “C” with corresponding warped grid image for a B-spline grid of 20×20×20 (total number of control points: 8000).

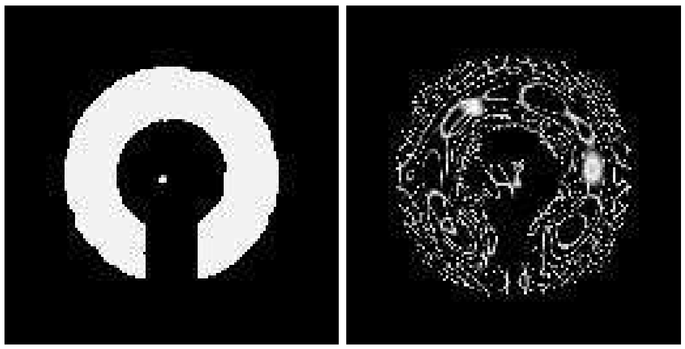

Figure 7.20: Result of sphere registered to “C” with corresponding warped grid image for an RBF transformation consisting of 8000 points.

Grid Size	8^3	10^3	12^3	16^3	20^3
Time (function) (sec)	22.3	22.12	22.1	21.7	23.1
Time (gradient) (sec)	133.2	131.72	131.09	133.09	135.5
Ratio	5.97	5.95	5.93	6.13	5.87
Iterations	5	10	10	10	10
NMV	68852	64920	62200	53036	46808

Table 7.5: Running times and other values for sphere registered to "C" with a B-spline transformation. See text for explanation of values.

Number of Points	512	1000	1728	4096	8000
Time (function) (sec)	168.18	180.31	203.55	304.7	538.13
Time (gradient) (sec)	1321.18	1390.81	1601.18	2233.1	3815.45
Ratio	7.86	7.71	7.87	7.33	7.09
Iterations	10	10	10	10	10
NMV	14286	8192	6678	9453	7110
Time steps	4704	4704	4956	5586	6468
Checkpoints	20	20	20	20	20

Table 7.6: Running times and other values for sphere registered to "C" with an RBF transformation. See text for explanation of values.

7.2.3 Registration of a Star and a Flower Shaped Image

For our next experiment, we used two shapes generated with the help of the "super-shape" formula (Bourke, 2003):

$$\begin{aligned} &x = R(\theta)\cos(\theta)R(\phi)\cos(\phi) \\ &y = R(\theta)\sin(\theta)R(\phi)\cos(\phi) \\ &z = R(\phi)\sin(\phi) \\ &\pi/2 \leq \phi \leq \pi/2 \\ &\pi \leq \theta \leq \pi \end{aligned} \tag{7.3}$$

where

$$R(x) = \left(\left| \frac{1}{a} \cos(mx/4) \right|^{n_2} + \left| \frac{1}{b} \sin(mx/4) \right|^{n_3} \right)^{-1/n_1}. \tag{7.4}$$

In this equation a, b, m, n_1, n_2, and n_3 are parameters which can be varied. The first shape, which resembles a flower, has parameters $a = 1$, $b = 1$, $m = 5.2$, $n_1 = 0.04$, $n_2 = 1.7$, and $n_3 = 1.7$ and the second shape, which resembles a star, has parameters $a = 1$, $b = 1$, $m = 5.2$, $n_1 = 0.2$, $n_2 = 1.7$, and $n_3 = 1.7$. The images are constructed by evaluating Eq. (7.3) for $\pi/2 \leq \phi \leq \pi/2$ and $\pi \leq \theta \leq \pi$ and then setting to an intensity value of 100 all voxels along the line connecting the origin and the point (x, y, z). The dimensions of the images are 128×128×128 and have isotropic spacing of 1.

We registered the two images using a B-spline transformation with grid sizes of 8×8×8, 10×10×10, 12×12×12, 16×16×16, and 20×20×20 as well as an RBF transformation with 512, 1000, 1728, 4096, and 8000 points. The points were randomly placed on the nonzero voxels of the gradient magnitude image. An SSD metric was used. For the RBF, we used a thin plate spline (multiquadric with $\tau = 0$, $k = 1$), and the degree of the polynomials used in the fast multipole calculations was 5 (i.e. $p = 4$ so that $p + k = 5$). The number of iterations used in the GMRES solver was 40. For the optimization, we used the conjugate gradient method (Press et al., 1992). Figures 7.23-7.30 and tables 7.7 and 7.8 show results and timing information for these experiments. The explanation for the values of these two tables are identical to those of tables 7.3-7.4. Note that prior to the registration, the number of misclassified voxels (NMV) was 264093. We see that the RBF registration computes a warp that better aligns the two images. In addition, as the

Grid Size	8^3	10^3	12^3	16^3	20^3
Time (function) (sec)	22.68	23.13	22.31	23.003	22.40
Time (gradient) (sec)	144.54	152.96	153.53	144.77	152.07
Ratio	6.3726	6.61	6.88	6.29	6.79
Iterations	10	10	10	10	10
NMV	38757	28486	33383	39199	39701

Table 7.7: Running times and other values for star registered to flower with a B-spline transformation. See text for explanation of values.

Number of Points	512	1000	1728	4096	8000
Time (function) (sec)	36.77	79.18	143.37	443.73	556.8
Time (gradient) (sec)	287.23	483.88	916.1	2950.04	4075.58
Ratio	7.8115	6.11	6.34	6.65	7.3
Iterations	10	10	10	10	10
NMV	31155	22366	14891	10729	10170
Time steps	4704	4704	4956	5586	6468
Checkpoints	20	20	20	20	20

Table 7.8: Running times and other values for star registered to flower with an RBF transformation. See text for explanation of values.

B-spline grid size is increased, the registration actually gets worse, though this might not have happened if we had regularized the B-spline.

7.3 Registration Evaluation with the IBSR Database

For the next set of experiments, we used magnetic resonance human brain images from the IBSR database from Harvard (IBSR, 2004). The database consists of 18 images and each one was segmented manually by experts. About 30-40 labels were then assigned to these segmented structures. Figure 7.33 shows a slice of one of these images and its corresponding segmentation. The database thus provides a good way of measuring the accuracy of the registration. In these experiments, we used images 9 and 10 from

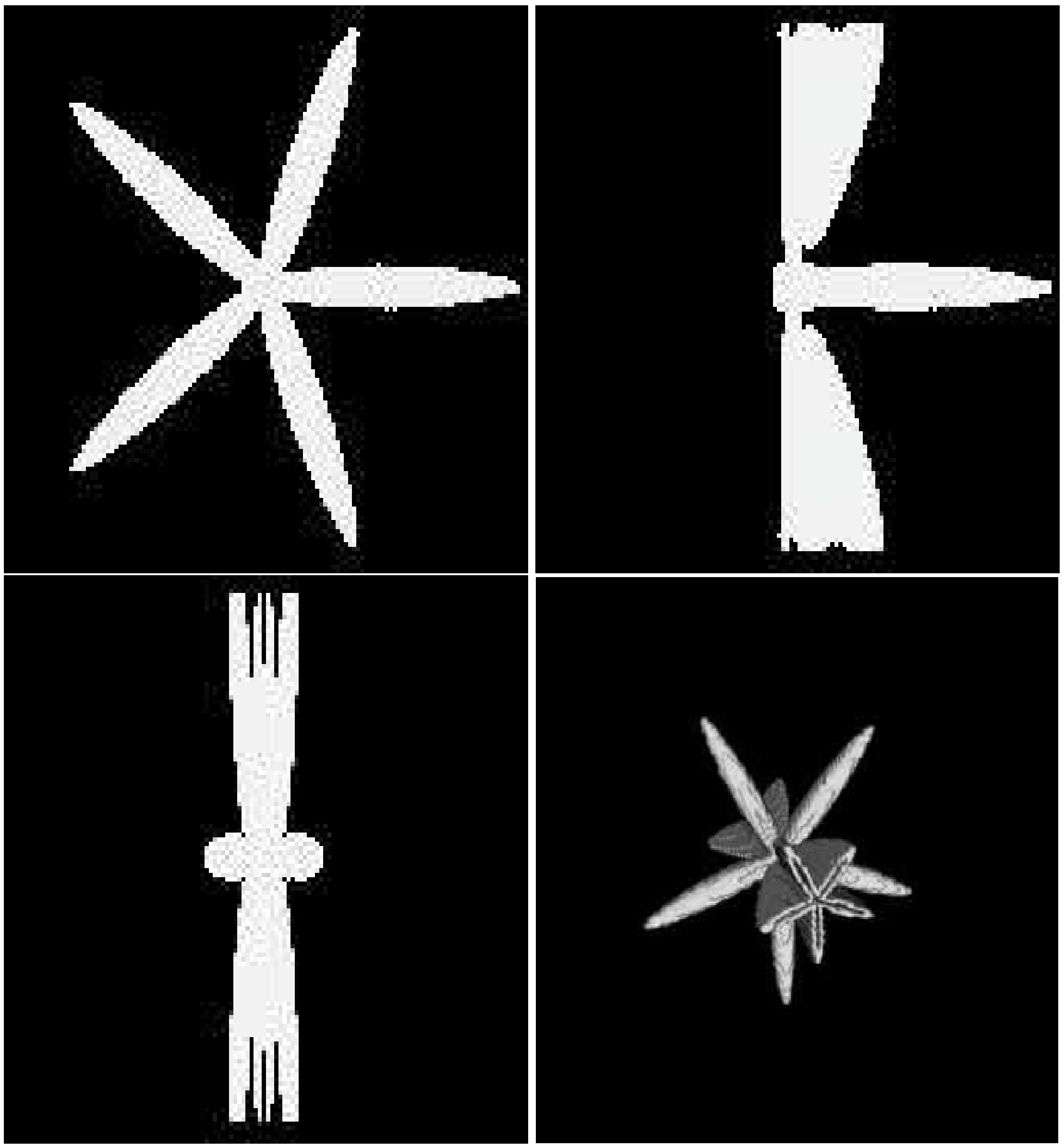

Figure 7.21: Shows 3 orthogonal slices and surface rendering of flower. The "supershape" parameters are $a = 1$, $b = 1$, $m = 5.2$, $n_1 = 0.04$, $n_2 = 1.7$, and $n_3 = 1.7$.

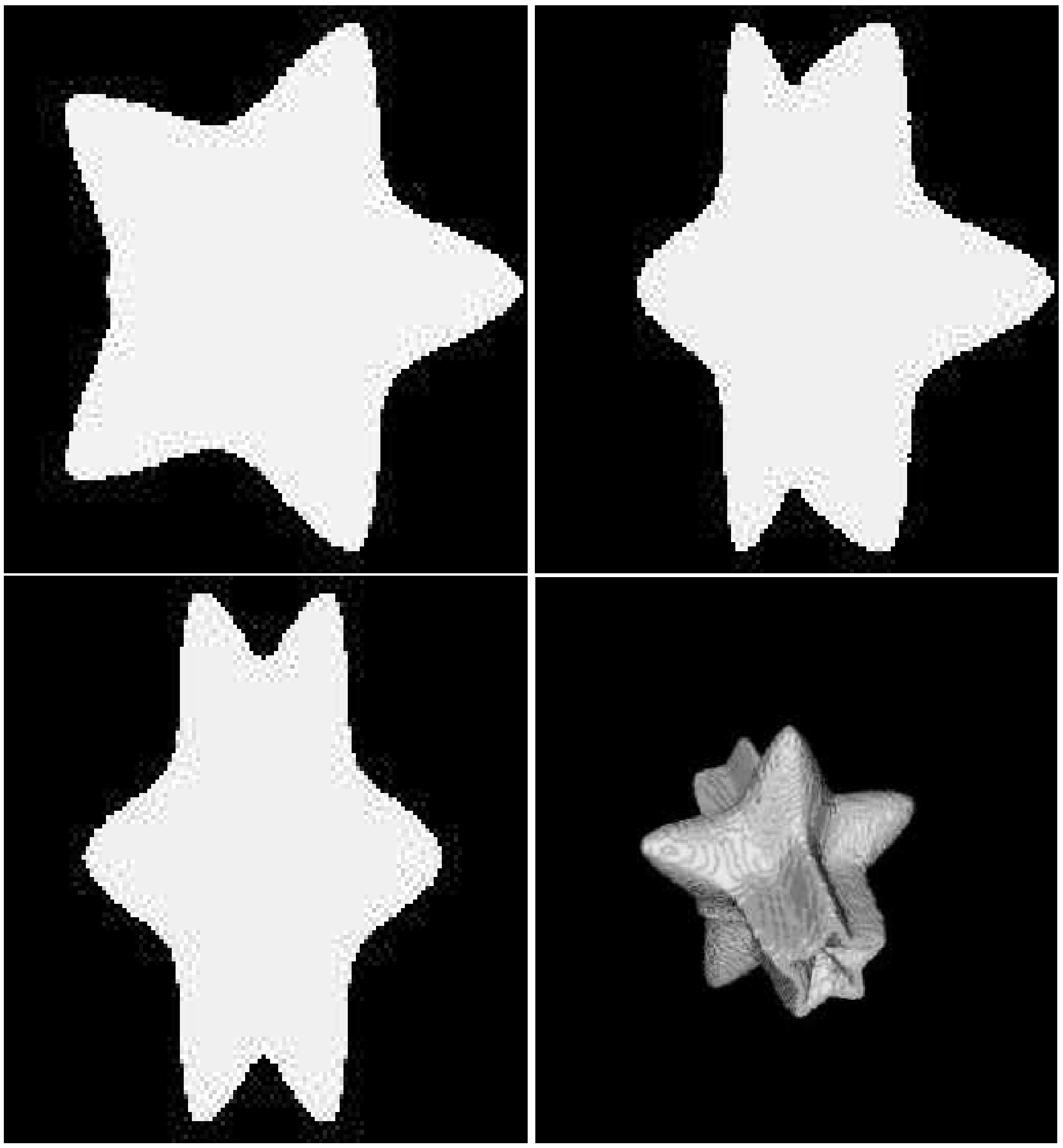

Figure 7.22: Shows 3 orthogonal slices and surface rendering of star. The “supershape” parameters are $a = 1$, $b = 1$, $m = 5.2$, $n_1 = 0.2$, $n_2 = 1.7$, and $n_3 = 1.7$.

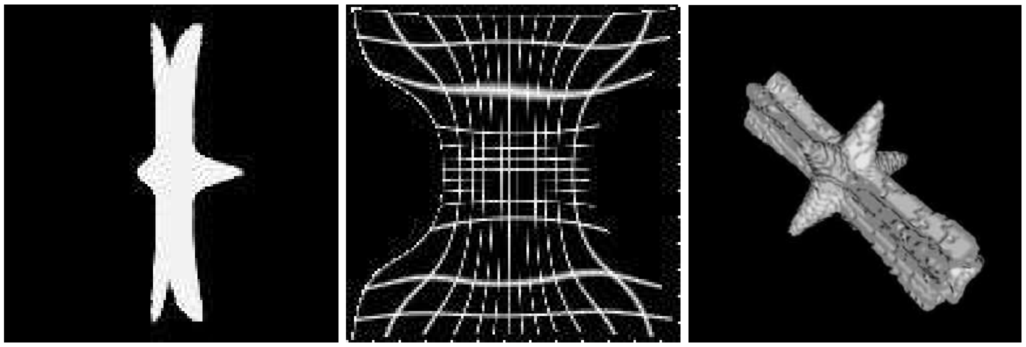

Figure 7.23: Result of star registered to flower with corresponding warped grid image as well as surface rendering of transformed image for a B-spline grid of 8×8×8 (total number of control points: 512).

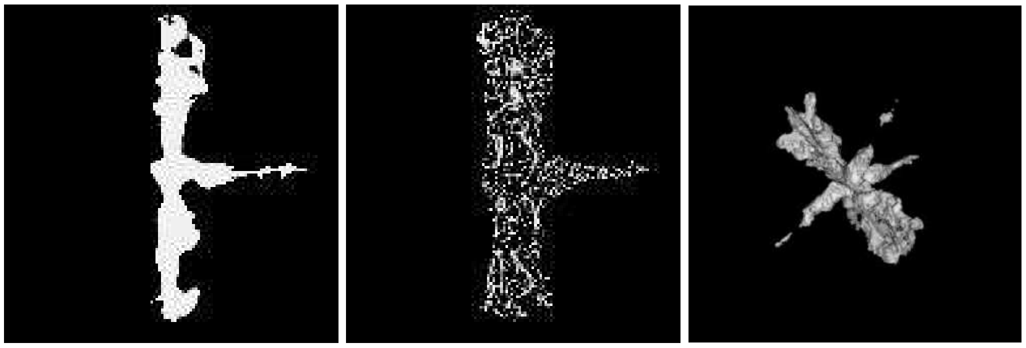

Figure 7.24: Result of star registered to flower with corresponding warped grid image as well as surface rendering of transformed image for an RBF transformation consisting of 512 points.

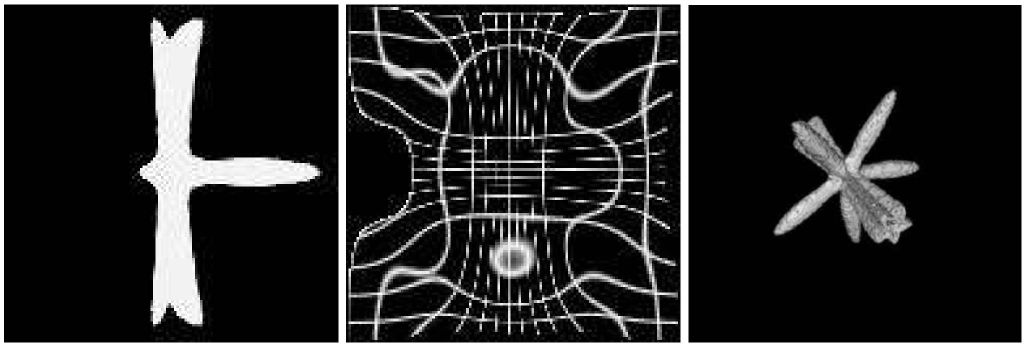

Figure 7.25: Result of star registered to flower with corresponding warped grid image as well as surface rendering of transformed image for a B-spline grid of 10×10×10 (total number of control points: 1000).

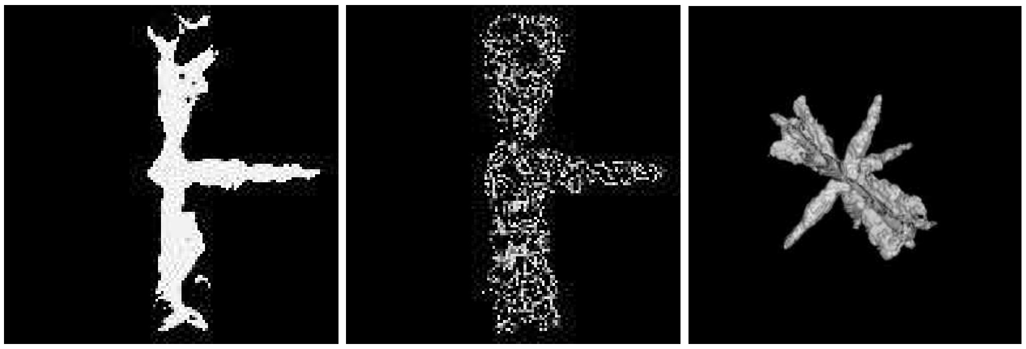

Figure 7.26: Result of star registered to flower with corresponding warped grid image as well as surface rendering of transformed image for an RBF transformation consisting of 1000 points.

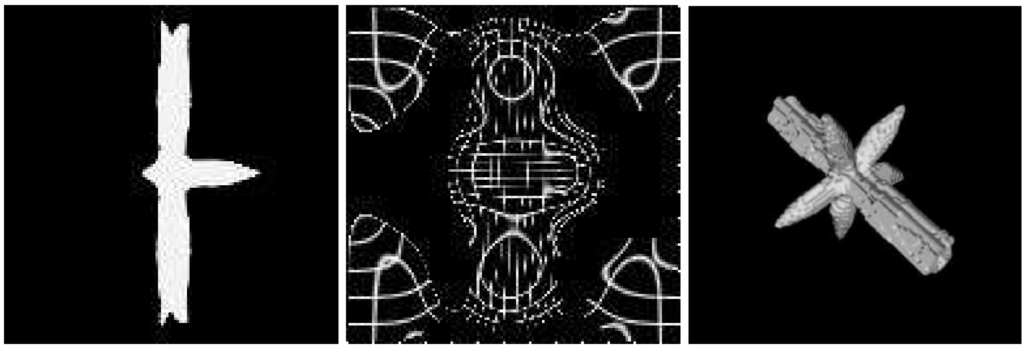

Figure 7.27: Result of star registered to flower with corresponding warped grid image as well as surface rendering of transformed image for a B-spline grid of $12\times12\times12$ (total number of control points: 1728).

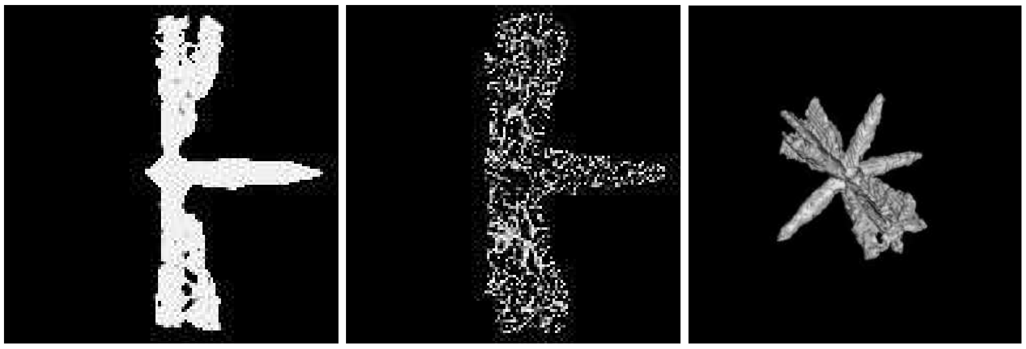

Figure 7.28: Result of star registered to flower with corresponding warped grid image for an RBF transformation consisting of 1728 points.

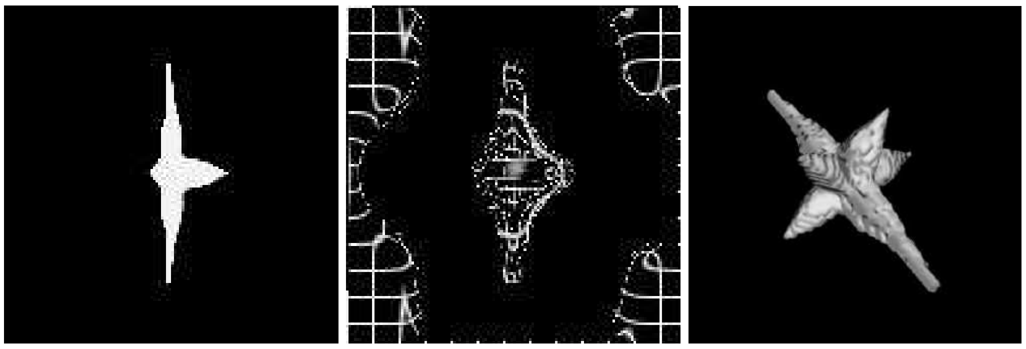

Figure 7.29: Result of star registered to flower with corresponding warped grid image as well as surface rendering of transformed image for a B-spline grid of 16×16×16 (total number of control points: 4096).

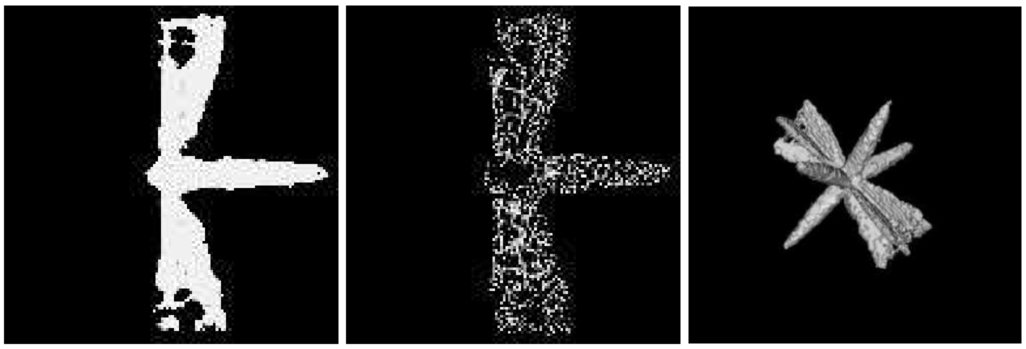

Figure 7.30: Result of star registered to flower with corresponding warped grid image as well as surface rendering of transformed image for an RBF transformation consisting of 4096 points.

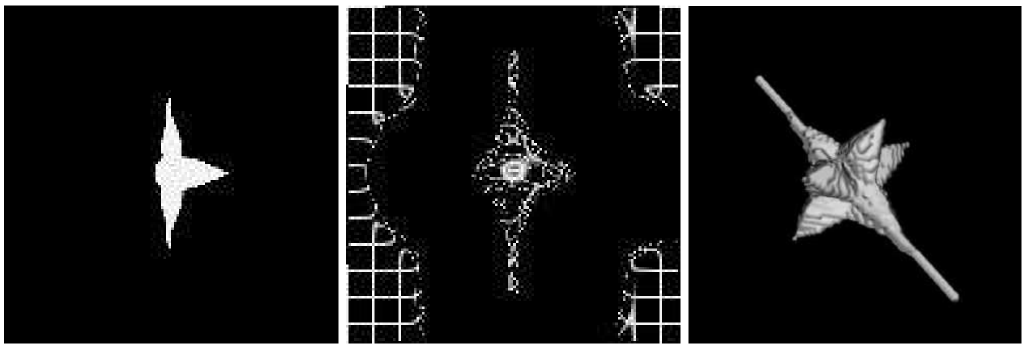

Figure 7.31: Result of star registered to flower with corresponding warped grid image as well as surface rendering of transformed image for a B-spline grid of 20×20×20 (total number of control points: 8000).

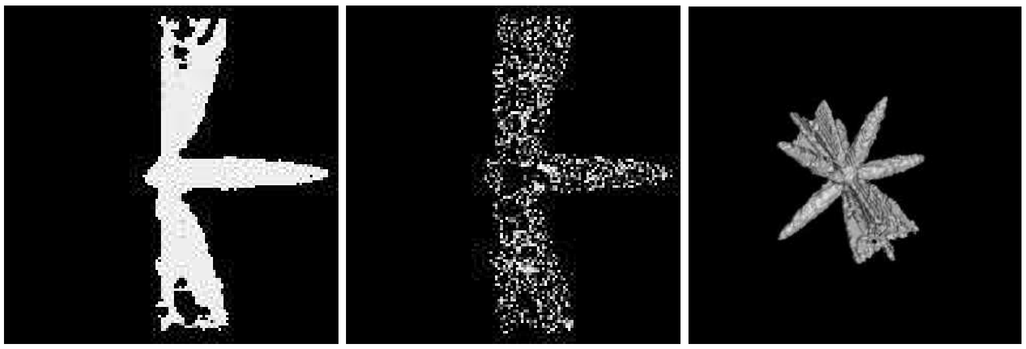

Figure 7.32: Result of star registered to flower with corresponding warped grid image as well as surface rendering of transformed image for an RBF transformation consisting of 8000 points.

the database. They were skull-stripped and cropped to dimensions 138$\times$147$\times$116 and 141$\times$152$\times$126, respectively.

In this section, in addition to experiments with the original IBSR images, we also describe experiments with added noise, with the RBF control points placed on a regular grid, and with different degrees of the FMM approximating polynomials.

7.3.1 Comparison to Affine and B-Spline

We evaluate the performance of our FMM with AD method by comparison to an affine registration and a B-spline registration. For the RBF, we used a thin plate spline (multiquadric with $\tau = 0$, $k = 1$), and the degree of the polynomials used in the fast multipole calculations was 5 (i.e. $p = 4$ so that $p + k = 5$). The number of iterations used in the GMRES solver was 40. For the optimization, we used the conjugate gradient method (Press et al., 1992). The points were randomly placed on the nonzero voxels of the gradient magnitude image. Figures 7.34 through 7.43 show the overlap calculations for various affine, B-spline, and RBF registrations. Table 7.9 shows the meaning of the letters in these figures. In addition, tables 7.10-7.13 show timing information and other values. The meaning of these values is identical to those of table 7.4. Note that prior to the affine registration, the number of misclassified voxels (NMV) was 426085, and after the affine registration, the the number of misclassified voxels was 408122.

We see that mutual information gives better results than sum of squared differences, as can be seen by comparing figures 7.34- 7.38 to figures 7.39-7.43. This is especially noticeable by the smaller brain structures such as the thalamus, caudate, putamen, pallidum, hippocampus, and amygdala (E-J) where the SSD gets increasingly worse as we increase the number of control points. The mutual information, by contrast, does not have this problem.

7.3.2 Registration Evaluation with Added Noise

The next experiment was the same as the previous RBF registration with 8000 points except that noise was added to the images as shown in figure 7.44. This will allow us to assess the robustness of our algorithm to noise. As shown in figures 7.45-7.46, we see that favorable results are obtained even in the presence of noise though not as good as without noise.

7.3.3 RBF Registration with Control Points Placed on a Grid

To help better compare RBFs to B-splines, we performed an RBF based registration with control points placed on a regular grid, like a B-spline, rather than irregularly on nonzero values of the gradient magnitude image. This will help us see if it is the meshless nature of RBFs which is important or is it simply the RBFs themselves which account for their success. The next experiment was the same as the previous 8000 point RBF registration without noise except that control points were placed on an $18 \times 20 \times 23$ grid (8280 points). As shown in figures 7.47-7.48, we see that the results placed on a grid are similar to results placed adaptively on nonzero gradient points, though most of the structures did slightly worse than the adaptive registration while some of the structures did slightly better. Based on these experiments, the advantage of the flexibility of control point placement for human brain images is not clear. However, further work on the optimal placement of control points is necessary to fully answer this question.

7.3.4 Comparison of RBF Registration with and without the FMM

We now describe experiments comparing how an RBF based registration that uses the FMM compares to one that does not. While the accuracy of the FMM is well understood, and is related to the polynomial degree $p + k$ (Cherrie et al., 2002). However, the use of this approximation in a gradient-based optimization needs to be examined. In particular, we are interested in finding out if the computed gradients (from the automatic differentiation) are the same or close in value. We therefore ran two sets of experiments with a sum of squared differences metric using the same images from the IBSR database as before with 8000 control points, one without the FMM and one with the FMM using several different polynomial degrees ($k = 1$ and p was set to the values shown in Table 7.14; see the discussion immediately following Eq. (4.10)). Table 7.14 shows the results. In the second column of the table, the average relative difference,

$$\frac{1}{24000} \sum_{i=1}^{24000} \frac{|a_i - b_i|}{|a_i|}, \tag{7.5}$$

is used to measure the accuracy, where a_i are the gradient components without FMM and b_i are the gradient components with FMM. We see that we get better agreement between both methods as we increase p, as would be expected, especially once p becomes 4 or higher. Additionally, while this experiment only analyzed the initial evaluation of the objective function in the optimization, we found that when we ran the full optimization for values of 4 or higher, there was no significant difference in the overlap measure (Eq. 7.1) for the brain structures. For these reasons, we chose a value of 4 for the experiments in this chapter.

7.4 Discussion

Given the above synthetic and real image experiments, it would appear that it is still too premature to strongly advocate the use of RBFs over B-splines in general at this time. The experiments, however, indicate that RBFs generally perform as well or better than B-splines and that in certain situations may improve performance. We have shown the feasibility of these powerful computational mechanisms for registration and believe

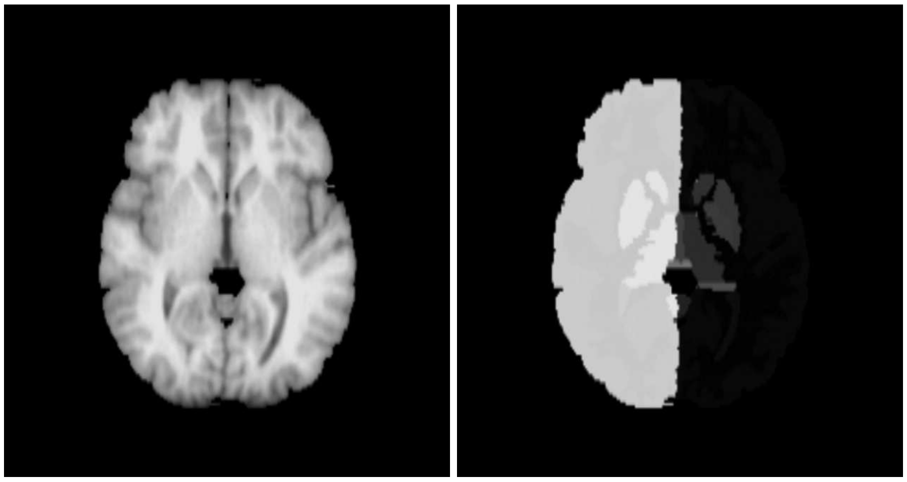

Figure 7.33: Shows a (skull stripped) slice of an image from the IBSR database along with its corresponding labeled image.

that, in spite of the computational overhead, they will prove to be useful in registration and elsewhere.

Letter	Structure	Groupings
A	All	2,3,4,5,7,8,10,11,12,13,14, 15,16,17,18,24,26,28,41,42,43,44, 46,47,49,50,51,52,53,54,58,60
B	Brain	2,3,7,8,10,11,12,13,16,17,18, 26,28,41,42,46,47,49,50,51, 52,53,54,58,60
C	Cortex	3,17,18,42,53,54
D	White Matter	2,41
E	Thalamus	10,49
F	Caudate	11,50
G	Putamen	12,51
H	Pallidum	13,52
I	Hippocampus	17,53
J	Amygdala	18,54
K	Subcort	10,11,12,13,26,49,50,51,52,58
L	Lateral Ventricle	4,43
M	CSF	4,5,14,15,24,43,44
N	Cerebellum	7,8,46,47
O	Brain Stem	16,28,60

Table 7.9: Key to brain labels.

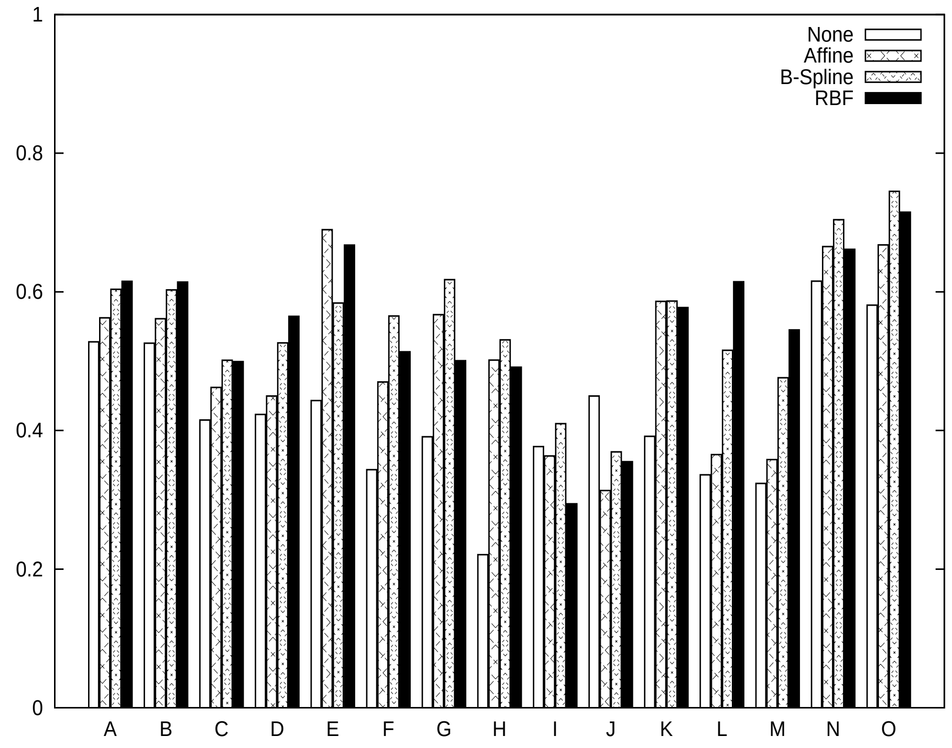

Figure 7.34: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, a B-spline registration with a grid size of $9\times10\times11$, and an RBF registration of 1000 points, all using an SSD metric.

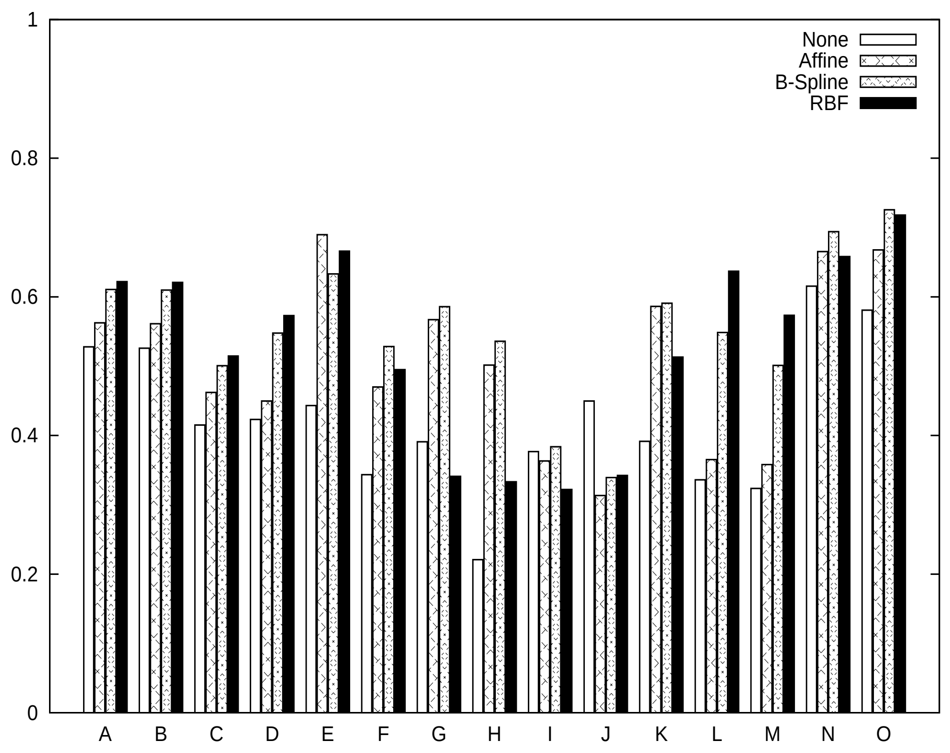

Figure 7.35: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, a B-spline registration with a grid size of 12×12×14, and an RBF registration of 2000 points, all using an SSD metric.

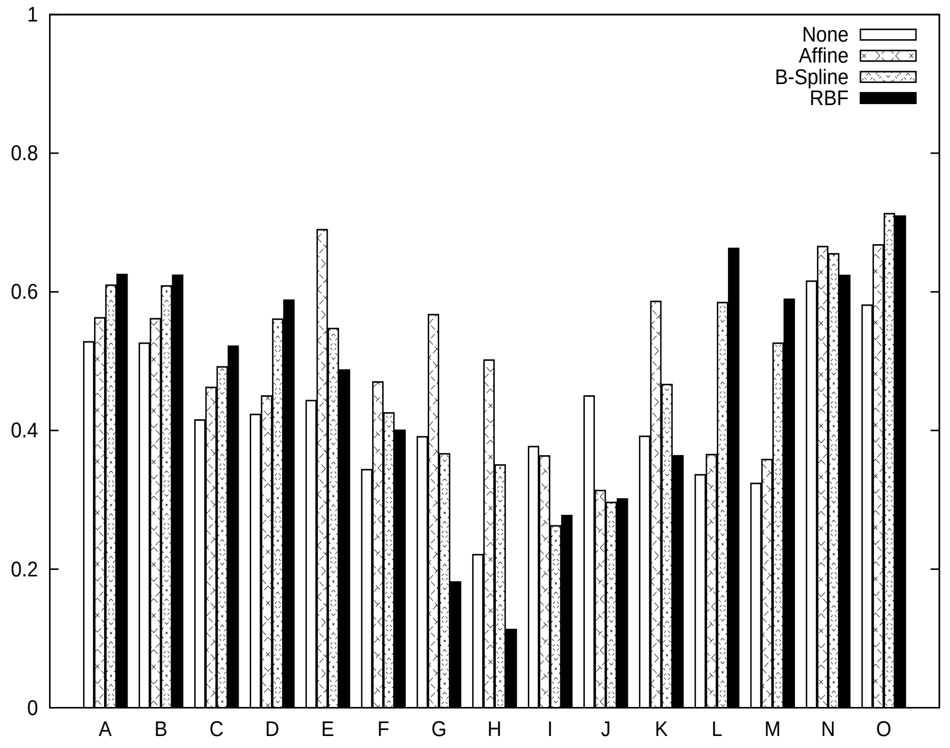

Figure 7.36: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, a B-spline registration with a grid size of $15 \times 15 \times 18$, and an RBF registration of 4000 points, all using an SSD metric.

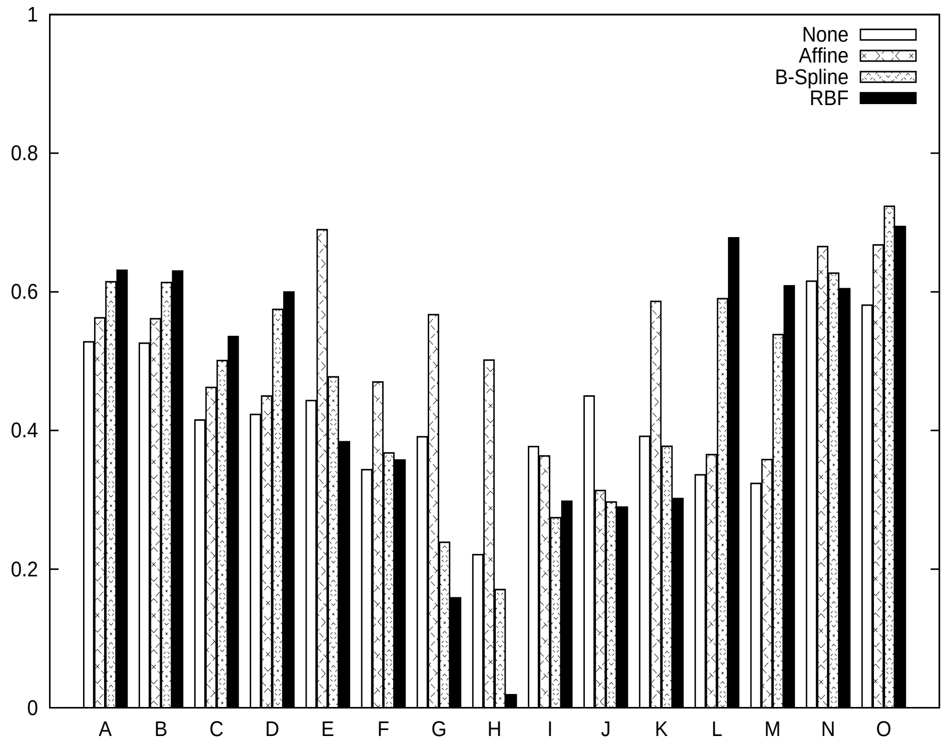

Figure 7.37: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, a B-spline registration with a grid size of $18\times20\times23$, and an RBF registration of 8000 points, all using an SSD metric.

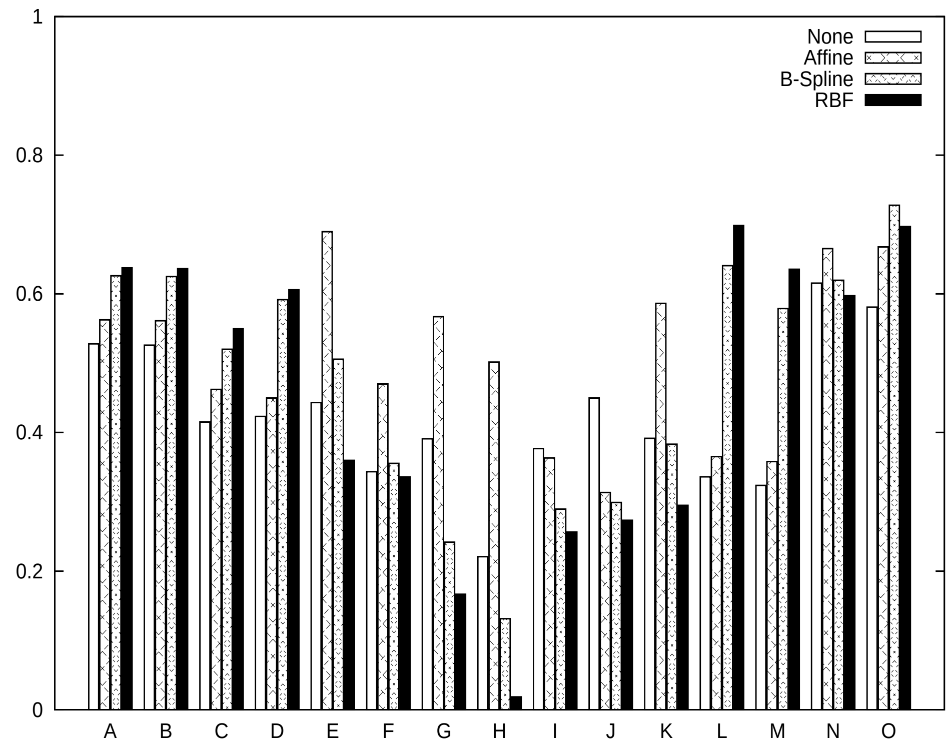

Figure 7.38: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, a B-spline registration with a grid size of $23\times25\times28$, and an RBF registration of 16000 points, all using an SSD metric.

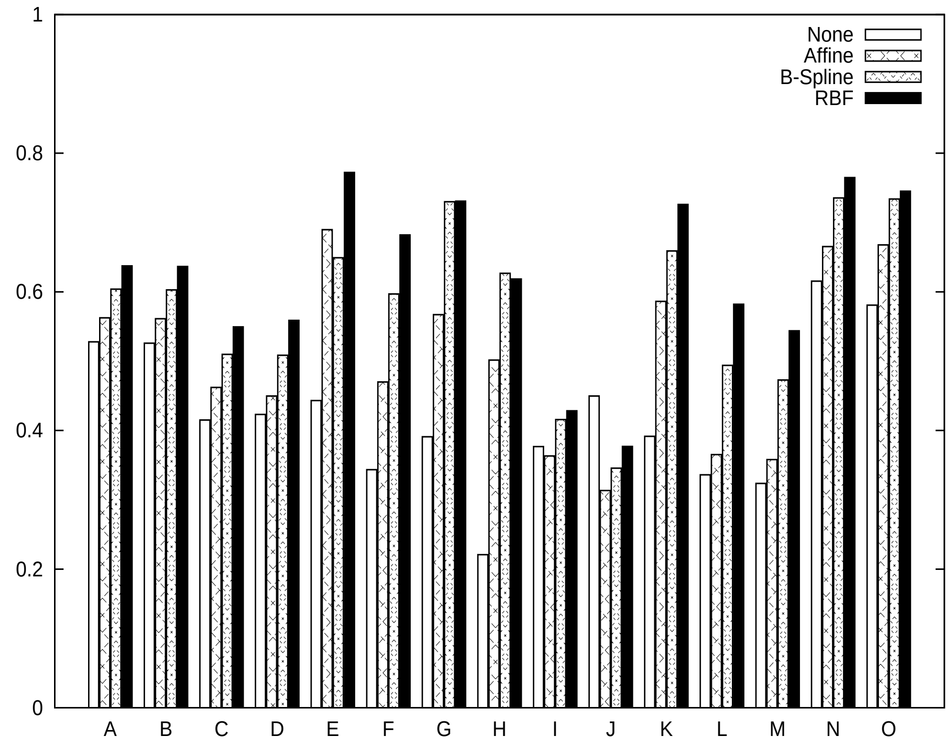

Figure 7.39: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, a B-spline registration with a grid size of 9×10×11, and an RBF registration of 1000 points, all using an MI metric.

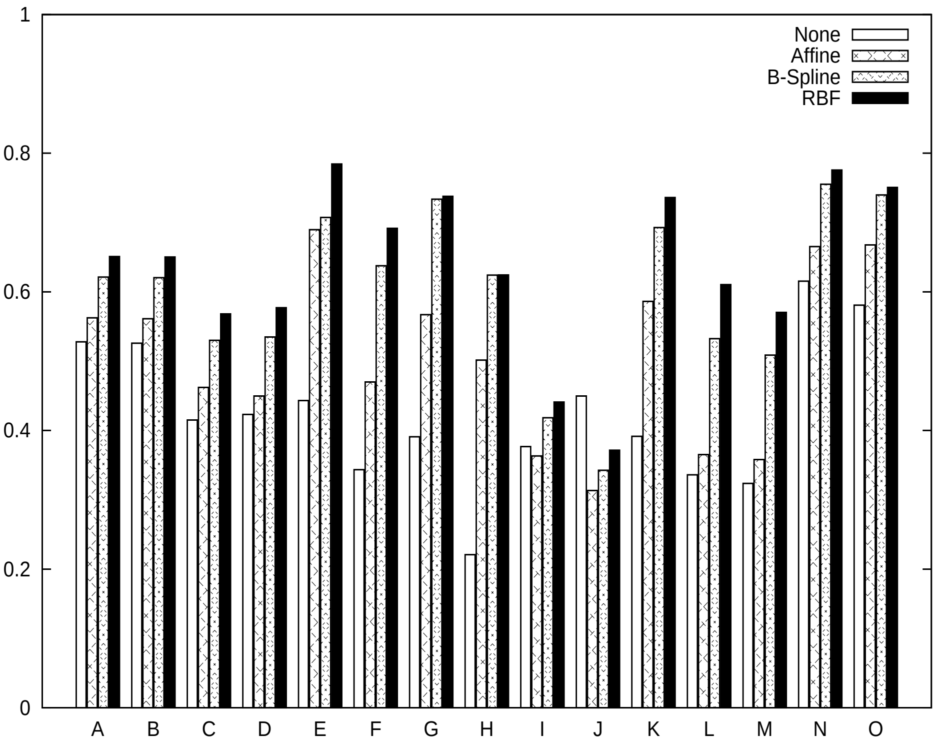

Figure 7.40: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, a B-spline registration with a grid size of $12\times12\times14$, and an RBF registration of 2000 points, all using an MI metric.

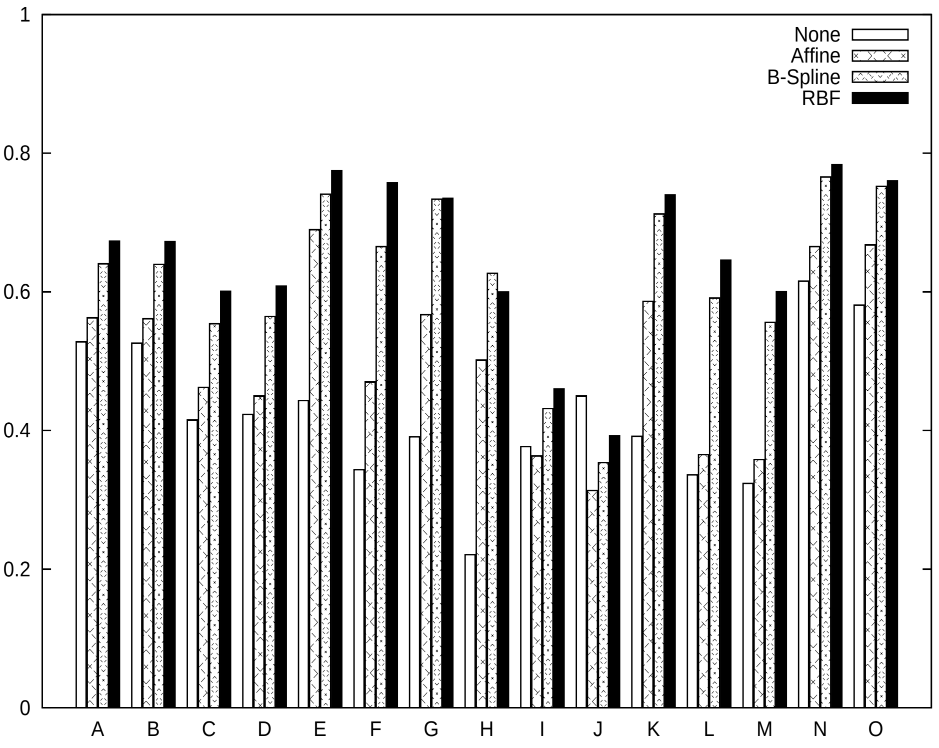

Figure 7.41: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, a B-spline registration with a grid size of $15\times15\times18$, and an RBF registration of 4000 points, all using an MI metric.

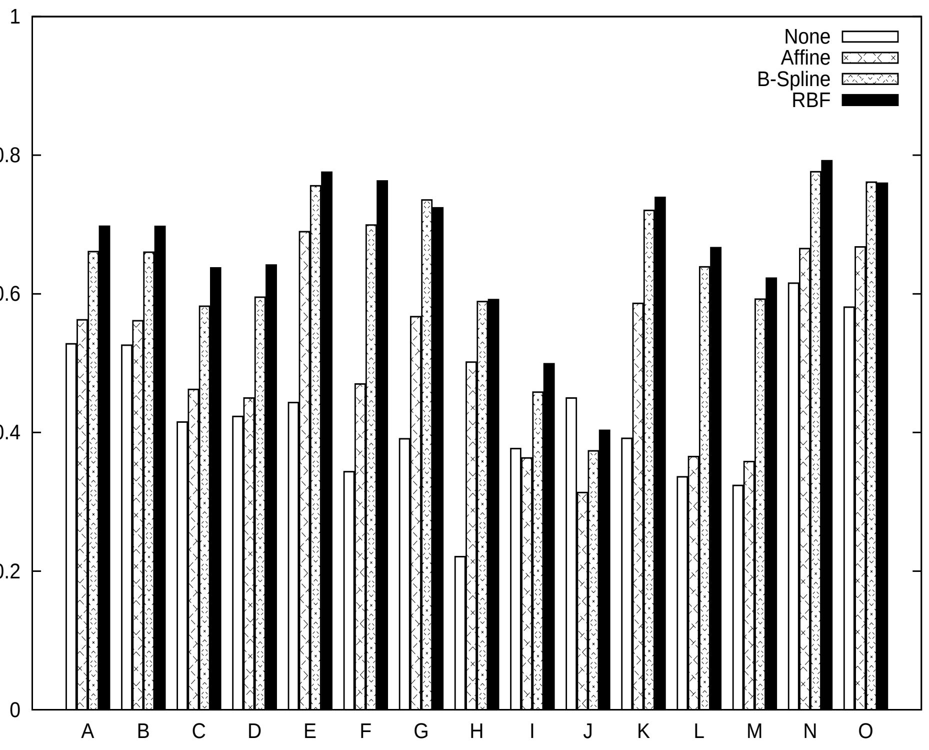

Figure 7.42: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, a B-spline registration with a grid size of $18\times20\times23$, and an RBF registration of 8000 points, all using an MI metric.

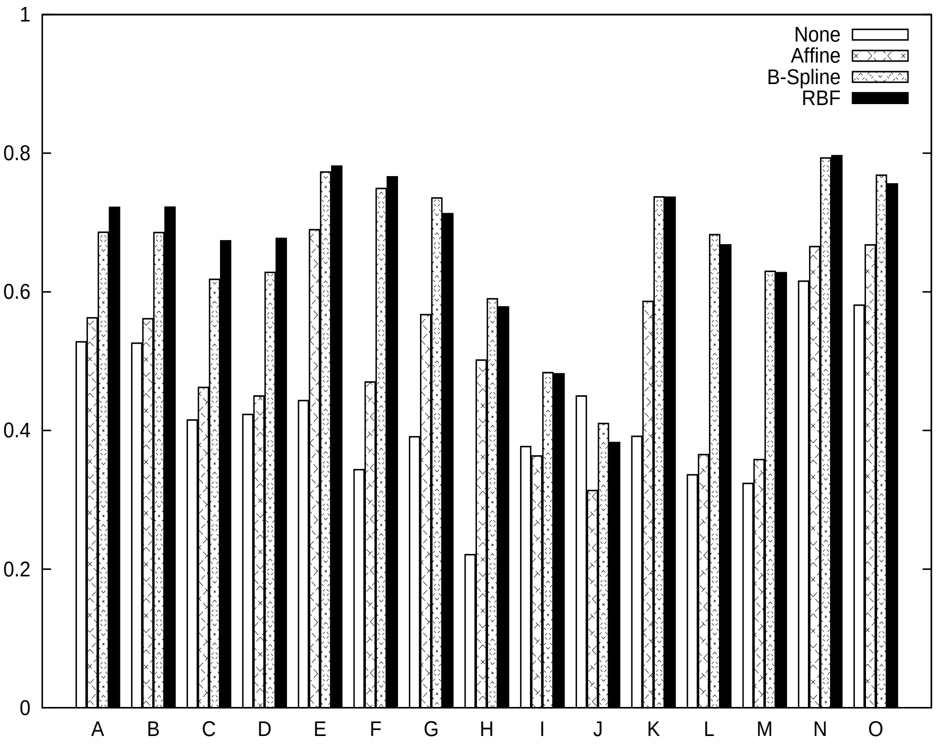

Figure 7.43: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, a B-spline registration with a grid size of 23×25×28, and an RBF registration of 16000 points, all using an MI metric.

Grid Size	9×10×11	12×12×14	15×15×18	18×20×23	23×25×28
Time-func (sec)	18.4	18.4	19.1	17.9	17.98
Time-grad (sec)	104.4	106.3	109.5	110.4	111.9
Ratio	5.69	5.78	5.74	6.17	6.23
Iterations	10	10	10	10	10
NMV	350161	336669	331172	320941	302968

Table 7.10: Running times and other values for an IBSR brain registration with a B-spline transformation and an SSD metric. See text for explanation of values.

Number of Points	1000	2000	4000	8000	16000
Time-func (sec)	84.2	185.5	458.96	1292.95	1391.07
Time-grad (sec)	531.09	1221.27	3116.72	9326.9	9632.09
Ratio	6.3	6.58	6.79	7.21	6.92
Iterations	10	10	10	10	10
NMV	323439	312686	303395	291562	279996
Time steps	5216	5468	5972	6980	8996
Checkpoints	20	20	20	20	20

Table 7.11: Running times and other values for an IBSR brain registration with an RBF transformation and an SSD metric. See text for explanation of values.

Grid Size	9×10×11	12×12×14	15×15×18	18×20×23	23×25×28
Time-func (sec)	40.8888	36.6	41.1666	39.333	37.227
Time-grad (sec)	287.727	266	297.9	288.09	274.18
Ratio	7.0368	7.2678	7.2364	7.3249	7.3651
Iterations	10	10	10	10	10
NMV	350161	336669	331172	320941	302968
Time steps	4710	4710	4710	4710	4710
Checkpoints	20	20	20	20	20

Table 7.12: Running times and other values for an IBSR brain registration with a B-spline transformation and an MI metric. See text for explanation of values.

Number of Points	1000	2000	4000	8000	16000
Time-func (sec)	69.265	147.879	478.9126	720	1332.504
Time-grad (sec)	434.5	955.36	3247.9	5985	9643.636
Ratio	6.27	6.46	6.78	8.3125	7.2373
Iterations	10	10	10	10	10
NMV	323439	312686	303395	291562	279996
Time steps	5216	5468	5972	6980	8996
Checkpoints	20	20	20	20	20

Table 7.13: Running times and other values for an IBSR brain registration with an RBF transformation and an MI metric. See text for explanation of values.

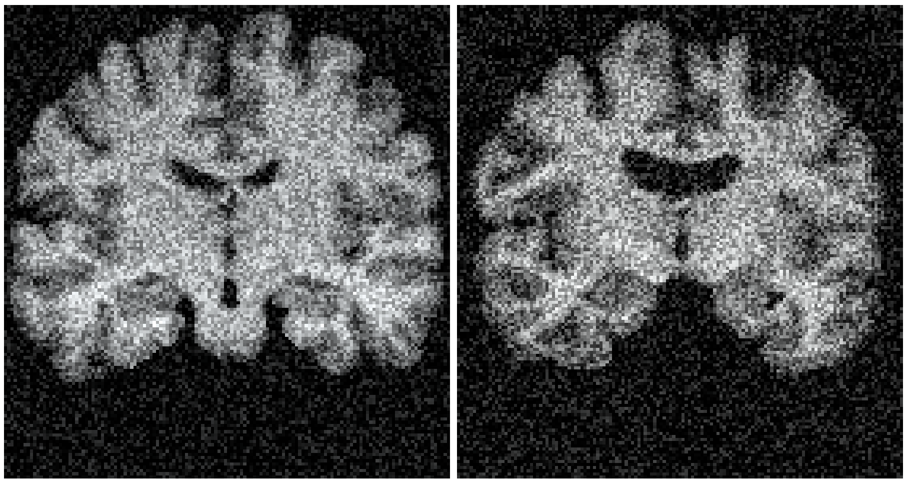

Figure 7.44: Slices of images from the IBSR database with added noise.

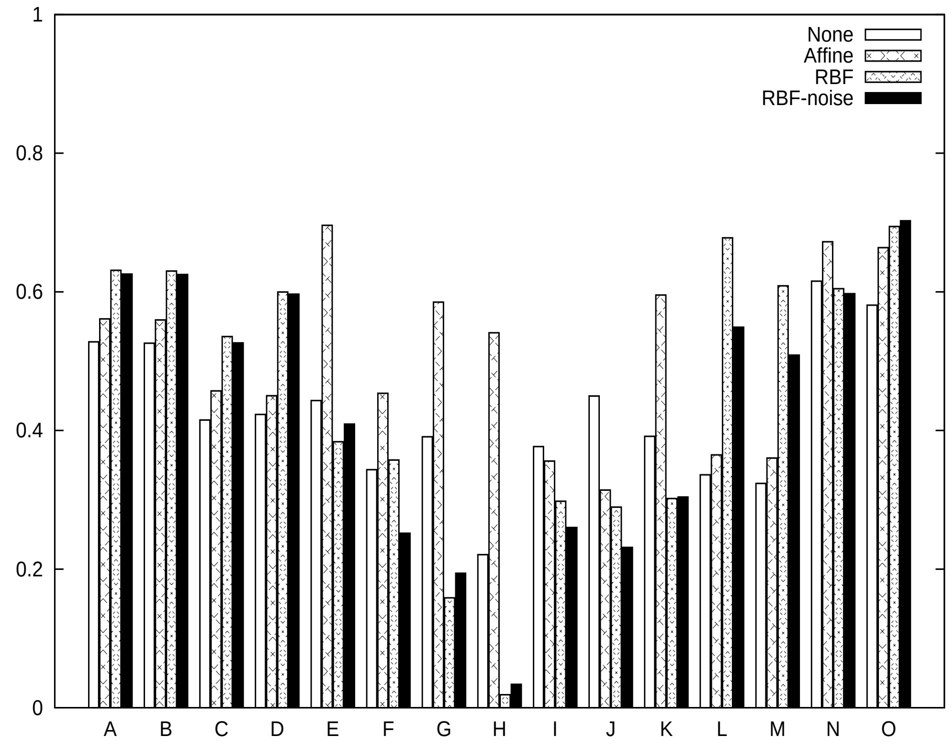

Figure 7.45: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, an RBF registration of 8000 points without noise, and an RBF registration of 8000 points with noise, all using an SSD metric.

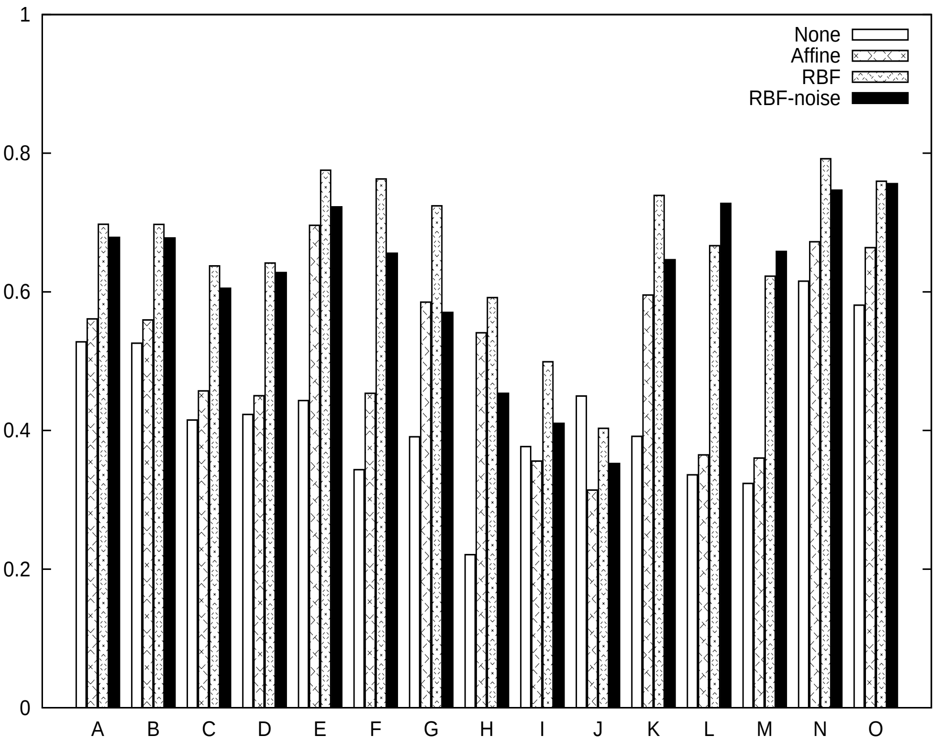

Figure 7.46: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, an RBF registration of 8000 points without noise, and an RBF registration of 8000 points with noise, all using an MI metric.

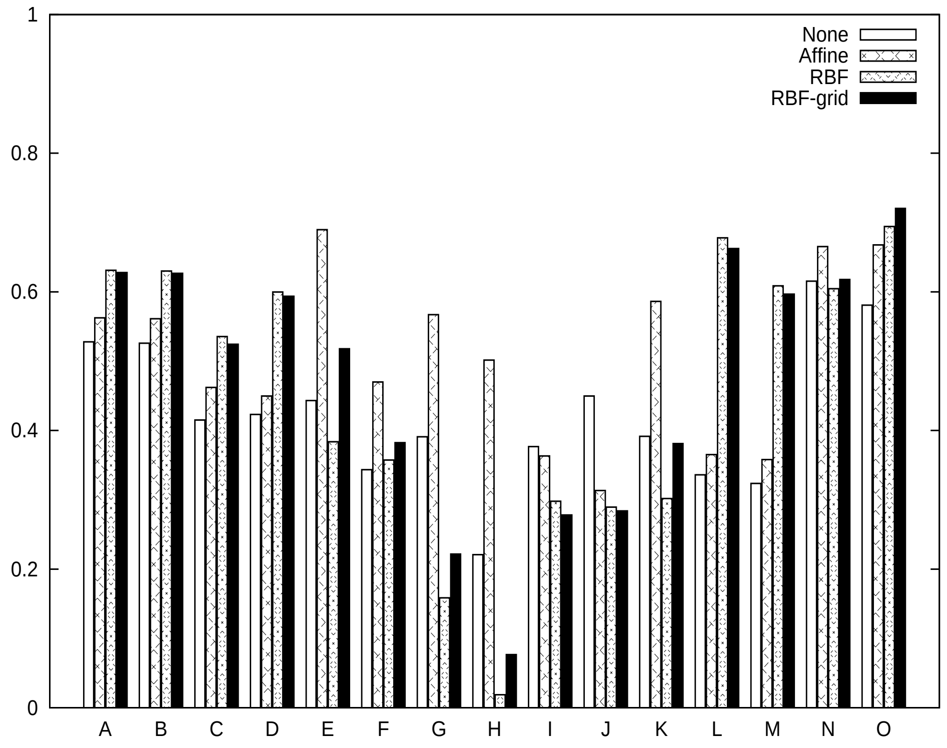

Figure 7.47: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, an RBF registration of 8000 points distributed on the gradient image, and an RBF registration with points placed on an $18\times20\times23$ grid (8280 points), all using an SSD metric.

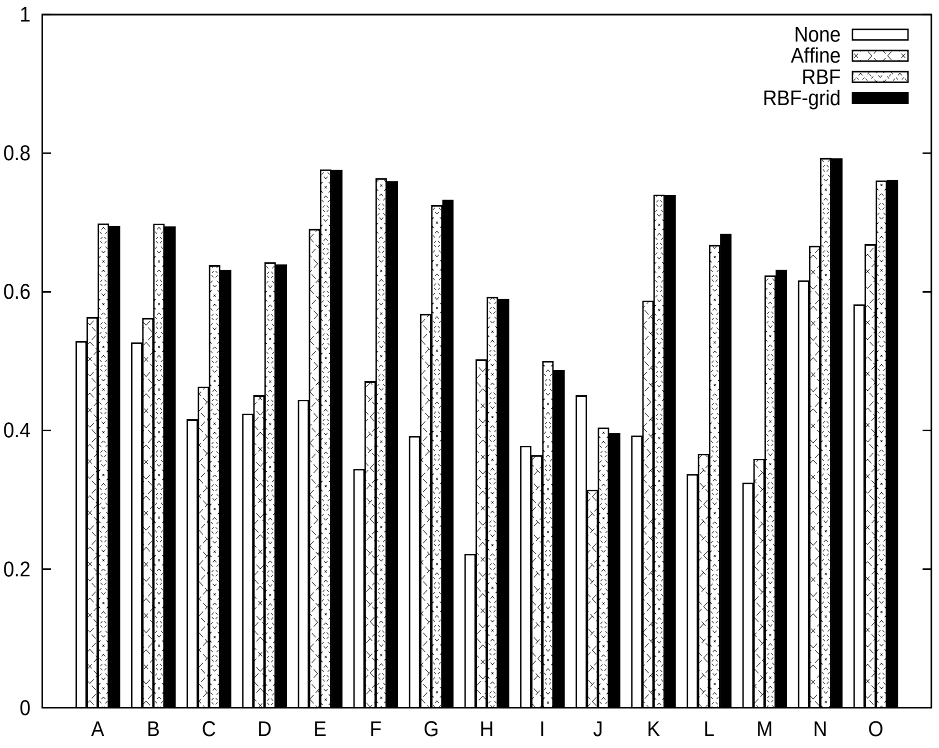

Figure 7.48: Shows the fractional overlap, Eq. (7.1), for labeled brain structures using no registration, an affine registration, an RBF registration of 8000 points distributed on the gradient image, and an RBF registration with points placed on an 18×20×23 grid (8280 points), all using an MI metric.

	Average Relative Error, Eq. 7.5
p=2	8.1299
p=3	0.603192
p=4	0.148773
p=5	0.0422957
p=6	0.0189292
p=7	0.0102788

Table 7.14: Registration accuracy with and without the FMM.

Chapter 8

Conclusion

This work presented a new system for meshfree nonrigid registration based on radial basis functions, automatic differentiation, and the fast multipole method. The motivation for using RBFs in this work was their meshfree nature and excellent smoothness properties. Our goal was to develop better numerical strategies for dealing with them in registration problems and thereby demonstrate their superiority to grid based methods in their ability to handle a wider class of images, especially 3D MRI brain images. Although more work remains in order to fully substantiate this claim, nevertheless, the *methodology* described in this work is new to the field of image registration, and we believe that it can be of use to other image analysis problems as well.

In addition, there are many other issues remaining for further research. For instance, one important area which we have not addressed in this work is that of the incorporation of constraints into the registration. This thesis was mainly concerned with intensity based registration. However, because of the meshfree nature of the transformation and the fact that constraints can be placed anywhere, constraints can therefore also be placed anywhere in space. Since the independent variables are the displacements of the control points, we can constrain the displacements and thereby restrict the motion of certain control to a specific areas.

Another issue we have not addressed is that of parallelism. Both the fast multipole method and the checkpointing method are readily parallelizable and we have not exploited this in our work. Recently, processor technology has hit a brick wall in terms of speed improvements, and, instead, manufactures are starting to pack multiple processors on a single chip. The average workstation soon may have numerous processors on one chip. Hence, parallelism will become increasingly important as it goes mainstream in the coming years, and algorithms will be required to make use of this parallelism to achieve better performance. Developing registration algorithms that exploit parallelism may thus be a fruitful avenue of research.

Furthermore, more advances and developments have been made in relation to solving the linear system associated with the RBFs as well as the FMM. For instance, domain decompositions methods have been used for solving the linear system (Beatson et al., 2001b). Although more complex than GMRES, better results have been reported with them (Beatson et al., 2001b). Also kernel independent methods have been developed (Beatson and Newsam, 1998; Ying et al., 2004, 2003), thus potentially increasing the class of RBFs that can be used. More work remains in incorporating these ideas into registration algorithms.

Yet another avenue of further research is higher order derivatives, needed, for example, in Newton-type optimization. In this work, we only considered first order derivatives, namely gradients. Unfortunately, unlike the cheap gradient rule, there is no such rule for higher order derivatives such as Hessians. But, as pointed out by Griewank (2000), computing every single component of the Hessian is usually not needed for most problems. However, deciding what parts of the Hessian to compute requires exploiting features unique to registration metrics, an endeavor which may yield interesting results.

Bibliography

Bajcsy, R. and Kovacic, S. (1989). Multiresolution elastic matching. *Comp. Vision Graphics Image Proc.*, 46:1–21.

Bankman, I., editor (2000). *Handbook of Medical Imaging: Processing and Analysis*. Academic Press, San Diego, CA.

Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and der Vorst, H. V. (1994). *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA.

Beatson, R. and Greengard, L. (1997). A short course on fast multipole methods. In Ainsworth, M. et al., editors, *Wavelets, multilevel methods and elliptic PDEs*, Numerical mathematics and scientific computation, pages 1–37. Oxford University Press, Oxford.

Beatson, R. K., Cherrie, J. B., and Mouat, C. T. (1999). Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration. *Advances in Computational Mathematics*, 11:253–270.

Beatson, R. K., Cherrie, J. B., and Ragozin, D. L. (2001a). Fast evaluation of radial basis functions: Methods for four-dimensional polyharmonic splines. *SIAM Journal on Mathematical Analysis*, 32(6):1272–1310.

Beatson, R. K. and Light, W. A. (1997). Fast evaluation of radial basis functions: methods for two-dimensional polyharmonic splines. *IMA Journal of Numerical Analysis*, 17(3):343–372.

Beatson, R. K., Light, W. A., and Billings, S. (2001b). Fast solution of the radial basis function interpolation equations: Domain decomposition methods. *SIAM Journal on Scientific Computing*, 22(5):1717–1740.

Beatson, R. K. and Newsam, G. N. (1998). Fast evaluation of radial basis functions: Moment-based methods. *SIAM Journal on Scientific Computing*, 19(5):1428–1449.

Bischof, C. H., Carle, A., Corliss, G. F., Griewank, A., and Hovland, P. (1992). ADIFOR: Generating derivative code from Fortran programs. *Scientific Programming*, 1:11–29.

Bischof, C. H., Carle, A., Khademi, P. M., and Mauer, A. (1994). The ADIFOR 2.0 system for the automatic differentiation of Fortran 77 programs. Technical Report MCS–P481–1194, Argonne, IL.

Bookstein, F. L. (1989). Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Trans. Pattern Anal. Machine Intell.*, 11(6):567–585.

Bourke, P. (2003). Supershape in 3D, `http://astronomy.swin.edu.au/~pbourke/surfaces/supershape3d/`.

Brown, D., Ling, L., Kansa, E., and Levesley, J. (2005). On approximate cardinal preconditioning methods for solving PDEs with radial basis functions. *Engineering Analysis with Boundary Elements*, 29(4):343–353.

Brown, L. G. (1992). A survey of image registration techniques. *ACM Comp. Surveys*, 24(4):325–376.

Buhmann, M. (2000). A new class of radial basis functions with compact support. *Mathematics of Computation*, 70(233):307–318.

Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. Y. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(6):1190–1208.

Canny, J. F. (1986). A computational approach to edge-detection. *IEEE Trans. Pattern Anal. Machine Intell.*, 8:679–698.

Cherrie, J. B., Beatson, R. K., and Newsam, G. N. (2002). Fast evaluation of radial basis functions: Methods for generalized multiquadrics in $\mathbf{R}^n$. *SIAM Journal on Scientific Computing*, 23(5):1549–1571.

Christensen, G. E. (1994). *Deformable Shape Models for Anatomy*. PhD thesis, Department of Electrical Engineering, Sever Institute of Technology, Washington University, St. Louis, MO.

Christensen, G. E., Miller, M. I., and Vannier, M. W. (1996). Individualizing neuroanatomical atlases using a massively parallel computer. *Computer*, pages 32–38.

Christensen, G. E., Rabbitt, R. D., and Miller, M. I. (1993). A deformable neuroanatomy textbook based on viscous fluid mechanics. In Prince, J. and Runolfsson, T., editors, *Proc. Conf. Information Sci. and Syst.*, pages 211–216.

Chui, H. (2001). *Non-Rigid Point Matching: Algorithms, Extensions and Applications*. PhD thesis, Yale University, New Haven, CT.

Collignon, A., Maes, F., Delaere, D., Vandermeulen, D., Suetens, P., and Marchal, G. (1995). Automated multi-modality image registration based on information theory. In Bizais, Y., Barillot, C., and Paola, R. D., editors, *Information Proc. Med. Imaging*, pages 263–274. Kluwer, Dordrecht.

Collins, D. L., Goualher, G. L., and Evans, A. C. (1998). Non-linear cerebral registration with sulcal constraints. In *Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention*, volume 1496 of *Lecture Notes in Computer Science*, pages 974–984. Springer, Heidelberg.

Coquillart, S. (1990). Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 187–196. ACM Press, New York.

Crum, W. R., Hartkens, T., and Hill, D. (2004a). Non-rigid image registration: theory and practice. *Br. J. Radiol.*, 77(suppl 2):S140–153.

Crum, W. R., Rueckert, D., Jenkinson, M., Kennedy, D., and Smith, S. M. (2004b). A framework for detailed objective comparison of non-rigid registration algorithms in neuroimaging. In Barillot, C., Haynor, D. R., and Hellier, P., editors, *Proceedings of the Seventh International Conference on Medical Image Computing and Computer-Assisted Intervention*, volume 3216 of *Lecture Notes in Computer Science*, pages 679–686. Springer, Heidelberg.

do Carmo, M. P. (1976). *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Upper Saddle River, NJ.

Dongarra, J. and Sullivan, F. (2000). Guest editors' introduction: The top 10 algorithms. *Computing in Science and Engg.*, 2(1):22–23.

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification*. Wiley-Interscience Publication, New York.

Fitzpatrick, J. M., Hill, D. L. G., and C. R. Maurer, J. (2000). Image registration. In Sonka, M. and Fitzpatrick, J. M., editors, *Handbook of Medical Imaging: Medical Image Processing and Analysis*, volume 2. SPIE Press, Bellingham, WA.

Fornefett, M., Rohr, K., and Stiehl, H. (2001). Radial basis functions with compact support for elastic registration of medical images. *Image and Vision Computing*, 19(1):87–96.

Franke, R. (1982). Scattered data interpolation: tests of some methods. *Math. Comput.*, 38(157):181–200.

Fujimura, K. and Makarov, M. (1998). Foldover-free image warping. *Graphical Models and Image Processing*, 60(2):100–111.

Greengard, L. (1987). *The Rapid Evaluation of Potential Fields in Particle Systems*. Ph.d. thesis, Yale University, New Haven, CT.

Greengard, L. (1988). *The Rapid Evolution of Potential Fields in Particle Systems*. MIT Press, Cambridge, MA.

Greengard, L. and Rokhlin, V. (1987). A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348.

Greengard, L. and Rokhlin, V. (1988). The rapid evaluation of potential fields in three dimensions. In Anderson, C. and Greengard, C., editors, *Vortex Methods*, Lecture Notes in Mathematics. Springer, New York.

Griewank, A. (1992). Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, 1:35–54.

Griewank, A. (2000). *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, PA.

Griewank, A., Juedes, D., and Utke, J. (1996). Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software*, 22(2):131–167.

Griewank, A. and Walther, A. (2000). Algorithm 799: Revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software*, 26(1):19–45.

Hajnal, J., Hawkes, D. J., and Hill, D. (2001). *Medical Image Registration*. CRC Press, Boca Raton, FL.

Hardy, R. L. (1990). Theory and applications of the the multiquadric-biharmonic method. *Comput. Math. Appl.*, 19(8/9):163–208.

Hart, F. P., Kriplani, N., Luniya, S. R., Christoffersen, C. E., and Steer, M. B. (2005). Streamlined circuit device model development with fREEDA and ADOL-C. In Bücker, H. M., Corliss, G., Hovland, P., Naumann, U., and Norris, B., editors, *Automatic Differentiation: Applications, Theory, and Implementations*, Lecture Notes in Computational Science and Engineering. Springer, New York.

Hellier, P., Barillot, C., Corouge, I., Gibaud, B., Le Goualher, G., Collins, D. L., Evans, A., Malandain, G., Ayache, N., Christensen, G. E., and Johnson, H. J. (2003). Retrospective evaluation of inter-subject brain registration. *IEEE Trans. Med. Imaging*, 22(9):1120–1130.

Hill, D. L. G., Batchelor, P. G., Holden, M., and Hawkes, D. J. (2001). Medical image registration. *Physics in Medicine and Biology*, 46:R1–R45.

Ibanez, L. and Schroeder, W. (2003). *The ITK Software Guide: The Insight Segmentation and Registration Toolkit*. Kitware, Inc., Albany, NY, `http://www.itk.org`.

IBSR (2004). Internet Brain Segmentation Repository. The MR brain data sets and their manual segmentations were provided by the Center for Morphometric Analysis at Massachusetts General Hospital and are available at `http://www.cma.mgh.harvard.edu/ibsr/`.

Kim, J. G., Hunke, E. C., and Lipscomb, W. H. (2006). A sensitivity-enhanced simulation approach for community climat e system model. In Alexandrov, V. N., van Albada, G. D., Sloot, P. M. A., and Dongarra, J., editors, *Computational Science – ICCS 2006*, volume 3994 of *Lecture Notes in Computer Science*, pages 533–540. Springer, Heidelberg.

Lea, D. J., Allen, M. R., Haine, T. W. N., and Hansen, J. (2002). Sensitivity analysis of the climate of a chaotic ocean circulation model. *Q. J. R. Meteorol. Soc.*, 128(586):2587–2605.

Lee, S., Wolberg, G., Chwa, K., and Shin, S. Y. (1996). Image metamorphosis with scattered feature constraints. *IEEE Trans. Visualization Comp. Graphics*, 2(4):337–354.

Lester, H. and Arridge, S. (1999). A survey of hierarchical non-linear medical image registration. *Pattern Recognition*, 32:129–149.

Liu, D. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528.

Maintz, J. and Viergever, M. (1998). A survey of medical image registration. *Medical Image Analysis*, 2(1):1–36.

Mattes, D., Haynor, D. R., Vesselle, H., Lewellen, T. K., and Eubank, W. (2003). PET-CT image registration in the chest using free-form deformations. *IEEE Trans. Med. Imag.*, 22(1):120–128.

Mauer-Oats, A. (1997). Checkpoint, `http://www.math.northwestern.edu/~amauer/projects/ checkpoint/`.

Maurer, C. R. and Fitzpatrick, J. M. (1993). A review of medical image registration. In Macuinas, R. J., editor, *Interactive Image-Guided Neurosurgery*, pages 17–44. Am. Assoc. Neurological Surgeons, Parkridge, IL.

Mount, D. M. and Arya, S. (2005). ANN: A library for approximate nearest neighbor searching, `http://www.cs.umd.edu/˜mount/ANN/`.

Papademetris, X., Staib, L. H., Jackowski, A. P., Win, L. Y., Schultz, R. T., and Duncan, J. S. (2004). Integrating intensity and feature nonrigid registration. In Barillot, C., Haynor, D. R., and Hellier, P., editors, *Proceedings of the Seventh International Conference on Medical Image Computing and Computer-Assisted Intervention*, volume 3216 of *Lecture Notes in Computer Science*, pages 763–770. Springer, Heidelberg.

Pennec, X. and Thirion, J. (1995). Validation of 3-D registration methods based on points and frames. In *Proc. Fifth Int. Conf. Comp. Vision*, pages 557–562.

Pluim, J., Maintz, J., and Viergever, M. (2003). Mutual-information-based registration of medical images: a survey. *IEEE Trans. Med. Imaging*, 22(8):986–1004.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York.

Rohde, G. K., Aldroubi, A., and Dawant, B. M. (2003). The adaptive bases algorithm for intensity-based nonrigid image registration. *IEEE Trans. Med. Imag.*, 22(11):1470–1479.

Rohr, K., Stiehl, H., Sprengel, R., Buzug, T., Weese, J., and Kuhn, M. (2001). Landmark-based elastic registration using approximating thin-plate splines. *IEEE Trans. on Medical Imaging*, 20(6):526–534.

Rueckert, D., Sonoda, L., Hayes, C., Hill, D., Leach, M., and Hawkes, D. (1999). Nonrigid registration using free-form deformations: application to breast MR images. *IEEE Trans. Med. Imaging*, 18(8):712–721.

Saad, Y. and Schultz, H. (1986). GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM. J. Sci. Stat. Comput.*, 7(3):856–869.

Schnabel, J. A., Tanner, C., Castellano-Smith, A. D., Leach, M. O., Hayes, C., Degenhard, A., Hose, R., Hill, D. L. G., and Hawkes, D. J. (2001). Validation of non-rigid registration using finite element methods. In *Information Proc. Med. Imaging*, volume 2082 of *Lecture Notes in Computer Science*, pages 344–357. Springer, Heidelberg.

Scholze, M., Rayner, P., Knorr, W., Kaminski, T., and Giering, R. (2002). A prototype Carbon Cycle Data Assimilation System (CCDAS): Inferring interannual variations of vegetation-atmosphere CO2 fluxes. Abstract CG62A-05. *Eos Trans. AGU*, 83(47).

Sederberg, T. W. and Parry, S. R. (1986). Free-form deformation of solid geometric models. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 151–160. ACM Press, New York.

Shapiro, V. and Tsukanov, I. (1999). Meshfree simulation of deforming domains. *Computer Aided Design*, 31(7):459–471.

Stevens, W. R. (1993). *Advanced Programming in the UNIX Environment*. Addison-Wesley, Reading, MA.

Strang, G. (1988). *Linear Algebra and Its Applications*. Harcourt, Inc., New York.

Strother, S. C., Anderson, J. R., Xu, X., Liow, J., Bonar, D., and Rottenberg, D. (1994). Quantitative comparisons of image registration techniques based on high-resolution MRI of the brain. *J. Comp. Assisted Tomogr.*, 18(6):954–962.

Thevenaz, P. and Unser, M. (2000). Optimization of mutual information for multiresolution image registration. *IEEE Transactions on Image Processing*, 9(12):2083–2099.

Tiddeman, B., Duffy, N., and Rabey, G. (2001). A general method for overlap control in image warping. *Computers and Graphics*, 25(1):59–66.

Toga, A. W. (1999). *Brain Warping*. Academic Press, San Diego, CA.

Tsukanov, I. and Shapiro, V. (2002). The architecture of SAGE - a meshfree system based on RFM. *Engineering with Computers*, 18(4):295–311.

Viola, P. and Wells, W. (1995). Alignment by maximization of mutual information. In *Proc. Fifth Int. Conf. Comp. Vision*, pages 16–23.

Wang, Y. and Staib, L. H. (2000). Physical model based non-rigid registration incorporating statistical shape information. *Med. Image Anal.*, 4:7–20.

Wang, Z., Navon, I. M., Zou, X., and Le Dimet, F.-X. (1995). A truncated Newton optimization algorithm in meteorology applications with analytic Hessian/vector products. *Computational Optimization and Applications*, 4:241–262.

Warfield, S. K., Rexilius, J., Huppi, P. S., Inder, T. E., Miller, E. G., William M. Wells, I., Zientara, G. P., Jolesz, F. A., and Kikinis, R. (2001). A binary entropy measure to assess nonrigid registration algorithms. In *Proceedings of the Fourth International Conference on Medical Image Computing and Computer-Assisted Intervention*, volume 2208 of *Lecture Notes in Computer Science*, pages 266–274. Springer, Heidelberg.

Wendland, H. (1995). Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4(1):389–396.

West, J., Fitzpatrick, J., Wang, M., Dawant, B., Maurer, C., Kessler, R., Maciunas, R., Barillot, C., Lemoine, D., Collignon, A., Maes, F., Suetens, P., Vandermeulen, D., van den Elsen, P., Hemler, P., Napel, S., Sumanaweera, T., Harkness, B., Hill, D., Studholme, C., Malandain, G., Pennec, X., Noz, M., Maguire, G., Pollack, M., Pelizzari, C., Robb, R., Hansen, D., and Woods, R. (1997). Comparison and evaluation of retrospective intermodality image registration techniques. *J. Comp. Assisted Tomogr.*, 21:554–566.

Wolberg, G. (1990). *Digital Image Warping*. IEEE Comp. Soc., Los Alamitos, CA.

Wu, Z. (1995). Multivariate compactly supported positive definite radial functions. *Adv. Comput. Math*, 4:283–292.

Ying, L., Biros, G., and Zorin, D. (2004). A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, 196(2):591–626.

Ying, L., Biros, G., Zorin, D., and Langston, H. (2003). A new parallel kernel-independent fast multipole method. In *SC ’03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 14. IEEE Computer Society, Washington, DC.

Zhang, X., Song, K., Lu, M., and Liu, X. (2000). Meshless methods based on collocation with radial basis functions. *Computational Mechanics*, 26(4):333–343.

Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. (1997). Algorithm 778. L-BFGS-B: Fortran subroutines for Large-Scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560.

Zitová, B. and Flusser, J. (2003). Image registration methods: A survey. *Image and Vision Computing*, 21:977–1000.