Christof Seiler
Supervisor: Mauricio Reyes, Ph.D.

Displacement Vector Field Regularization for Modelling of Soft Tissue Deformations

Master Thesis in Biomedical Engineering

Bern, June 10, 2008

University of Bern
Medical Faculty
MEM Research Center
Institute for Surgical Technology and Biomechanics

Acknowledgements

The thesis has been written as part of the Master program in Biomedical Engineering at the MEM Research Center, University of Bern, in collaboration with the University of Applied Sciences of Biel. I would like to express my thanks to all the people supporting me during this time.

First, I would like to thank my supervisor Mauricio Reyes of the MEM Research Center, who came up with the idea of modelling soft tissue deformations by using Markov Random Fields. I have enjoyed all the meetings we have had during the thesis; it was always very inspiring.

I am also very grateful for the support from Rasmus Paulsen of the Technical University of Denmark and Philippe Büchler of the MEM Research Center for their useful comments and valuable input.

I would like to thank Hansjörg Riedwyl, CEO of ISS AG, for giving me the opportunity to work part time during the thesis.

Special thanks to Harlad Studer, a fellow student and a very good friend, who was one to talk to during critical times.

Many thanks to Stefan Feissli, Christoph Studer and Jay Sungu for their moral support during the thesis.

Last but not least, I would like to express my thanks to my family, I am sure it was not always easy to listen to my complaints.

Contents

1

Introduction

Many methodologies dealing with prediction or simulation of soft tissue deformations on medical image data require preprocessing of the data in order to produce a different shape representation that complies with standard methodologies, such as mass-spring networks, Finite Element Methods (FEM), etc. On the other hand, methodologies working directly in the image space normally do not take into account mechanical behavior of tissues and tend to lack physics foundations driving soft tissue deformations. This thesis presents a method to simulate soft tissue deformations based on coupled concepts from image analysis and mechanics theory.

A methodology is presented to simulate soft tissue deformations in the image space that is based on a Maximum a Posteriori (MAP) model of the deformations, which considers novel developed energy terms to account for tissue material properties, boundary conditions and related confidence maps. Optimization is performed under a Markov Random Field (MRF) approach, which is further extended into a Hierarchical-MRF (HMRF) approach. The introduction of a HMRF approach enables to use fast local optimizers, as opposed to global optimizers which are computationally very expensive and in practice do not guarantee convergence. The hierarchical approach is also robust with respect to local minima [33].

Finally, MRF and HMRF are well suited for parallel implementation [25, 24], which led to an implementation on the Graphics Processor Unit (GPU) in addition to the CPU implementation.

An inspiration for the thesis was the work presented in [31]. The authors use Markov Random Field regularization to find point correspondence of shapes. This technique has then been applied to build a well-correlated statistical training data set and for the registration of 3D surfaces.

2

State of the Art

One of the current problems with non-rigid registration techniques is their lack of physics foundations concerning mechanical properties and energies driving the deformations. Conversely, classical methods to compute soft tissue deformations such as Finite Element Methods (FEM), Mass-spring networks, etc., suffer from the need of transforming the data into a volumetric mesh representation of the object. As a consequence, image segmentation and volumetric mesh generation are two inevitable steps to be performed. Whereas image segmentation can be performed semi- or fully automatically with some acceptable accuracy, automatic volumetric meshing algorithms are prone to errors and dependent on object topology, desired boundary conditions [4], etc. Therefore semi-manual techniques are often used, which is tedious and time consuming. Furthermore, jumping from the image space (i.e. voxels) to the shape space (i.e. mesh) involves approximations due to the discreet and different nature of both data representations [8]. In addition, voxel intensities have valuable information about material properties of the structure, which is required to be translated to the shape space (linear and non-linear interpolations, barycentric-based techniques, etc.) at the price of introducing further errors due to these approximations. The aforementioned limitations of these two approaches (mesh- and image-based techniques) are known by the scientific community and efforts have been made in this direction.

The next sections briefly introduce classical methodologies and previous approaches.

2.1 Finite Element Method

Finite Element Methods (FEM) are based on continuum mechanics. The basic concept of FEM models is to minimize the potential energy of a deformed object:

$$PE = U - W \tag{2.1}$$

where PE, U and W are potential energy, total external work and total internal work, respectively. External work is a result of displacing points of an object by applying forces. By applying external forces to an object the equilibrium state of that object is shifted, as a result the internal structure of an object reacts and adjusts to the new situation by internal material displacements. The equilibrium situation can be described as the minimum strain energy of the material, this requires solving a Partial Differential Equation (PDE). To be able to solve the PDE the object is divided into small elements joined by nodes. Depending on the integration type of these elements and the mesh density, the amount of points that are calculated differs. Regardless the integration type, the interpolation between points is computed with so called shape functions. The chosen order of the shape functions strongly depends on the application. FEM methods are very accurate but also very computational expensive. In Fig. 2.1 an example mesh of a simple

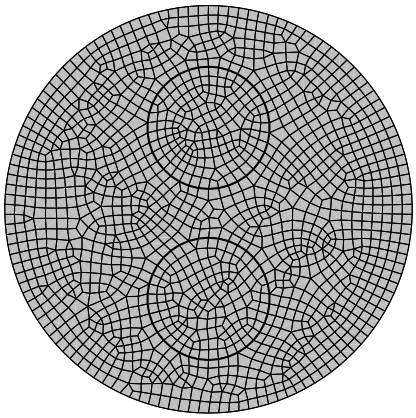


Fig. 2.1. Example of a 2D mesh with three regions divided in triangular and quadratic element types.

geometry is depicted. In simple geometries the meshing problem is usually solved with good quality whereas with more complex structures more complex methods and more crucial the need of user interaction are required. The next section describes some solutions to the problem of automatic meshing and points out the difficulties and limitations of these approaches.

2.1.1 Automatic Meshing

For example in [5] an automatic FEM meshing technique is proposed. The method generates an one-to-one representation of voxels into hexahedra elements and a posterior smoothing approach is applied to deal with jagged edges of the geometry. However, the high number of generated elements yield long computations making this technique only applicable to micro-CT images, where a high precision is needed, which can be achieved by having a direct voxel-to-element representation.

Another interesting approach for automated meshing can be found in [35], where a target specimen is meshed by morphing pre-existent geometries. Presented are two algorithms. The first algorithm is based on automated warping. The source and target geometries are warped onto an 'auxiliary' surface. The warping is governed by an energy minimization process. The second algorithm is based on manual landmarks, where manually placed landmarks produce a mapping of the source and target geometries. Between the landmark points, thin-plate splines were used for interpolation. In the presented work, predictions of deformation, strain and stress of the target vertebra under axial loading show only minor differences between morphed and individually generated models but the methods still require a considerable amount of manual work, and some trial and error to determine adequate control volumes and landmarks to guide the deformation.

2.2 Mass-Spring

Mass-spring models are defined by a discrete number of points. The points are connected to each other and form a web. The connections represent springs assigned with certain mechanical properties. Deformations are governed by Newton's law of motion. The displacements are given by

$$M\ddot{x} + C\dot{x} + Kx = f, \tag{2.2}$$

where M, C and K are the mass matrix of the object, the damping and stiffness matrix, respectively, x and f represent the vector of the coordinates of the points describing the

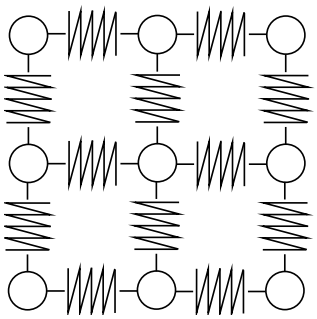


Fig. 2.2. Example of mass-spring model.

object and the vector of external forces applied to x. A common approximation used for real time simulations [12] is

$$Kx \approx f, \tag{2.3}$$

where the assumption that the deformation is very close to static is made, meaning the velocity of the deformation process is very low. In this case the derivatives will be zero for the damping and mass part of the equation. In Fig. 2.2 an example of a Mass-Spring model is illustrated.

2.2.1 Force Propagation

In [6] soft tissues are modeled as a mass-spring system and tissue deformation is simulated as a process of force propagation among the mass points on a per-node basis. When an external force is applied to a node, the force propagates from the point of contact, namely the stimulated node, to its neighboring nodes via the interconnecting springs. The process proceeds in an ordered manner from the nearest neighbors to the farthest ones, until constrained by the maximum penetration depth. The approach is computationally efficient and easy to implement since matrix formulations and operations are not needed. However, extra preprocessing work is required to determine the optimum penetration depth, which is also dependent on the topology of the deformable object. Furthermore, preliminary studies have shown that the deformation results of the proposed model deviate from that of the FEM when the external force is large.

2.3 Elastic Registration

Medical image registration has been a popular research topic for quite some time, detailed surveys are presented in [27], [21] and [42]. The goal of registration is to transform a source image into a target image by preserving relative image features. In the set of available registration methods, elastic methods are of special importance for the current work. Elastic registration does not use any parametric mapping functions like most other registration techniques do. In elastic registration the primary goal is not to search for the right parameters of mapping functions, but to find the right elastic deformation of the image. The images are viewed as pieces of rubber sheet and external forces are applied to bring them to match the target image, where the deformation of each piece is ruled by internal stiffness or smoothness constraints. To find the optimal deformation an hierarchical iterative approach is often chosen.

One example is [36], where the deformation field is modeled with B-splines. As mentioned above the drawback of conventional non-rigid registration methods is the lack of biomechanical information used to model the deformations.

2.4 Training-Based Methods

2.4.1 Combination of Statistical Registration and Finite Element Methods

In [29] an approach is presented for deformable registration of brain tumor images to a normal brain atlas, here the dissimilarity of the images hinders the usability of readily available deformable image registration techniques. Through statistical modelling of sought tumor-induced deformations the method combines a biomechanical model of tumor mass-effect and a deformable image registration technique.

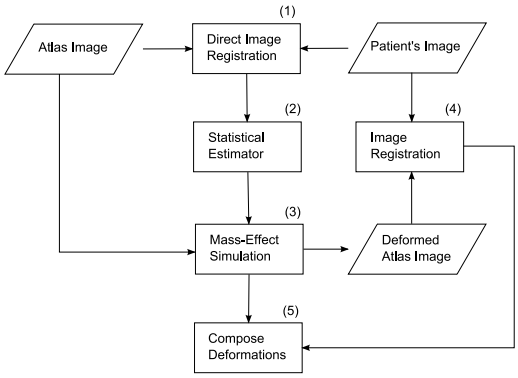


Fig. 2.3. Flow-chart of registration process.

In Fig. 2.3 a flow-chart with the basic steps involved in the described approach is presented. Step (1) is the registration of the atlas image and the patient‘s brain used as a rough estimate for step (2). In step (2) the estimate is used to define the parameters of the FEM simulations. The following parameters are used for the FEM model: the center and radii of the tumor, radii of the peri-tumor edema region (center is the same as for the tumor), and the outward pressure induced by the tumor in all directions. The parameters are estimated based on the statistical model established prior to the registration process. The statistical model is created by simulating a wide range of parameters applied to the atlas image. In step (3) the FEM simulation is executed. Step (4) is another image registration with the patient‘s image, but this time with the deformed atlas image. Finally in step (5) the deformed image after the FEM simulation and the registered image is combined.

The authors reported significant reduction in the registration error. Nonetheless, the reliability of the method depends on the statistical training performed on a set of FEM simulations using different parameters.

2.4.2 Segmentation and Tracking of Deformable Shapes

In [26] a model-based segmentation of a scene, a sequence of images containing a certain shape that is of interest, is obtained by applying a deformable model. The model consists of two components. First, global deformation modeled by rigid transformation (translation, rotation and scale) and non-rigid transformation controlling the shape variability of the object that is being tracked. Second, local deformation considered as random perturbations modeled by MRF. In this work the training of the shape variability is required.

2.4.3 Neural Networks

In [18] a method to model the deformation of elastic objects through the use of artificial neural networks is presented. This technique is useful for objects that have complex

material properties and geometry. The simulation can further be made directly in the image space. But the quality strongly depends on the quality of training data. To obtain the training data Boundary Element Method (BEM) are used, for which meshing is required.

2.5 Autowaves

A very interesting part of research is done in finding physical processes that show analogies to soft tissue deformation. For instance, in [41] a relationship between autowaves and soft tissue deformation has been established. Autowaves are signals that need no outside support to induce release of stored energy in an active medium, the released energy then triggers the same process in adjacent regions. The fundamental difference to classical waves is that once energy has been released, a certain time needs to pass until another release of energy can happen. The aim is to formulate soft tissue deformation as the propagation of autowaves. The method is designed to model 3D shapes that represent organs for real-time surgical simulation. The authors show that the methodology can handle nonlinear material properties and perform well for large-range displacements. A major drawback is the fact that only surfaces of 3D models are simulated at the moment. The authors plan to extend the methodology to 3D solid volumes.

3

Mathematical Foundations

In this chapter a basic introduction to the Bayesian paradigm applied to image analysis is presented. It is then extend to formulate image analysis problems by means of Markov Random Fields (MRF). This chapter is fundamental for the understanding of the model that has been built. In Chapter 4, the concept shown in this chapter will be adapted to vector field displacement regularization.

3.1 The Bayesian Paradigm

In this text the Bayesian paradigm in the field of image analysis is explained. In practice the ideal image x is always blurred and noisy. So y is a deterministic or random transformation $y = f(x)$ of the true scene x. To restore y to the original image x, an inversion of f needs to be found. But in general f is not invertible. One way is to use the Bayesian paradigm.

The formulation of the inversion problem in Bayesian terms allows to introduce the concept of statistics into the field of image analysis. The aim is to express prior knowledge about an image in terms of probabilities, e.g. prior knowledge can be an assumption about the regularity of pixel values. The prior knowledge is then combined with the observed image data, e.g. observed image y does not differ much from the original image x.

Probabilities are described with the help of acceptance functions, which assign acceptance values to pixel configurations. In the end, the configuration that has the best values is assumed to be the best possible approximation $\hat{x}$ of x.

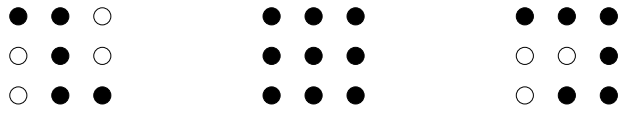
The goal is to build a statistical model that contains, on the one hand, prior knowledge about the original image and on the other hand, the relation between observed data and prior knowledge. To illustrate an example is presented.

Example 3.1 (Prior knowledge and correlation to observed data). In Fig. 3.1a an observed image y is shown. In Fig. 3.1b+c two hypothetical images g_1 and g_2 are depicted. The white and black dots represent two different pixel intensities. The aim is to evaluate which of the two hypothetical images is more likely to be the original image x. The evaluation is based on acceptance functions, which assign values to pixel configurations. The lower the value the more likely the pixel configuration. In this example the first function is defined as

$$R(g) = \sum_{s \sim t \in S} (1 - \delta(g_s, g_t)), \tag{3.1}$$

where S is the set of sites or pixels, δ is the Kronecker symbol which is define as

$$\delta(u, v) = \begin{cases} 1 & \text{if } u = v \\ 0 & \text{else,} \end{cases} \tag{3.2}$$



(a) (b) (c)

Fig. 3.1. (a) Observation data y, (b) first hypothetical image g_1 and second hypothetical image g_2

$s \sim t$ represent the set of neighboring sites in vertical and horizontal direction:

$$\text{(horizontal)} \quad s\bullet \quad t\bullet \qquad \text{(vertical)} \quad \begin{matrix} s\bullet \\ t\bullet, \end{matrix}$$

and g is a hypothetical image. The function R represents the regularity of pixel intensities and thus represents prior knowledge about the original image. Fig. 3.1b has no neighbor pixel that differ whereas Fig. 3.1c has 10. So in terms of R the image in Fig. 3.1b is more likely to be the original image than image in Fig. 3.1c.

Now the fidelity of the observation image is checked, or in other words the correlation between the hypothetical image and the observed image. In this constructed example two hypothetical images g_1 and g_2 are compared to the observed image y:

$$D(g, y) = \sum_{s \in S} (1 - \delta(g_s, y_s)). \tag{3.3}$$

For Fig. 3.1b D equals 4, because four pixel differ between observed and hypothetical image, and for Fig. 3.1c D equals 3. So image in Fig. 3.1c is more likely to be the original image.

Finally, the two acceptance functions are combined:

$$U(g) = \sum_{s \sim t \in S} R(g_s, g_t) + \sum_{s \in S} D(g_s, y_s). \tag{3.4}$$

where Fig. 3.1b has the score $0 + 4 = 4$ and Fig. 3.1c is $10 + 3 = 13$. So overall Fig. 3.1b is more likely to be the original image x.

3.2 Prior and Posterior Distributions

Let $\boldsymbol{X}$ be the set of all original images x and $\boldsymbol{Y}$ a set of all observed images y.

Now two probabilities are defined, for the sake of simplicity $P(X = x)$, where X is a subset of $\boldsymbol{X}$, in this case the image x, is written as $P(x)$ throughout the text. With the *prior probability* the degree to which an image $x \in \boldsymbol{X}$ fulfills the regularity conditions and constraints is measured as

$$P(x). \tag{3.5}$$

This means that each image $x \in \boldsymbol{X}$ is assigned a certain probability of being the original image. $P(x)$ depends on the image x only and not on y.

The *conditional probability*

$$P(y|x), \tag{3.6}$$

represents the probability of observing an image y when image x is the original image.

The combination of both probabilities can be done according to the Bayes theorem:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}, \tag{3.7}$$

since $P(\cdot|y)$ is an adjustment of $P(x)$ after the observation of y, it is called the *posterior probability*. $P(y)$ can also be written as

$$\sum_z P(x)P(z,y), \tag{3.8}$$

where $z \in \boldsymbol{X}$. Thus, to calculate $P(y)$ all probabilities of $\boldsymbol{X}$ need to be calculated, which in practice is not feasible due the huge configuration space $\boldsymbol{X}$. However, usually this is not needed because $P(y)$ does not depend on x and thus can be assumed constant. Therefore,

$$P(x|y) \propto P(y|x)P(x). \tag{3.9}$$

The final aim is to reach the maximum posterior probability $P(x|y)$ for a certain pixel observation y. This will be discussed in the next section.

3.3 Maximum a Posteriori

To reach the best configuration of pixels the posterior probability is maximized, which leads us to the maximum a posteriori (MAP) estimate:

$$\hat{x} = \arg\max_x P(x|y). \tag{3.10}$$

The probability distributions will have the Gibbsian form, therefore Gibbs distributions are explained in the next section.

3.4 Gibbs Distributions

The basic idea to use Gibbs distributions to calculate conditional probabilities is borrowed from statistical physics. The Gibbs distributions [15] are defined as follows

$$P(x) = \frac{\exp(-U(x))}{\sum_z \exp(-U(z))} = \frac{1}{Z}\exp(-U(x)), \tag{3.11}$$

where U, Z are the energy function and the partition function. The partial function is the total energy sum of all possible states of the physical system. In practice these values are usually not possible to calculate due to the vast configuration space, so it is approximated with a constant value. To decide on the best configuration no absolute probabilities are needed, it is sufficient to know relative probabilities and pick the greatest one, so by approximating the partial function Z the results are not influenced.

With the help of Gibbs distributions the MAP estimate of (3.9) can now be formulated in terms of energy functions:

$$P(x|y) \propto \frac{1}{Z}\exp(-U_{\text{observation}}(x,y) - U_{\text{prior}}(x)), \tag{3.12}$$

where U_{prior} corresponds to $P(x)$ and $U_{\text{observation}}$ to $P(x|y)$.

In the next sections it is shown how the minimization problem can be solved in terms of MRFs.

3.5 Neighborhood Systems and Cliques

To be able to calculate energy functions in an efficient way neighborhood systems are designed and solved through MRF. First some definitions of neighborhood system and cliques need to be made before we can jump into the formulation of MRF.

A neighborhood system in terms of MRF is define as:

Definition 3.2. *A collection $N_s = \{N_s : s \in S\}$ of sets is called a neighborhood system $\mathcal{N}$, if $s \notin N_s$ and $s \in N_t$ if and only if $t \in N_s$. The sites $t \in N_s$ are called neighbors of s.*

Definition 3.3. *In the image space neighborhood systems are of the following type: S is a finite lattice $\{(i,j) \in \mathbb{Z} \times \mathbb{Z} : -m \leq i,j \leq m\}$ and for some number c,*

$$N_{ij} = \{(k,l) : 0 < (k-i)^2 + (l-j)^2 \leq c\}.$$

Definition 3.4. *A subset C of S is a clique if any two different elements of C are neighbors. The set of cliques will be denoted by $\mathcal{C}$.*

To illustrate the definitions two examples are given.

Example 3.5 (Neighborhood system $c = 1$). A neighborhood system $c = 1$ is shown in

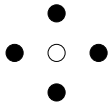


Fig. 3.2. Neighborhood system for $c = 1$

Fig. 3.2. The set of cliques $\mathcal{C}$ for the $c = 1$ neighborhood system is depicted in 3.3.

Fig. 3.3. Clique types for neighborhood system $c = 1$

Example 3.6 (Neighborhood system $c = 2$). The neighborhood system for $c = 2$ is shown

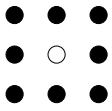


Fig. 3.4. Neighborhood system for $c = 2$

in Fig. 3.4, where the corresponding $\mathcal{C}$ is given in 3.5.

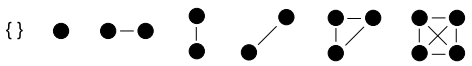


Fig. 3.5. Clique types for neighborhood system $c = 2$ (the rotations are not listed)

3.6 Markov Random Fields

First, general random fields will be explained before the special case of MRF. More detailed information can be found in [39, 15].

Random fields are defined on a finite index set S with elements s called sites. Random variables are associated to each element in S. A random variable can take any state of $\boldsymbol{X_s}$, where s is the spatial location within $\boldsymbol{X}$. In this text random variables represent the pixel intensity with a certain probability, consider P(x) the prior probability. A random field is then

$$\boldsymbol{X} = \prod_{s \in S} \boldsymbol{X_s}, \tag{3.13}$$

for $P(x) > 0$ for every $x \in \boldsymbol{X}$.

The MRF differs in the way that probabilities are only defined on a neighborhood system. The neighborhood Markov property is expressed as

$$P(x_s | x_t, t \neq s) = P(x_s | x_t, t \in N_s) \forall s \in S, \tag{3.14}$$

where N_s is a neighborhood system at site s, x_s and x_t are pixel values at site s and t, respectively. It means that the probability distribution of each site only depends on the state of neighboring sites.

Gibbs distributions provide the means to calculate the MAP in terms of functions. To be able to calculate these distributions based on neighborhood pixel configuration the correlation between Gibbs distributions and Markov Random Fields (MRF) is necessary. The relation between the Gibbs distribution and Markov Random Fields are specified in the following theorem

Theorem 3.7. *(Hammersley-Clifford). Let $\mathcal{N}$ be a neighborhood system. Then $\boldsymbol{X}$ is a Markov random field with respect to $\mathcal{N}$ if and only if $P(x)$ is a Gibbs distribution with respect to $\mathcal{N}$.*

A proof can be found in [39, 13].

So for a given neighborhood system $\mathcal{N} = \{N_s : s \in S\}$ energy functions in terms of Gibbs distributions need do be defined. The energy functions are a sum of locally defined potential functions:

$$U_{\text{total}} = \sum_{C \in \mathcal{C}} V_C, \tag{3.15}$$

where $\mathcal{C}$ and U_{total} are the set of cliques defined on $\mathcal{N}$ and the energy function, respectively.

By combining (3.12), (3.14) and (3.15) it all boils down to

$$P(x|y) \propto exp(-U_{\text{total}}(x, y)), \tag{3.16}$$

where $U_{\text{total}} = \alpha U_{\text{prior}} + (1 - \alpha) U_{\text{observation}}$ and α is a weighting parameter.

The MAP $P(x|y)$ can now be realized by the minimization of energy functions U_{total} that are only defined on a local neighborhood system. In the following sections an energy function is defined based on potential functions defined for cliques.

3.7 Total Energy Function

It is now possible to define functions on the neighborhood system. The total energy is given by

$$U_{\text{total}}(x, y) = \alpha U_{\text{prior}}(x) + (1 - \alpha) U_{\text{observation}}(x, y), \tag{3.17}$$

in which $\alpha \in [0 : 1]$ weights the influence of the prior and observation terms. The prior energy function represents the regularization term and the observation energy function is the correlation between the observed data and the estimated original image. To illustrate possible potential functions, examples are given in the following sections.

3.8 Prior Energy Function

The prior energy function includes prior knowledge about pixel configuration in terms of regularities. For instance, the smoothness prior finds application in a lot of image processing problems. A simple version of the smoothness prior is shown for illustration purposes. The used energy function are defined on a $c = 1$ neighborhood system:

$$U_{\text{prior}}(x) = \sum_{C_1 \in \mathcal{C}} V_{C_1} + \sum_{C_2 \in \mathcal{C}} V_{C_2}, \tag{3.18}$$

where C_1, C_2 are two different kind of cliques. Each involving two pixels, which we can visualize as

$$(V_{C_1}) \quad s\bullet \quad t\bullet \quad (V_{C_2}) \quad \begin{matrix} s\bullet \\ t\bullet \end{matrix}$$

The two types of cliques are assigned with potential functions to describe the smoothness in terms of differences:

$$V_{C_{1,2}} = (x_s - x_t)^2. \tag{3.19}$$

The lower the potential the more likely the current configuration would be. In this example, pixel with the same neighborhood value are more likely than pixel with different values.

3.9 Observation Energy Function

The observation potentials describe the observation of the image and its correlation to the estimate of the original image. Basically, in most cases this breaks down to the question of what kind of noise distribution function is appropriate. In a very simple case where we assume no noise at all, which is not the normal case in real world application, the observation potential requires to keep the image pixel value as close as possible to the initial configuration. For instance, this could be realized by a least square function

$$U_{\text{observation}}(x, y) = \sum_{C_3 \in \mathcal{C}} V_{C_3}, \tag{3.20}$$

where C_3 is a clique at single sites s and

$$V_{C_5} = (x_s - y_s)^2, \tag{3.21}$$

where x and y are hypothetical pixel value and observed pixel value, respectively.

3.10 Energy Minimization

To calculate the presented MAP of the pixel configuration of an image, the problem was transformed to energies, which are defined on a neighborhood system with potential functions assigned to cliques. To find the MAP it is now possible to minimize the energy functions. Reconsider the following situation:

$$P(x|y) \propto \exp(-U_{\text{total}}(x, y)), \tag{3.22}$$

$$\hat{x} = \arg\max_x P(x|y) = \arg\min_x U_{\text{total}}(x, y). \tag{3.23}$$

To maximize $P(x|y)$, $U_{\text{total}}(x, y)$ can be minimized. Even after reformulating the MAP with MRF it is not possible to calculate all configurations and pick the best one. Therefore the configuration space needs to be sampled in some way. There are various ways of sampling the configuration space for MRF, in this text we want to focus on two, a global and a local minimizer.

3.10.1 Markov Chain Monte Carlo with Simulated Annealing

Minimization with Markov Chain Monte Carlo (MCMC) with Simulated Annealing (SA) [39]. This is a global minimizer, so theoretically it is possible to find a global minima. Markov chains are Markov fields in one dimension, a short introduction into Markov chains is given in the next section. Markov chains have no memory of the past steps, the next step is calculated based on the current step. Monte Carlo methods are methods that incorporate randomness into the minimization, the next step is estimated by means of probability, so new configurations do not always assure energy minimization. Simulated Annealing is a concept taken from physics and represents a cooling schedule of the minimization, in which at high temperatures more random configurations are possible. The colder the system gets the less random (more rigid) it behaves.

Markov Random Chains

A Markov Random Chains (MRC) is a chain of states in which next state of the system is only dependent on the current state. Translated into a mathematical formula this yields

$$P(X_t = x_t | X_0 = x_0, X_1 = x_1, ..., X_{t-1} = x_{t-1}) = P(X_t = x_t | X_{t-1} = x_{t-1}), \qquad (3.24)$$

where X_t and x_t are the random variable at time t and its realization at time t, respectively. The probability is not influenced by variables from X_0 to X_{t-2}.

Pseudo Code

Algorithm 1: Markov Chain Monte Carlo method with Simulated Annealing (MCMCSA)

Data: observation image y
Result: estimation of the original $\hat{x}$

$T \leftarrow T_{MAX}$
initialize x
repeat
  foreach *site* $s \in S$ **do**
    randomly select $e \in \Gamma$
    $\triangle U = U_{\text{total}}(x_s, y_s) - U_{\text{total}}(x_s + e, y_s)$
    $P_{\text{accept}} \leftarrow \min(1, \exp(-\triangle U/T))$
    $x_s \leftarrow x + e$ with the probability P_{accept}
  $T \leftarrow T \cdot \text{coolingRate}$
until $T \leftarrow T_{MIN}$

In Alg. 1 U_{total} is calculated according to (4.5) and Γ is a set of values describing the next estimate, e.g. $\Gamma = \{(0.1, -0.1), (0.001, 0.001), \dots\}$. By varying Γ one can set the step size for the optimization method. The temperature T represents the cooling of the system, which in the beginning allows the algorithm to get an overview of the function and as time passes, the search focuses in one direction. Theoretically this optimization method guarantees convergences to the global minimum, but in practice it requires a lot of computational effort and the convergence is strongly dependent on the cooling schedule.

3.10.2 Iterated Conditional Modes

Iterated Conditional Modes (ICM) [39] is a deterministic approach. No randomness is involved, so it can not escape local minima. On the other hand it converges usually within a few iterations.

Pseudo Code

Algorithm 2: Iterated Conditional Modes (ICM)

Data: observation image y
Result: estimation of the original image $\hat{x}$
initialize x
repeat
  foreach site $s \in S$ **do**
    $x_s \leftarrow \arg\min_{\mathbf{e}\in\Gamma}(U_{\text{total}}(x_s + e, y_s)$
until U_{total} stabilizes

In Alg. 2, U_{total} is calculated according to (4.5) and Γ is a set of values describing the next estimate, e.g. $\Gamma = \{(0.1, -0.1), (0.001, 0.001), \dots\}$. By varying Γ one can set the step size for the optimization method.

4

New Method for Soft Tissue Deformations: Hierarchy Displacement Model (HDM)

In the following the model to simulate soft tissue deformations using HMRF regularization is described in detail, in the text it will be referred to as Hierarchy Displacement Model (HDM). Firstly, the input images needed for the model are explained. Secondly, the prior and observation energy are derived. Thirdly, the hierarchical approach for solving MRF regularization for soft tissue deformation is presented. Lastly, a theoretical conflict with the clique definitions is elaborated.

The aim of the method is to compute vector displacement fields that conform both to a given boundary condition image and a set of mechanical properties of the underlying tissue. In the following it is demonstrated how this is solved using Markov Random Field (MRF) regularization. Using MRF enables us to describe the underlying mechanical properties of the tissue using a prior energy term, and to set the boundary conditions using an observation energy term.

4.1 Displacement Vector Fields Image

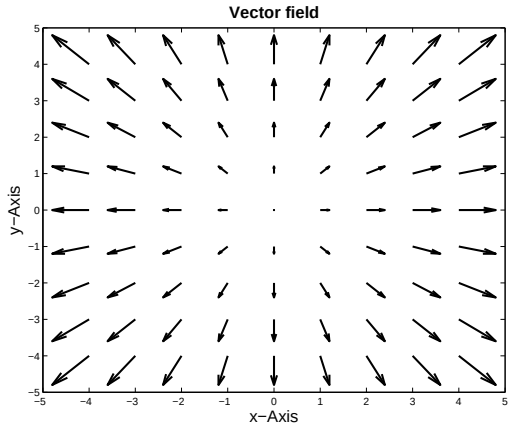


Fig. 4.1. The depicted vector field can be used to describe expansion of an object when applied as a displacement vector field.

In the previous chapter the regularization of scalar images was presented to illustrate MRF. In the HDM, vector images are regularized. Vector images are discrete vector fields. In vector fields a vector is assigned to every point in the Euclidean space:

$$f : \mathcal{R}^n \mapsto \mathcal{R}^n, \tag{4.1}$$

where f is the mapping function. For the HDM in this thesis, the mapping function is

$$f : \mathcal{R}^2 \mapsto \mathcal{R}^2. \tag{4.2}$$

In the discrete version, the vector field is defined over a finite index set S with elements s called sites, where to each s a vector is assigned. In Fig. 4.1 a vector field is defined with

$$f(x, y) = \begin{bmatrix} x \\ y \end{bmatrix} \tag{4.3}$$

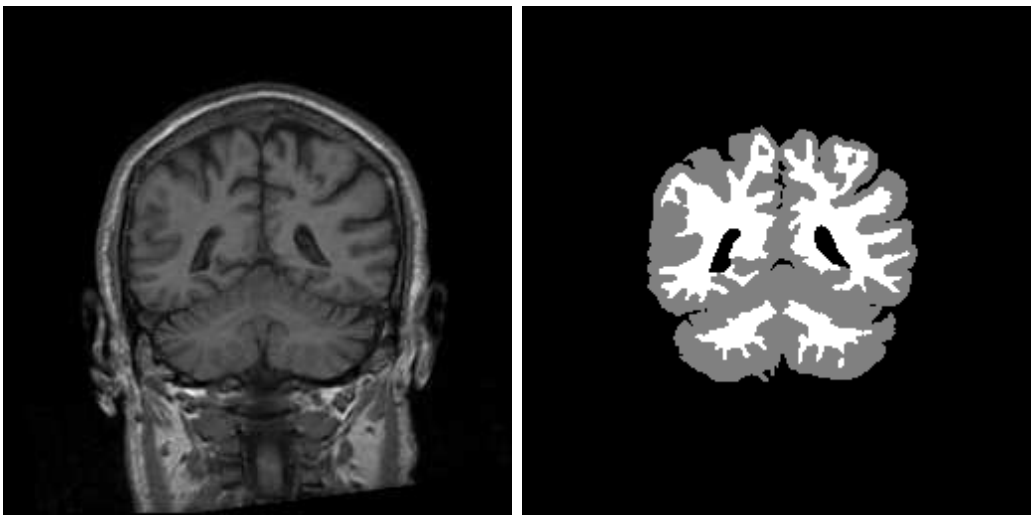
where f is sampled at all sites of S.

The displacement vector field is described using a multivariate random variable $\boldsymbol{D}$. A realization of the vector field is described by $\boldsymbol{d}$. In other words, $\boldsymbol{D}$ is the set of all vector images. To each concrete vector image $\boldsymbol{d}$ a probability is assigned. $\boldsymbol{D_s}$ and $\boldsymbol{d_s}$ are the multivariate random variable and the concrete displacement vector, where s denotes the spatial location of the displacement vector.

Because no initial guess for the initial displacement vector field is available the field is set to null vectors at all sites.

4.2 Input Image

In Fig. 4.2a an example of an input image is depicted. In this case a T1-weighted MRI brain image is shown from dorsal.



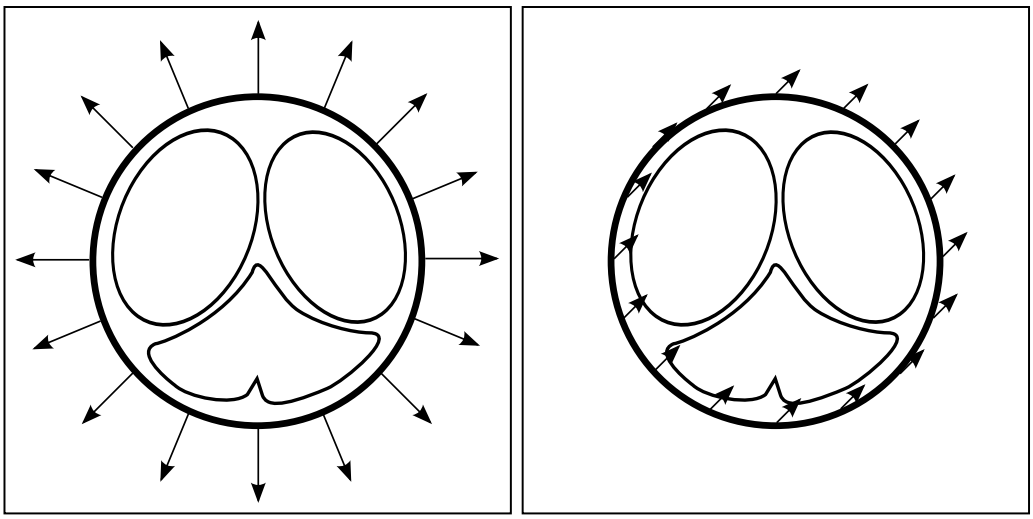
(*a*) (*b*)

Fig. 4.2. T1-weighted MRI brain image viewed dorsal (a) and expert segmented image (b). MR brain data set 788_6_m and its manual segmentation was provided by the Center for Morphometric Analysis at Massachusetts General Hospital and is available at http://www.cma.mgh.harvard.edu/ibsr/.

4.3 Segmented Input Image

In the example seen in Fig. 4.2b, the segmented input image divides the input image into different areas: gray, white and other matter. Each area has underlining mechanical properties assigned.

4.4 Boundary Conditions Image



(*a*) (*b*)

Fig. 4.3. The depicted vector field can be used to describe expansion of an object when applied as a displacement vector field (a). Confidence image for the displacement at the skull (b). The vectors shown in (b) do not have a geometrical meaning. They are oriented in a 45 degree angle because both components of the vector are 1 on the boundary of the skull.

The boundary condition image represented by $\boldsymbol{y}$ defines known displacement vectors. To simulate an outward expansion the schematic boundary condition image in 4.3a is presented. The image shows displacements at the skull. This illustrates an example of image registration, where a source image is transformed into a target image.

4.5 Confidence Image

The confidence image defined as $\boldsymbol{c}$ expresses the confidence in a boundary condition image. In the current implementation, it is a binary vector-valued image where a value of 0 means no confidence in the boundary condition image at that pixel and a value of 1 means complete confidence. In Fig. 4.3b the confidence at the skull is set to 1 for both components to simulate an expansion. The use of confidence fields is described in [30].

4.6 Biomechanical Property Image

The biomechanical properties image $\boldsymbol{m}$ assigns Young's modulus values to segmented image areas. Young's modulus values describe the stiffness of a material by

$$E = \frac{\sigma}{\varepsilon}, \tag{4.4}$$

where E, σ and ε are Young's modulus, stress and strain, respectively. In practice mechanical properties can be assigned using segmentation and classification algorithms.

Example 4.1. A value of 40 MPa for white matter, 20 MPa for grey matter and 10 MPa for other matter. The indicated Young's modulus values are only for illustration purposes and do not correlate with real data.

4.7 Neighborhood System

For the energy function, cliques involving three pixels are created. These cliques are defined in a MRF way and thus valid within a certain neighborhood system. The chosen neighborhood system is of type $c = 2$, for details on neighborhood systems it is referred to Chapter 3.

4.8 Total Energy

The total energy to minimize:

$$U_{\text{total}}(\boldsymbol{y}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{m}) = U_{\text{prior}}(\boldsymbol{d}, \boldsymbol{m}) + U_{\text{observation}}(\boldsymbol{y}, \boldsymbol{c}, \boldsymbol{d}), \tag{4.5}$$

where U_{prior} represent the prior biomechanical information and $U_{\text{observation}}$ the correlation of the displacement vector field with given boundary condition data.

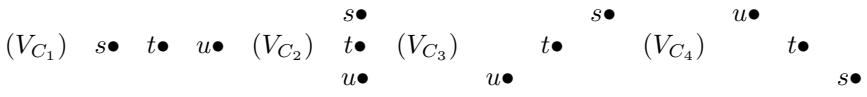
Note 4.2. No weighting function is needed. The weighting of prior an observation term is realized inside of the functions to be able to weight each vector component of the image separately.

4.9 Prior Energy

In the current implementation, the prior energy is based on the mechanical properties of the underlying tissue. The prior energy should therefore be formulated so it is at its minimum when the deformation of the tissue conforms to the expected mechanical properties. Mechanical properties of the tissue are modeled using a finite difference approach, where the local tissue characteristics are based on Young's modulus. The used energy function is then

$$U_{\text{prior}}(\boldsymbol{d}, \boldsymbol{m}) = \sum_{C_1 \in \mathcal{C}} V_{C_1} + \sum_{C_2 \in \mathcal{C}} V_{C_2} + \sum_{C_3 \in \mathcal{C}} V_{C_3} + \sum_{C_4 \in \mathcal{C}} V_{C_4}, \tag{4.6}$$

where C_1, C_2, C_3 and C_4 are four different kind of cliques. Each involving three pixels, which can be visualized as



The four types of cliques are assigned with potential functions to described the biomechanical process of deformation:

$$V_{C_1} = \left(\frac{m_t}{m_u}(d_{x_t} - d_{x_s}) - (d_{x_u} - d_{x_t})\right)^2 + \left(\frac{m_t}{m_s}(d_{x_t} - d_{x_u}) - (d_{x_s} - d_{x_t})\right)^2 \tag{4.7}$$

$$V_{C_2} = \left(\frac{m_t}{m_u}(d_{y_t} - d_{y_s}) - (d_{y_u} - d_{y_t})\right)^2 + \left(\frac{m_t}{m_s}(d_{y_t} - d_{y_u}) - (d_{y_s} - d_{y_t})\right)^2 \tag{4.8}$$

$$V_{C_{3,4}} = \Big|\underbrace{\frac{m_t}{m_u}(\boldsymbol{d_t} - \mathbf{d_s})}_{(a)} - \underbrace{(\boldsymbol{d_u} - \boldsymbol{d_t})}_{(b)}\Big|^2 + \Big|\underbrace{\frac{m_t}{m_s}(\boldsymbol{d_t} - \boldsymbol{d_u})}_{(c)} - \underbrace{(\boldsymbol{d_s} - \boldsymbol{d_t})}_{(d)}\Big|^2, \tag{4.9}$$

where $\boldsymbol{d_i} = \begin{bmatrix} d_{x_i} & d_{y_i} \end{bmatrix}^T$ and m_i are the displacement vector at site i and the biomechanical property image sampled at i, respectively. Indicated by letters (a) to (d) there are four

parts that define the prior knowledge, analog for (4.7) and (4.8). Parts (a) and (b) describe the change of Young's modulus in one direction and (c) and (d) the change in the opposite direction. The goal of the prior energy is to reach:

$$\frac{m_t}{m_u}(\boldsymbol{d_t} - \boldsymbol{d_s}) = (\boldsymbol{d_u} - \boldsymbol{d_t}), \tag{4.10}$$

at every site. The relative change at site t is equal to the relative change at u by a ratio of m_t/m_u. The differences $(\boldsymbol{d_t} - \boldsymbol{d_s})$ and $(\boldsymbol{d_u} - \boldsymbol{d_t})$ describe the elasticity in terms of relative displacements at site t and u, respectively. To assign an energy value to the relative difference this translates to

$$\left| \frac{m_t}{m_u}(\boldsymbol{d_t} - \boldsymbol{d_s}) - (\boldsymbol{d_u} - \boldsymbol{d_t}) \right|^2. \tag{4.11}$$

This means that the minimum energy at site t is given by 0. The bigger the difference the higher the energy. Also by taking the square of the energy value, values > 1 contribute more than values < 1. The energy function also detects non-monotonic fields. A monotonic field is defined as

$$|\boldsymbol{d_s}| \leq |\boldsymbol{d_t}| \leq |\boldsymbol{d_u}| \tag{4.12}$$

or

$$|\boldsymbol{d_s}| \geq |\boldsymbol{d_t}| \geq |\boldsymbol{d_u}|. \tag{4.13}$$

Example 4.3 (Zero energy at site t). In this example a situation is presented where two neighboring pixel have different Young's modulus values. The minimum energy is reached when (4.10) is fulfilled. A possible configuration for three displacement vectors $\boldsymbol{d_s}$,$\boldsymbol{d_t}$ and $\boldsymbol{d_u}$ is given.

$$\boldsymbol{d_s} = \begin{bmatrix}1\\1\end{bmatrix} \text{ mm}, \boldsymbol{d_t} = \begin{bmatrix}2\\2\end{bmatrix} \text{ mm}, \boldsymbol{d_u} = \begin{bmatrix}4\\4\end{bmatrix} \text{ mm}, m_t = 20 \text{ MPa}, m_u = 10 \text{ MPa}$$

$$\frac{20 \text{ MPa}}{10 \text{ MPa}} \left(\begin{bmatrix}2\\2\end{bmatrix} \text{ mm} - \begin{bmatrix}1\\1\end{bmatrix} \text{ mm} \right) = \left(\begin{bmatrix}4\\4\end{bmatrix} \text{ mm} - \begin{bmatrix}2\\2\end{bmatrix} \text{ mm} \right)$$

Example 4.4 (Non-monotonic field at site t). In the second example both pixel have the same Young's modulus. Inserting the example values into (4.11) gives the energy value.

$$\boldsymbol{d_s} = \begin{bmatrix}1\\1\end{bmatrix} \text{ mm}, \boldsymbol{d_t} = \begin{bmatrix}2\\2\end{bmatrix} \text{ mm}, \boldsymbol{d_u} = \begin{bmatrix}1\\1\end{bmatrix} \text{ mm}, m_t = 10 \text{ MPa}, m_u = 10 \text{ MPa}$$

$$\left| \frac{10 \text{ MPa}}{10 \text{ MPa}} \left(\begin{bmatrix}2\\2\end{bmatrix} \text{ mm} - \begin{bmatrix}1\\1\end{bmatrix} \text{ mm} \right) - \left(\begin{bmatrix}1\\1\end{bmatrix} \text{ mm} - \begin{bmatrix}2\\2\end{bmatrix} \text{ mm} \right) \right|^2 = 2.8$$

Even though the difference between sites and the biomechanical properties are the same, a non-zero value is calculated. Non-monotonic fields cause overlapping of sites, which is not realistic.

4.10 Observation Energy

The current application uses the observation energy to set the boundary conditions:

$$U_{\text{observation}}(\mathbf{y}, \mathbf{c}, \mathbf{d}) = \sum_{C_5 \in \mathcal{C}} V_{C_5} + \sum_{C_6 \in \mathcal{C}} V_{C_6}, \tag{4.14}$$

where C_5 and C_6 are cliques at single sites s and

$$V_{C_5} = p_{x_s} |y_{x_s} - d_{x_s}| \tag{4.15}$$

$$V_{C_6} = p_{y_s} |y_{y_s} - d_{y_s}|, \tag{4.16}$$

where $\boldsymbol{p}$ is the penalty vector field image. The penalty image is proportional to the confidence image

$$p_{x_s} = Kc_{x_s}, \qquad p_{y_s} = Kc_{y_s}, \tag{4.17}$$

where $\boldsymbol{c_s} = \begin{bmatrix} c_{x_s} & c_{y_s} \end{bmatrix}^T$, $\boldsymbol{p_s} = \begin{bmatrix} p_{x_s} & p_{y_s} \end{bmatrix}^T$ and K is the penalty constant. By issuing a penalty, a low energy value is sensitive to the difference between the boundary vector and the displacement vector. Hence, the probability is high when the displacement vector at a site coincides with the boundary condition vector whenever boundary conditions are available, indicated by the confidence vector.

4.11 Hierarchical Markov Random Fields

The idea is to propagate the geometrical information through hierarchical levels. Low resolution scale levels express global geometrical features whereas high resolution scale levels are used for fine tuning. Geometrical information is needed to converge to a global solution. By using a hierarchical approach, local energy optimizers can be used to find global solutions. Local optimizers are stable and predictable whereas global optimizer do not guarantee convergence and the computation time is enormous, this is further discussed in the next section.

In [33] HMRF for segmentation is presented, it follows the adaptation to HDM. Let $G = (S, L)$ be a graph composed of a set S of nodes and a set L of edges. A tree is a connected graph that has no cycle. Each node has a unique parent node, except the root node r. A quadtree, as illustrated in Fig. 4.4, is a special case of a tree. Each node in a quadtree has four child nodes except the leave nodes, which are terminal nodes. Set S can be partitioned into subsets we call *scale levels*. The subset, $S = S^0 \cup S^1 \cup \cdots \cup S^R$, are distinguished according to the path length from each node to the root. S^0 is the subset with the most elements and $S^R = \{r\}$ the one with the least elements. A subset contains 4^{R-n} sites, where R represents the scale and n the distance from the root node r.

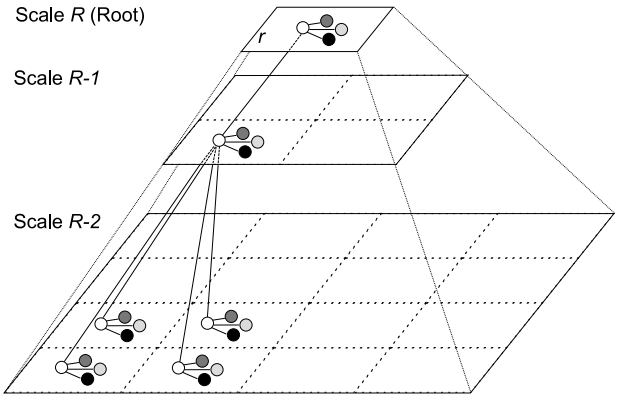


Fig. 4.4. Dependency graph corresponding to a quadtree structure. With white circles representing displacement vectors, light gray circles representing boundary condition vectors, dark gray circles representing confidence vectors and black circles representing mechanical property vectors.

Different interpolation methods are used for projecting the images from one scale level to the next. $\boldsymbol{d}$ is interpolated linearly, because linear elastic behavior was used to describe the material properties of the image. On the other hand, for the interpolation of $\boldsymbol{m}$, $\boldsymbol{y}$ and $\boldsymbol{c}$, nearest neighbor interpolation is used to preserve information at interfaces between different tissue types and prevent over-smoothing. In Fig. 4.4 the propagation

between scale levels is shown, starting on top with the root element r. To each white circle $\boldsymbol{d_s} \in \boldsymbol{d}$, three circles $\boldsymbol{m_s} \in \boldsymbol{m}$, $\boldsymbol{y_s} \in \boldsymbol{y}$ and $\boldsymbol{c_s} \in \boldsymbol{c}$, are attached, representing interpolated pixel values at corresponding locations s.

4.12 Energy Minimization

In practice, the crucial point of optimization is a good initial guess. With initial guesses far off the global minimum, local energy optimizers like ICM will most likely stay at a local minimum and global energy optimizers like MCMCSA methods will take a long time; if the cooling schedule is not adjusted to the problem, it might even stop at a local minimum as well. In the current approach, the overall geometrical structure of tissues and their corresponding mechanical properties are crucial in obtaining a good initial guess. A top-down hierarchical approach is used to supplement the local nature of the used energy terms. Each level of the hierarchy is used as an initial guess for the next level. Iterated Conditional Modes is used as local optimizer within each hierarchy level because a good initial guess from the previous level can be assumed.

4.13 Violation of Clique Definition

It seems that we have violated Def. 3.4 by using cliques where s and u are not neighbors. In [14] the same kind of cliques are used for constrained restoration and the recovery of discontinuities of images. It seems that the violation of the clique definition is justified by the fact that finite differences are used, so the cliques

$$s\bullet \quad t\bullet \quad u\bullet \ ,$$

can be reformulated as

$$(t-s)\bullet \quad (u-t)\bullet \ ,$$

thus the correctness of the clique definition is proven.

5

Simulations

In the following, five different simulations are analyzed. The first simulation is a very basic one. It shows the deformation of one square when an uniform displacement is applied at two boundaries of the image whereas the other two boundaries are fixed. The goal of this simulation is to show the behavior of HDM when squared geometries are modeled.

The second simulation is a modification of the first one. It will be shown that in the first simulation the interfaces are too smooth compared to the ground truth displacement vector field. It is shown that by modifying the number of hierarchical levels the result are more accurate.

The third simulation represents an extreme case of hierarchical level manipulation, only the final scale levels is calculated. It is shown that only a local minimum of the solution is reached. Even by increasing the number of iterations no improvement is visible.

An image similar to an MRI image of the brain is taken for the last two simulations. A uniform expansion in all direction is applied to the synthetic MRI image. The results show that circular geometry are handled better than squared geometry. To show the improvement that can be achieved by accounting for the underlining biomechanical property of an image, a comparison with an elastic deformation registration method [2] is illustrated.

At last, a simulation with a radial expansion in all directions is analyzed. This simulation provides the model with different boundary conditions than the other four experiments. The boundary conditions are not only at the border of the image but also inside the image. It is explained how the boundary conditions inside the image are obtained in practice and how to interpret them.

5.1 Ground Truth

To validate the results of the simulation the same mechanical deformation are modeled with FEM and solved in ABAQUS[1] and used as ground truth images. A comparison of data using a root mean square deviation (RMSD) was realized,

$$RMSD(x) = \sqrt{\frac{1}{|S|}\sum_{s\in S} x_s^2}, \tag{5.1}$$

where $x_s = |\mathbf{a_s} - \hat{\mathbf{d}}_\mathbf{s}|$, $\mathbf{a_s}$ is the displacement vector obtained from ABAQUS, $\hat{\mathbf{d}}_\mathbf{s}$ is the displacement vector obtained from our model and $|S|$ is the total number of sites.

[1] ABAQUS is a software tool to create FEM models, Version 6.7-1 was used.

5.2 Mean Energy Value

The *mean energy value* mentioned in the following section is calculated as follows:

$$\bar{U} = \frac{1}{|S|}\sum_{s \in S} U_s, \tag{5.2}$$

where U_s is the total energy at site s and $|S|$ is the total number of sites.

5.3 Iterations

Considering the restrictions on graphics hardware, read-only and write-only memory and no update within the iteration, we carried out experiments to find out whether or not the restrictions significantly reduced the quality of the simulation. Firstly, the simulation was run in CPU mode. The stop criteria for the optimization on the CPU implementation is

$$\varepsilon < RMSD(\bar{U}_i - \bar{U}_{i-1}), \tag{5.3}$$

where $\bar{U}_i$ and ε represent the mean energy value at iteration step i and the stop criteria, respectively. Due to the restricted hardware capabilities on the GPU no dynamic stop criteria computation can be done for the GPU version, thus the number of iterations need to be defined before the start of a simulation. To find the optimal number of iterations a heuristic approach examining the model with a range of values was used. Based on the result obtained from the CPU implementation and the convergence graph of the GPU implementation, the optimal number of iterations are chosen.

5.4 Step Size

For the ICM energy optimization the step size set was set to $\{-1, 0.1, 1, 0.1\}$. Hence, the accuracy of the results are in the range of 0.1. As described in the Sec. 6.2.4, there is a maximum of arithmetic operations allowed per shader program on some of todays graphics hardware. With the GPU used for this simulations (specification can be found in Appendix B) the maximum allowed number is 512 which was exceeded when a bigger step size set was used, e.g. $\{-1, 0.1, -0.01, 1, 0.1, 0.01\}$. On newer hardware these problems can be neglected. It can be assumed that with a bigger set, more accurate results can be obtained and faster convergence is possible.

5.5 Image Profiles

In order to identify wrong behavior of the model it is helpful to plot profile graphs of the displacement vectors image. The following components are considered:

- In horizontal profiles, depicted in Fig. 5.1a, the x component of the displacement vector is considered
- In vertical profiles, depicted in Fig. 5.1b, the y component of the displacement vector is considered
- In diagonal profiles, depicted in Fig. 5.1c-d, the magnitude of the displacement vector is considered

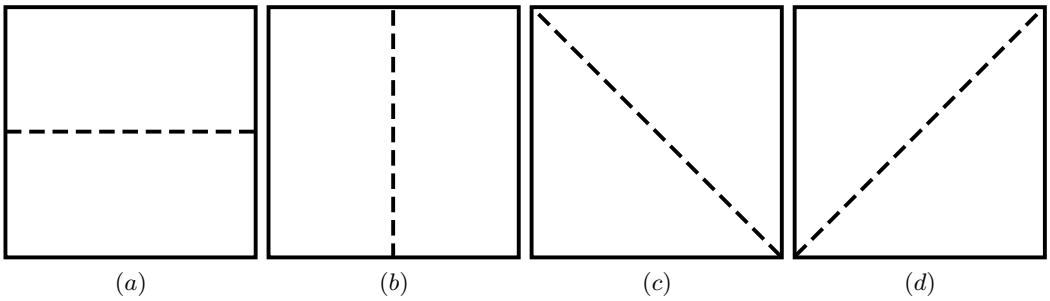


Fig. 5.1. (a) horizontal profile, (b) vertical profile, (c) diagonal profile from top-left to bottom-right corner and (d) diagonal profile form bottom-left to top-right corner.

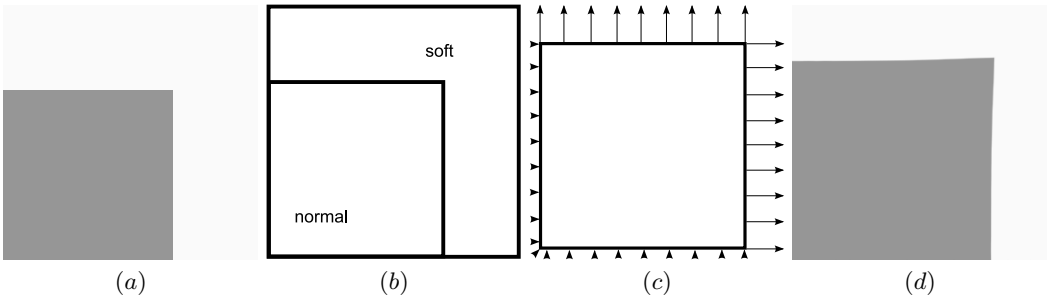


Fig. 5.2. A set of synthetic images with square geometry and two different biomechanical properties. (a) input image, (b) segmentation of the input image, (c) representation of boundary condition image with two borders having a displacement and two being fixed and (d) deformed input image.

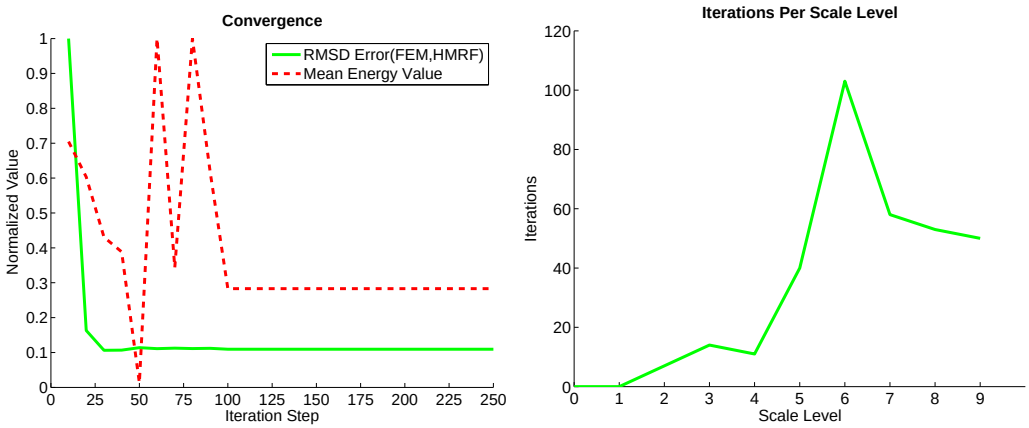
5.6 Square Geometry

5.6.1 Test Data

A set of synthetic data is created to simulate square shaped tissues. In Fig. 5.2a, a test image with two tissues is depicted. Namely, normal matter (gray color) and soft tissue (white color), depicted in Fig. 5.2b. Secondly, a boundary condition image is constructed as seen in Fig. 5.2c, where the displacement vectors are shown. In this example, the boundary condition image describes a displacement at two borders whereas the other two are fixed (i.e. an arrow with no magnitude means no displacement at this position). To be able to control the amount of confidence in the boundary condition image, a confidence image is also provided. In the presented example the confidence image coincides with 5.2c except the vectors are unit vectors. Finally an image describing the local material properties is used. In this example, it corresponds to the input image where each pixel is assigned the Young's modulus stiffness based on the underlying tissue. A value of 20 MPa is used for normal matter and 10 MPa for soft tissue.

5.6.2 Iterations

The results in Fig. 5.3a indicate that after approximately 50 iteration steps, the RMSD error has stabilized, whereas the mean energy value still fluctuates. The data suggest that the field is changing but without significant impact on the quality of the simulations. In Fig. 5.3b the iteration peak is at scale level 6 at around 100 iterations.



(a) (b)

Fig. 5.3. Graphs for evaluation of optimal number of iterations: (a) convergence graph with normalized y-axis for better comparability, (b) number of iterations per scale level measured with the CPU implementation.

Table 5.1. Experimental data obtained to evaluate quality drawbacks due to GPU restrictions, 3600 elements of type CPS4R (4-node bilinear, reduced integration with hourglass control) are used for the FEM calculations

Method	Mean energy value	RMSD error FEM - HMRF	Iterations	Computation time
CPU	0.15469	1.54026 pixel	336	3.94 sec.
GPU	0.00350	1.72135 pixel	500	1.27 sec.
FEM	-	-	-	1.30 sec.

Tab. 5.1 confirms the interpretation of the graphs and shows that the RMSD error still shows a difference of ≈ 0.2 pixel when 50 iterations per scale level are used. Profile differences, depicted in Fig. 5.5, help to further analyze the results.

In Tab. 5.1 the FEM computation times are very similar to the times obtained with the GPU implementation. The fact that the shape under study is geometrically very simple, favors the computation with FEM because only 3600 elements were used for the calculations. For more complex shapes the computation times for FEM will increase whereas for the GPU implementation the time will be in the same order of magnitude, because the number of pixel that need to be calculated stay the same. This is also valid for the other simulations in this chapter.

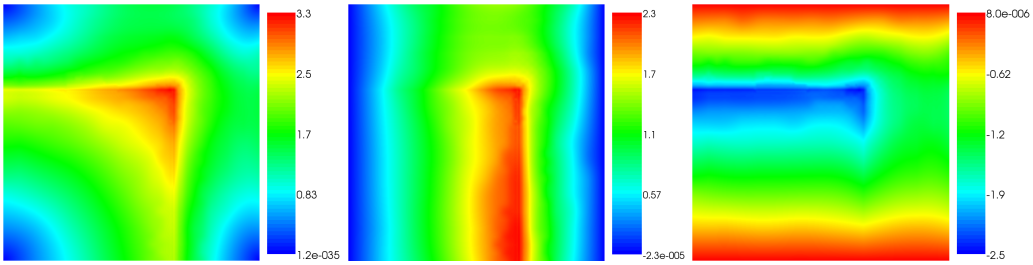
5.6.3 Computation Time

Table 5.2. Comparison of computation times between GPU and CPU implementation with 50 iteration steps

Image Size	Time on GPU	Time on CPU	Speed-up Factor
300 x 300	1.27 sec.	67.02 sec.	52.78
600 x 600	4.94 sec.	264.83 sec.	53.61
1200 x 1200	20.28 sec.	1056.09 sec.	52.08

In Tab. 5.2 three experiments with different image sizes are listed. The speed-up factor is around 53 and does not depend on the image size. To be able to compare GPU and CPU code the number of iterations per scale level is fixed to 50 and pixel update during iterations is disabled.

5.6.4 Difference Image



(*a*) (*b*) (*c*)

Fig. 5.4. ABAQUS and HMRF displacement vector field images differences: (a) magnitude of the vector, (b) x-component of the vector and (c) y-component of the vector.

The problem areas of the simulation can clearly be identified by Figs. 5.4a-c. The biggest differences can be found at the interface of issues.

5.6.5 Profile Difference

The difference images show that the problem areas occur in the region of tissue interfaces, Figs. 5.5a-d confirm this. To further analyze the problem areas at the tissue interfaces, hierarchical profiles are depicted in the next section.

5.6.6 Hierarchical Profiles

The per scale level plots, depicted in Figs. 5.6 and 5.7, of the hierarchical process do not show any irregularities. The differences at tissue interfaces are caused by over-smoothing, which is a consequence of the interpolation process. In the next section a way of improving the results is shown. In the simulations with more complex circular shapes, over-smoothing is less significant.

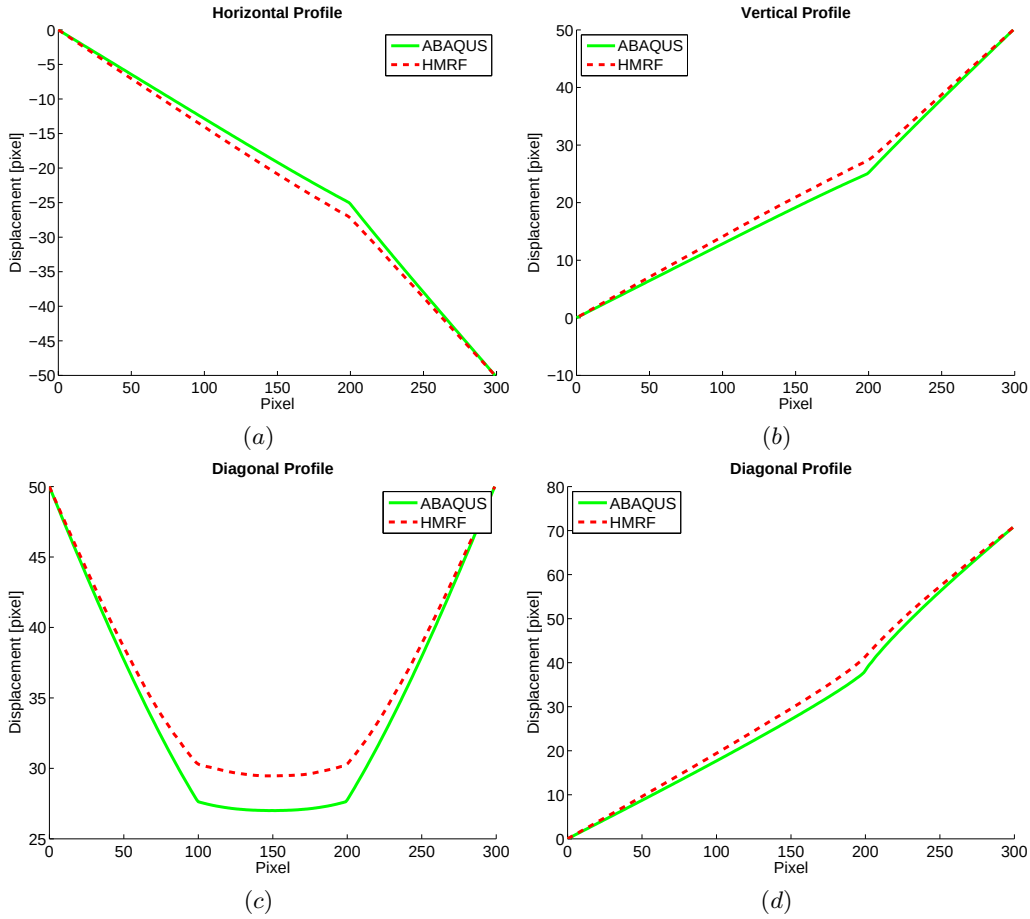


Fig. 5.5. Comparison of ABAQUS and HMRF results with profile images: (a) horizontal profile, (b) vertical profile, (c) diagonal profile from top-left to bottom-right corner and (d) diagonal profile form bottom-left to top-right corner.

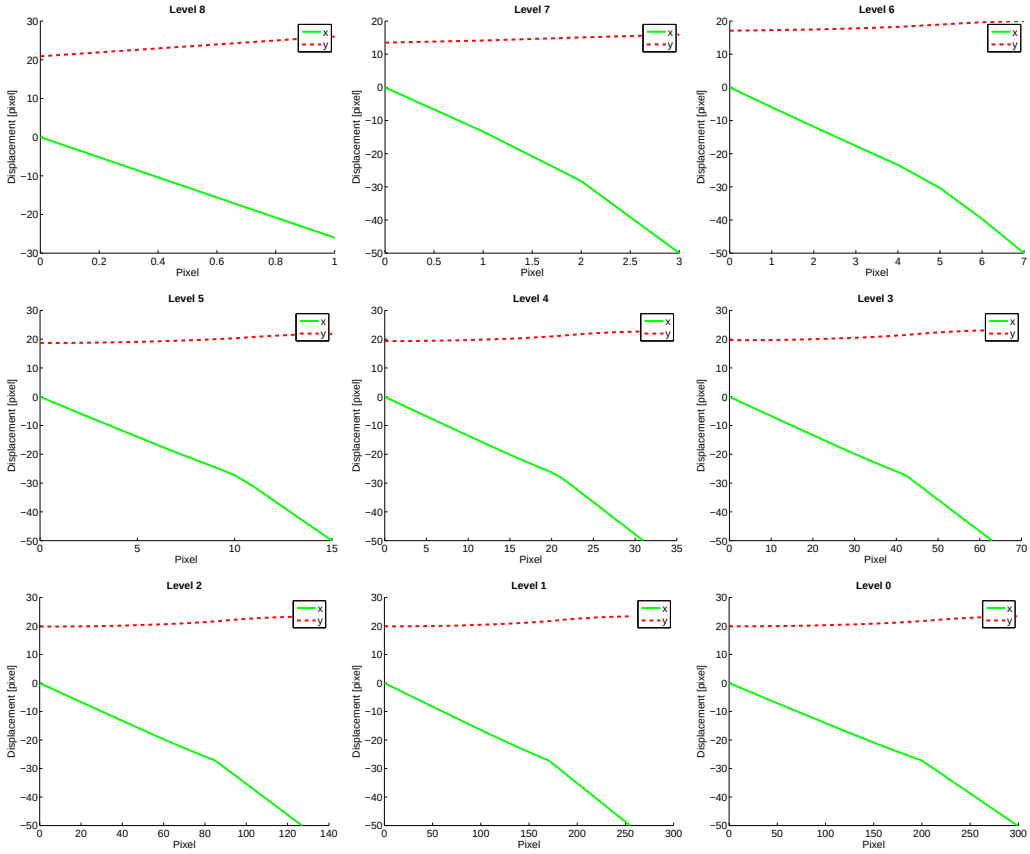


Fig. 5.6. Development of hierarchical process shown with profile images. Horizontal profiles of hierarchical scale levels 8 to 0 starting at 8, where 8 is the coarsest, to 0 the finest image resolution image. Both components are shown, x, y.

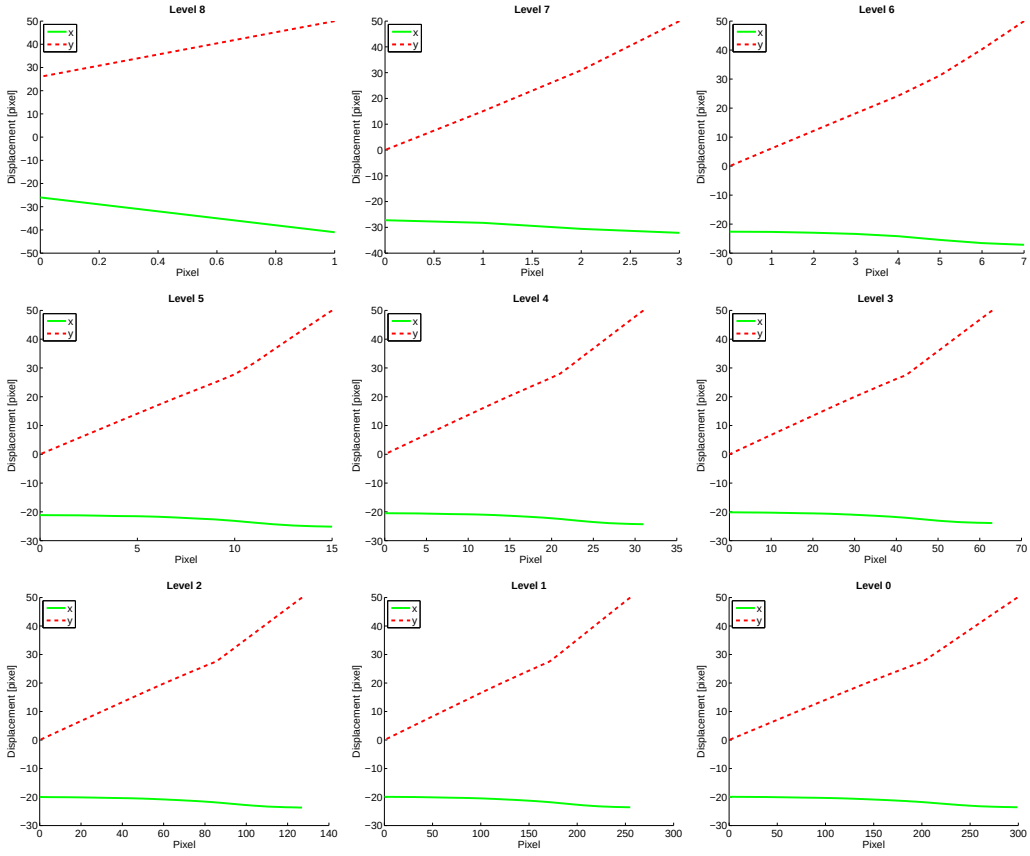


Fig. 5.7. Development of hierarchical process shown with profile images. Vertical profiles of hierarchical scale levels 8 to 0 starting at 8, where 8 is the coarsest, to 0 the finest image resolution image. Both components are shown, x, y.

5.7 Modified Square Geometry

This is a modified version of the previous simulation. With this experiment it is shown that by reducing the number of hierarchical scale levels it is possible to significantly improve the quality of the simulation. The assumption is that the first scale levels do not provide the model with enough biomechanical information to initialize a good first guess. The results from Sect. 5.6 showed that the simulation could not recover from the first initial guess. The idea is to skip the first 4 scale levels and thus provide the model with detailed biomechanical information from the beginning.

5.7.1 Iterations

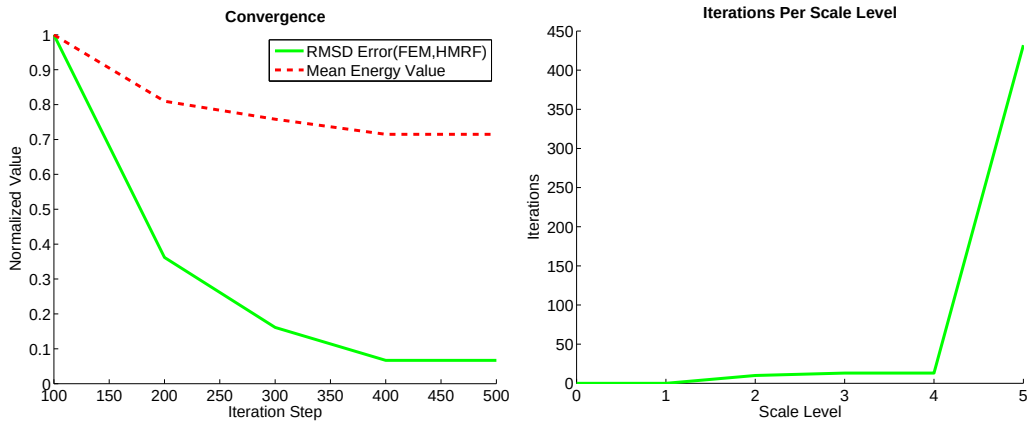


Fig. 5.8. Graphs for evaluation of optimal number of iterations: (a) convergence graph with normalized y-axis for better comparability, (b) number of iterations per scale level measured with the CPU implementation.

Table 5.3. Experimental data obtained to evaluate quality drawbacks due to GPU restrictions, 3600 elements of type CPS4R (4-node bilinear, reduced integration with hourglass control) are used for the FEM calculations

Method	Mean energy value	RMSD error FEM - HMRF	Iterations	Computation time
CPU	0.00356	0.59 pixel	468	5.453 sec.
GPU	0.00359	0.46 pixel	3000	9.516 sec.
FEM	-	-	-	1.30 sec.

What is interesting about Tab. 5.3 is the fact that the computation time for CPU is less than for the GPU implementation. This can be explained by the fixed number of iterations, in this case 500 as seen in Fig. 5.8, that are calculated for each scale level. In case of the CPU implementation the number of iterations decrease rapidly and reach zero at scale level 1 and 0. A possible explanation for this behavior can be found in the scale level 5, which gives a very good starting point for the calculation.

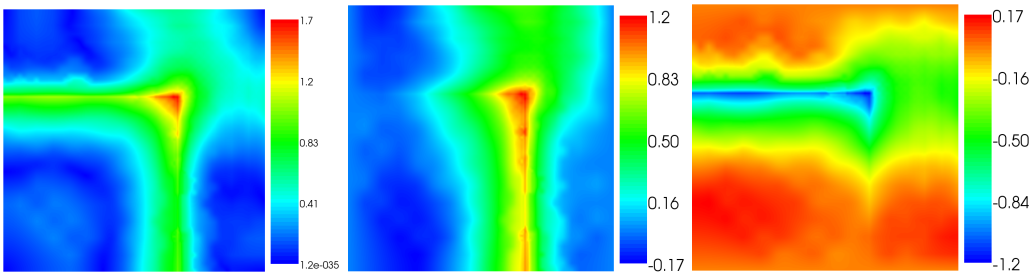
5.7.2 Computation Time

Table 5.4. Comparison of computation times between GPU and CPU implementation with 500 iteration steps, starting at scale level 5

Image Size	Time on GPU	Time on CPU	Speed-up Factor
300 x 300	9.52 sec.	711.75 sec.	74.76
600 x 600	37.52 sec.	2590.91 sec.	69.05
1200 x 1200	151.33 sec.	10329.7 sec.	68.26

Less hierarchical levels and more computation favors the GPU implementations, that is why the speed-up factor is about 1.5× the speed reported in Sec. 5.6, when the number of iterations is fixed.

5.7.3 Difference Image



(*a*) (*b*) (*c*)

Fig. 5.9. ABAQUS and HMRF displacement vector field images differences: (a) magnitude of the vector, (b) x-component of the vector and (c) y-component of the vector.

There is still a difference in the same regions as in Sec. 5.6 but the error is smaller, depicted in Fig. 5.9.

5.7.4 Profile Difference

In Fig. 5.10 the modified simulation shows a very good fit to the ground truth data from ABAQUS. Only minor differences can be seen at the tissue interfaces, e.g. in Fig. 5.10a at pixel 200.

The process starts with better initial scale level calculations and can thus only fine tune the results in the scale levels that follow.

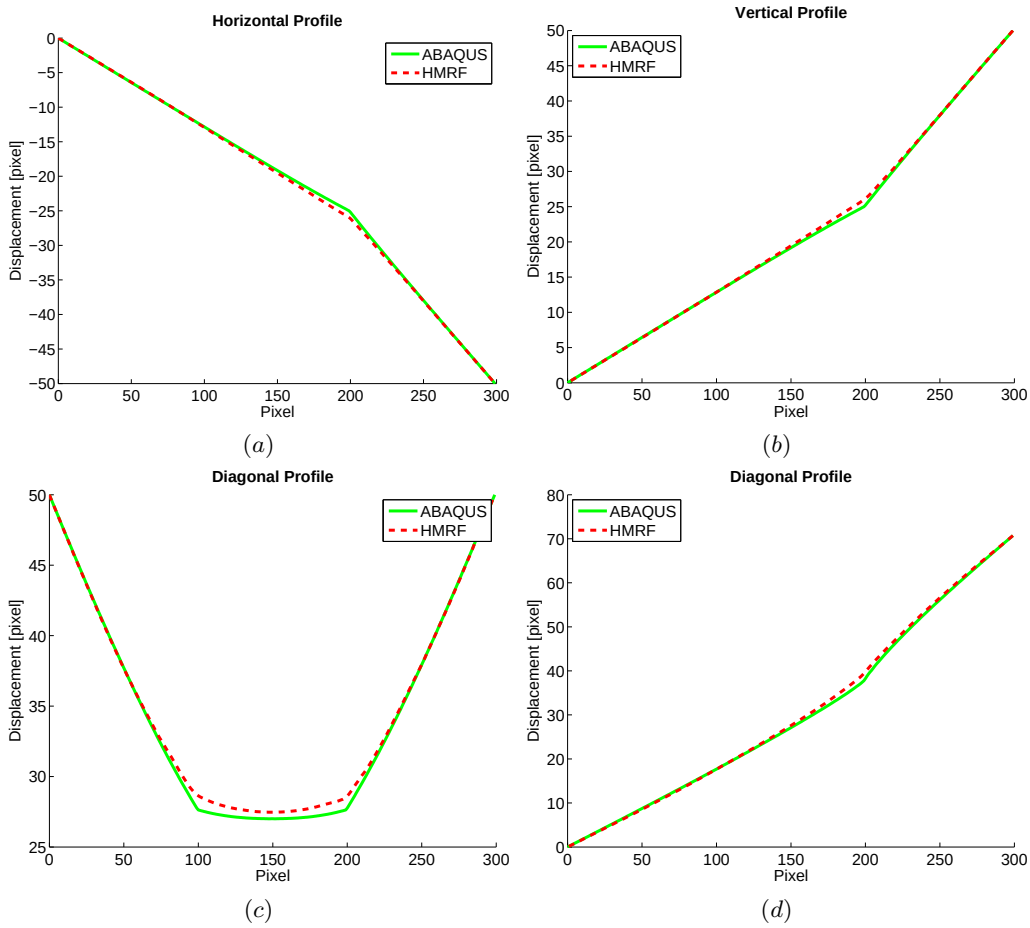


Fig. 5.10. Comparison of ABAQUS and HMRF results with profile images: (a) horizontal profile, (b) vertical profile, (c) diagonal profile from top-left to bottom-right corner and (d) diagonal profile form bottom-left to top-right corner

5.8 Non-Hierarchical Square Geometry

In the third simulation not only the first 4 scale levels are skipped but all of them except the last one. This situation is equivalent to a non-hierarchical approach. The goal is to show the importance of the hierarchical optimization process.

5.8.1 Convergence

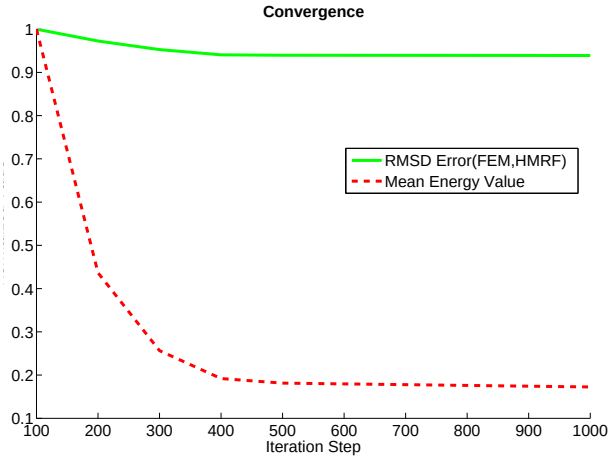


Fig. 5.11. Graph for evaluation of optimal number of iterations, normalized y-axis for better comparability.

The RMSD error and the mean energy value converge both at around 500 iterations. Hence, the following simulation are done with 500 iterations.

5.8.2 Profile Difference

The results in Fig. 5.12 show that only a local minima is reached. To escape the local minima the minimization would need to go through pixel configuration with greater energy values than the current one, this is not possible when ICM is used.

5.9 Conclusion on Variation of Hierarchical Scale Levels

The results obtained in Sects. 5.6, 5.7 and 5.8 highlight that the quality of the simulation can be improved by choosing the right number of hierarchical scale levels. The results suggest that the model is very sensitive to the starting point of the hierarchy.

- If too early, a chance of over-smoothing the solution exists, seen in 5.6.
- If too late, only a local minima is reached, seen in 5.8.

In the shown examples a heuristic approach was used to find out the right starting scale level. In practice this is not possible, so there is potential for future research.

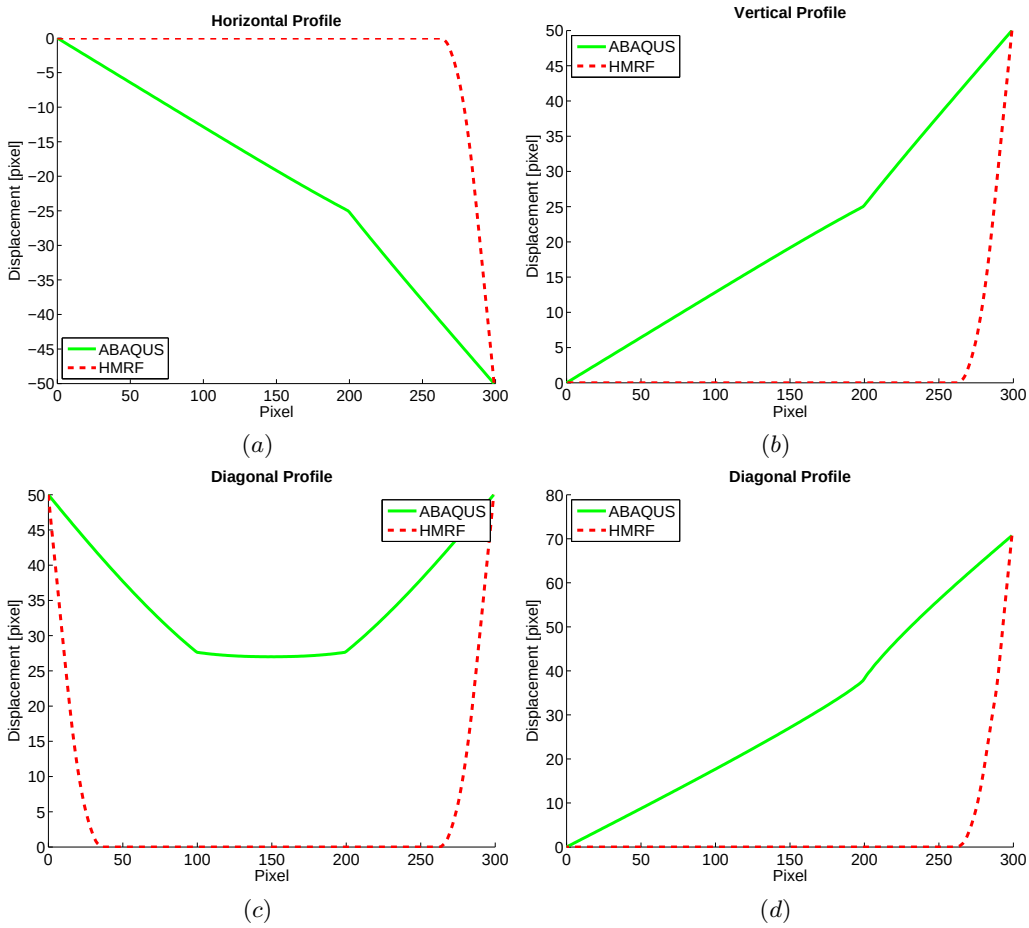


Fig. 5.12. Comparison of ABAQUS and HMRF results with profile images: (a) horizontal profile, (b) vertical profile, (c) diagonal profile from top-left to bottom-right corner and (d) diagonal profile form bottom-left to top-right corner.

5.10 Circular Geometry

As in previous sections synthetic data is used for the simulation. The goal of this experiment is to show the ability of HMRF regularization to handle circular shaped geometries.

5.10.1 Test Data

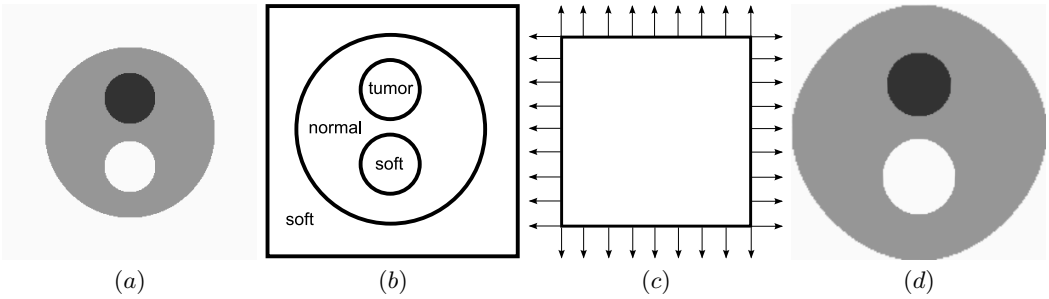


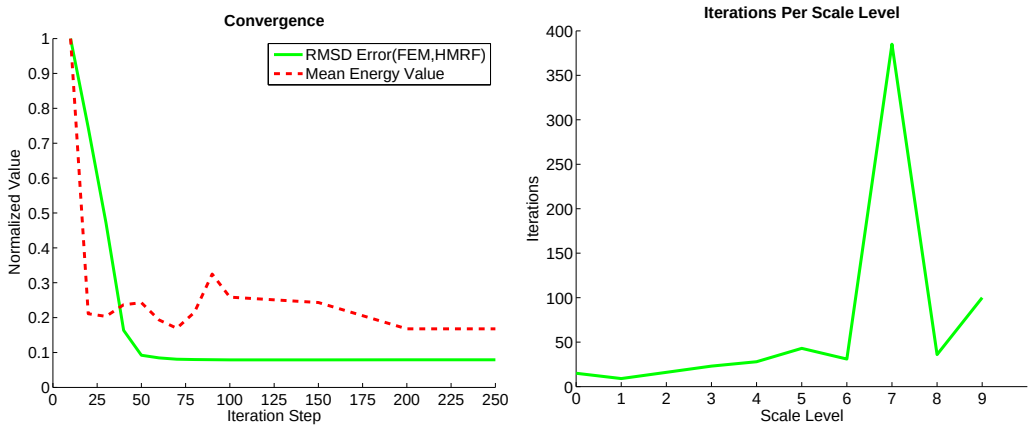
Fig. 5.13. Synthetic MRI brain images: (a) input image, (b) segmentation of the input image, (c) representation of boundary condition image with outward expansion and (d) deformed input image

The data is generated so it resembles an MRI image of a brain, as seen in Fig. 5.13. To simplify notations three tissue types are considered. Namely, normal matter (gray color), tumoral tissue (black color) and soft tissue (white color), depicted in Fig. 5.13b. Secondly, a boundary condition image is constructed as seen in Fig. 5.13c, where the displacement vectors are shown. In this example, the boundary condition image describes an outward expansion. In real applications, the boundary condition image can for example be computed using a non-rigid registration algorithm [34, 2]. To be able to control the amount of confidence in the boundary condition image a confidence image is also provided. In the presented example the confidence image coincides with 5.13c except the vectors are positive unit vectors. Finally, an image describing the local material properties is used. In this example, it corresponds to the input image where each pixel is assigned the Young's modulus stiffness based on the underlying tissue. A value of 40 MPa is used for tumors, 20 MPa for the normal tissue and 10 MPa for soft tissue. In a real application, the mechanical properties can be assigned using a segmentation and classification algorithm.

5.10.2 Iterations

In Fig. 5.14b a computation with the CPU version of the algorithm is presented. The numbers of iterations for each individual scale level are shown, scale 9 is the starting level, which is just one pixel, and 0 is the final full resolution image. At scale level 7, the number of iterations reach a peak value. It can be assumed that at this state major geometry structures appear, which results in big changes of the field configuration. It can be seen that after iteration step 70 the RMSD error has converged. The mean energy value is fluctuating within a certain range, but according to the RMSD error this has no significant influence on the quality of the simulation.

In Tab. 6.1 the numerical values confirm the interpretation of Fig. 5.14. The computation times do not vary as much as in Tab. 5.6 because at high resolution scale levels much less iteration are needed.



(a) (b)

Fig. 5.14. Graphs for evaluation of optimal number of iterations: (a) convergence graph with normalized y-axis for better comparability, (b) number of iterations per scale level measured with the CPU implementation

Table 5.5. Experimental data obtained to evaluate quality drawbacks due to GPU restrictions, 4255 elements of type CPS3 (3-node linear) and CPS4R (4-node bilinear, reduced integration with hourglass control) are used for the FEM calculations

Method	Mean energy value	RMSD error FEM - HMRF	Iterations	Computation time
CPU	0.12984	1.43 pixel	686	23.67 sec.
GPU	0.19404	1.44 pixel	700	1.63 sec.
FEM	-	-	-	1.30 sec.

5.10.3 Computation Time

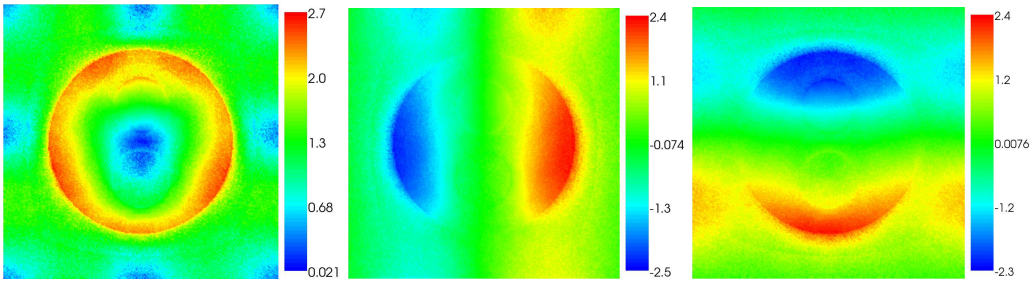
Table 5.6. Comparison of computation times between GPU and CPU implementation with 70 iteration steps

Image Size	Time on GPU	Time on CPU	Speed-up Factor
300 x 300	1.63 sec.	91.39 sec.	56.07
600 x 600	6.42 sec.	363.89 sec.	56.68
1200 x 1200	26.25 sec.	1445.94 sec.	55.08

In Tab. 5.6 a listing on the performance gain with GPU compared to CPU is shown. Three different image sizes have been examined, a speed-up factor of around 55 was found. To be able to compare GPU and CPU implementations the number of iterations per scale level is set to 70 and pixel update during iterations is disabled.

5.10.4 Difference Image

In Fig. 5.15a the difference image with ABAQUS results is shown. It can be seen that the region that differ the most are around the outer corner of the synthetic brain. Further it is interesting to look at the two Figs. 5.15b+c in which the difference per component is



(a) (b) (c)

Fig. 5.15. ABAQUS and HMRF displacement vector field images differences: (a) magnitude of the vector, (b) x component of the vector and (c) y component of the vector

depicted. In both images the results show that most of the differences occur at the outer border of the synthetic brain, which confirm the interpretation of Fig. 5.15a.

5.10.5 Profile Difference

The profile difference in Fig. 5.16 confirm the explanation of the difference images. It can be seen that the two results differ at the outer corner of the synthetic brain.

5.11 Comparison Elastic Registration

The comparison for elastic registration is made with the test data from Sect. 5.10.

In general non-rigid deformation algorithms do not include biomechanical property information which results in non-realistic deformations. Therefore a comparison with an existing elastic deformation registration method [2] was made. 5.17a shows the HMD displacement vector field obtained from the current regularization applied to a grid image. 5.17b is the ABAQUS displacement vector field applied to a grid image. The elastic registration method was used to register the deformation image received from ABAQUS with the input image 5.13a. The HDM and ABAQUS grids show similar deformations whereas the elastic registration method does not consider biomechanical properties, which can be seen by looking at tissue interfaces. The elastic registration image looks smooth because the change of biomechanical properties at the interfaces is not taken into account.

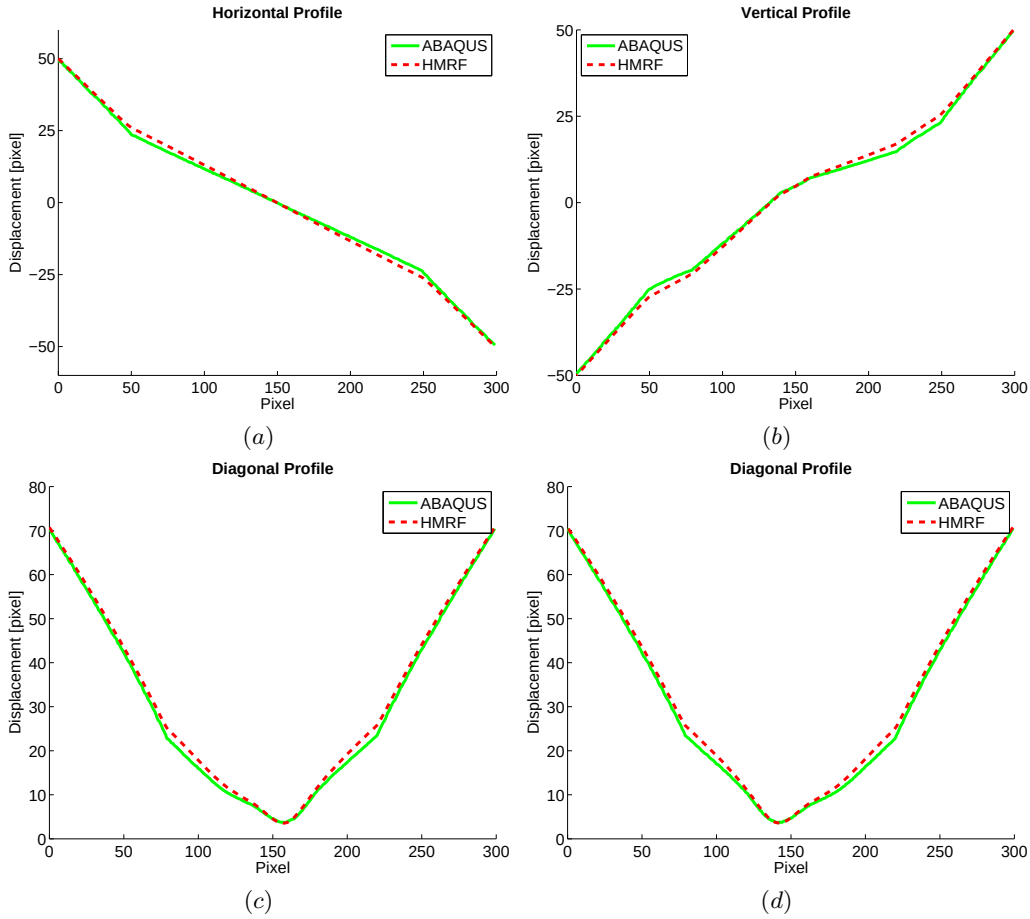


Fig. 5.16. Comparison of ABAQUS and HMRF results with profile images: (a) horizontal profile, (b) vertical profile, (c) diagonal profile from top-left to bottom-right corner and (d) diagonal profile form bottom-left to top-right corner

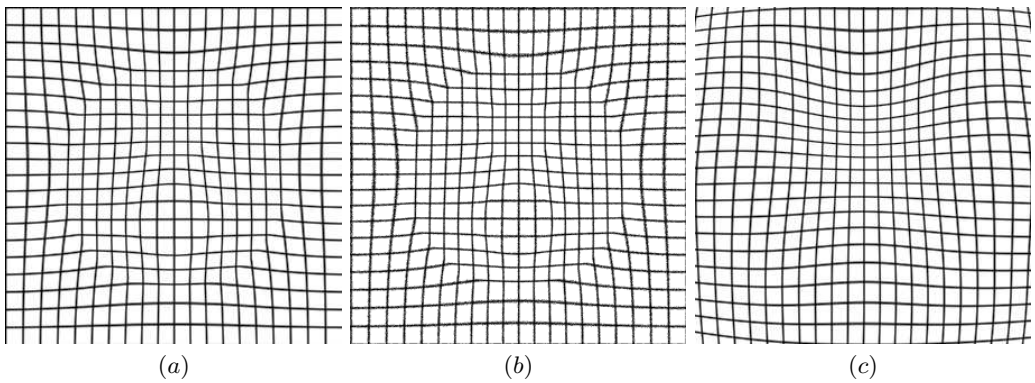


Fig. 5.17. Deformation grids: (a) HMRF method, (b) ABAQUS and (c) elastic registration method

5.12 Circular Geometry with Radial Expansion

The same synthetic data resembling an MRI image of a brain as described in Sect. 5.10 was used for this simulation. In this case however, boundary condition vectors are set normal to the skull to simulate radial displacements.

5.12.1 Test Data

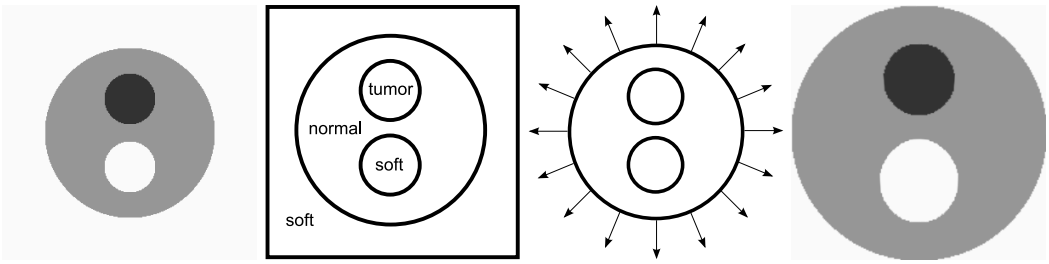


Fig. 5.18. Synthetic MRI brain images: (a) input image, (b) segmentation of the input image, (c) representation of boundary condition image with outward expansion and (d) deformed input image

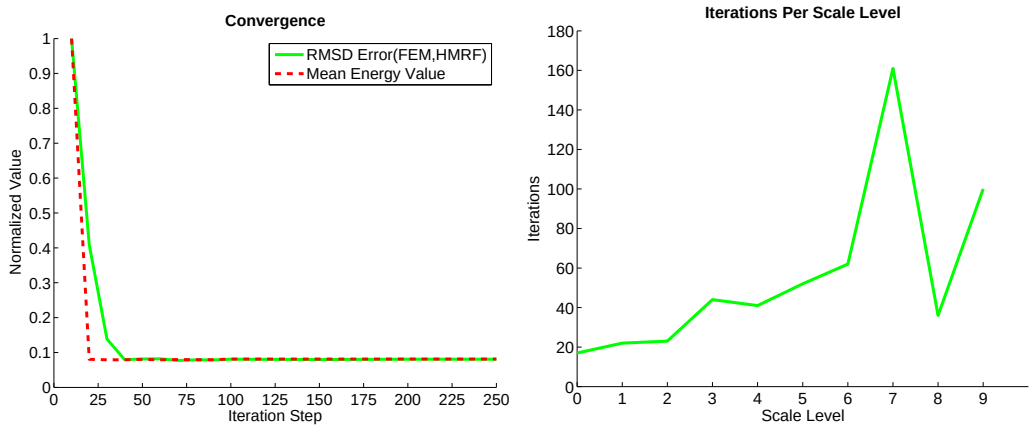
Input image, segmentation image and biomechanical property image are identical with Sect. 5.10. The boundary condition and the confidence image differ as follows. As depicted in Fig. 5.18c the boundary conditions are set to simulate an outward expansion with displacement vectors normal to the outer border of the synthetic brain image. To simulate a real application it is assumed that the area outside of the brain is known, thus the confidence image was set to value 1 in both directions. As mentioned in Fig. 5.10 the boundary condition image can for example be computed using a non-rigid registration algorithm [34, 2], in this case it is a linear interpolation between the skull of the synthetic brain and the border of the image.

5.12.2 Iterations

The same geometry and similar boundary condition cause the interpretations of Fig. 5.19 to coincide with the observations in Sect. 5.10. The same is true for Tab. 5.7.

Table 5.7. Experimental data obtained to evaluate quality drawbacks due to GPU restrictions, 1549 elements of type CPS3 (3-node linear) and CPS4R (4-node bilinear, reduced integration with hourglass control) are used for the FEM calculations

Method	Mean energy value	RMSD error FEM - HMRF	Iterations	Computation time
CPU	31.7299	0.60 pixel	558	39.34 sec.
GPU	32.4421	0.60 pixel	500	1.25 sec.
FEM	-	-	-	0.70 sec.



(a) (b)

Fig. 5.19. Graphs for evaluation of optimal number of iterations: (a) convergence graph with normalized y-axis for better comparability, (b) number of iterations per scale level measured with the CPU implementation

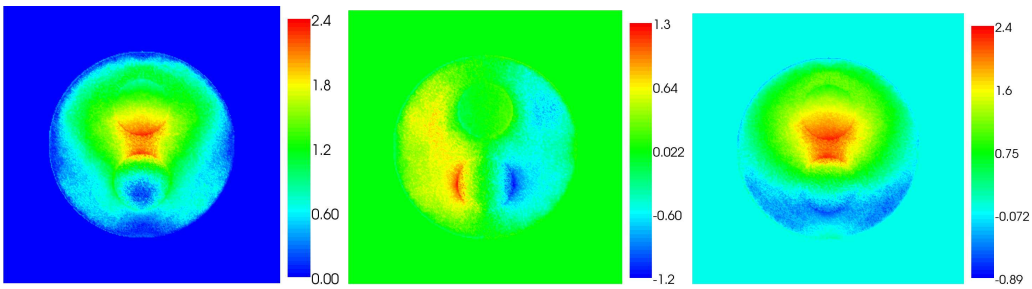
Table 5.8. Comparison of computation times between GPU and CPU implementation with 50 iteration steps

Image Size	Time on GPU	Time on CPU	Speed-up Factor
300 x 300	1.25 sec.	68.45 sec.	54.76
600 x 600	4.90 sec.	270.90 sec.	55.29
1200 x 1200	20.24 sec.	1078.13 sec.	53.27

5.12.3 Computation Time

The computation times in Tab. 5.8 are also very similar to the results obtained in Sect. 5.10.

5.12.4 Difference Image



(a) (b) (c)

Fig. 5.20. ABAQUS and HMRF displacement vector field images differences: (a) magnitude of the vector, (b) x component of the vector and (c) y component of the vector

The fact that more boundary condition data is provided, improved the quality of the simulation. This is shown by the difference images in Figs. 5.20a-c.

5.12.5 Profile Difference

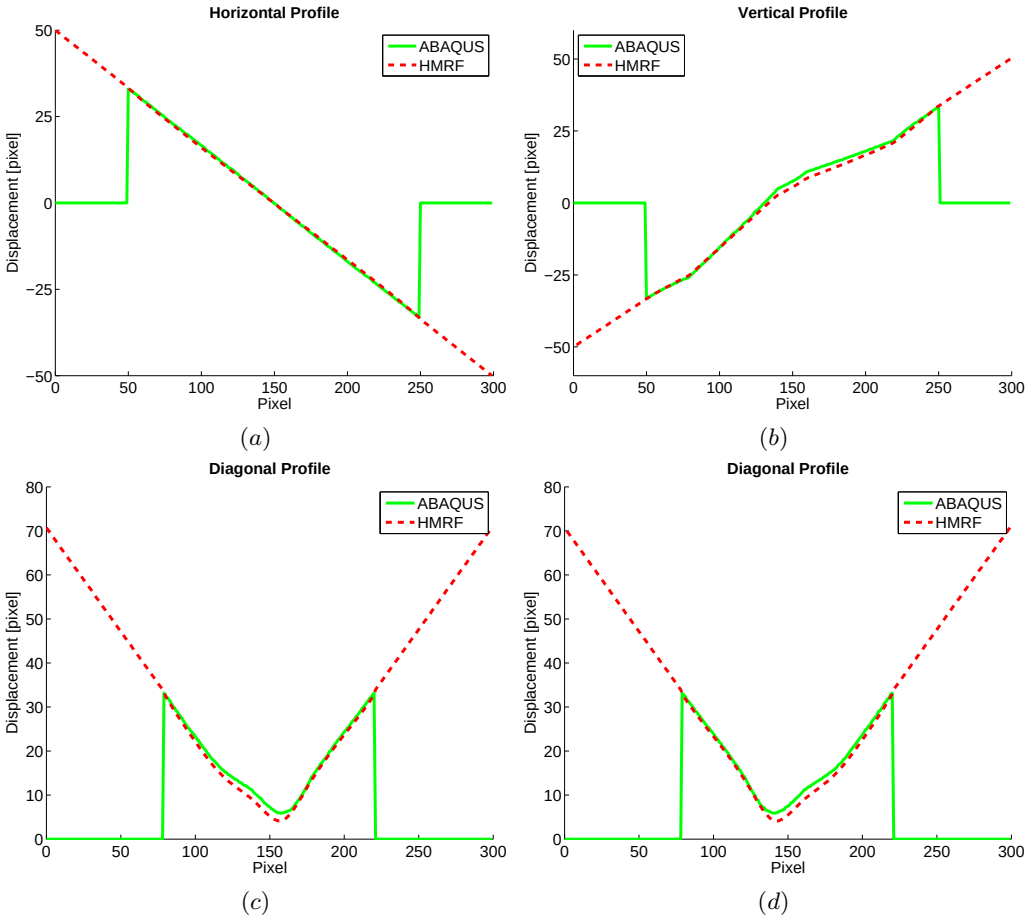


Fig. 5.21. Comparison of ABAQUS and HMRF results with profile images: (a) horizontal profile, (b) vertical profile, (c) diagonal profile from top-left to bottom-right corner and (d) diagonal profile form bottom-left to top-right corner

The results in Fig. 5.21 show an improvement in accuracy if compared to the circular simulation in Sect. 5.10. This proves that the simulation handles boundary conditions inside the image area well. The ABAQUS results show zero displacements outside of the brain. In fact the ABAQUS model was only a circular shape, the outside of the brain was not modeled, so these areas can be neglected in the comparison.

6

Implementation

In this chapter the implementation of HDM is described. Two implementations have been realized, one for the CPU and one for graphics hardware.

6.1 CPU

The pseudo code for the implementation on CPU is listed in Alg. 3. U_{total} is calculated according to (4.5) and Γ is a set of values describing the next estimate, e.g. $\Gamma = \{(0.1, -0.1), (0.001, 0.001), \dots\}$. By varying Γ one can set the step size for the optimization method. Line 3 to 6 are the projection from one scale level to the next. Line 7 to 10 is the ICM minimization. The loop will continue until the full resolution image is reached.

Algorithm 3: Computation on the quadtree (with notations from Fig. 4.4) and optimization with ICM

Data: boundary condition image $\boldsymbol{y}$, confidence image $\boldsymbol{c}$, mechanical property image $\boldsymbol{m}$
Result: displacement vector field $\hat{\boldsymbol{d}}$

```
1  initialize d to null vector field
2  for n=R to 0 do
3      projection of d from scale n+1 to n by linear interpolation
4      projection of y from scale 0 to n by nearest neighbor interpolation
5      projection of c from scale 0 to n by nearest neighbor interpolation
6      projection of m from scale 0 to n by nearest neighbor interpolation
7      repeat
8          foreach site s ∈ S do
9              d_s ← arg min_{e∈Γ}(U_total(y_s, c_s, d_s + e, m_s))
10     until U_total stabilizes
```

In Appendix A more details on the implementation on source code level are provided.

6.2 Graphics Hardware

Even though ICM is used as optimization method, computation of the entire quadtree is expensive. 50 to 100 iterations per scale level are needed to reach convergence. By implementing the HMRF regularization on graphics hardware the computation time can be reduced drastically. Due to the fact that MRF is only looking at neighborhood information it is fairly easy to implement it in a highly parallelized way. In the following sections the framework, implementation details and restrictions are analyzed.

6.2.1 Frameworks

General-Purpose computation on GPUs (GPGPU) is gaining momentum because of new APIs that enable the developer of using the GPU, without requiring to know detailed specifics on the low level hardware functionality. It enables the developer to access all functionalities through a abstract high-level programming interface for an easy and fast implementation. There are many frameworks available for general GPGPU, in this text two are mentioned.

First, the BrookGPU framework implemented at Stanford University as a realization of stream programming [20, 19]. The BrookGPU is a compiler and runtime implementation for the so called Brook program language. The Brook program language is an extension of basic ANSI C providing two new main programming constructs, streams and kernels. The user of BrookGPU can write code in the Brook language to formulate computational tasks with the help of streams and kernels. This enables a straight forward implementation of algorithms for GPUs and reflects the hardware restrictions in terms of software constructs. The written Brook code is compiled with BRCC, the Brook compiler, and generates Cg and C/C++ code. During runtime the Brook RunTime (BRT) library access the right implementation for the current hardware. At the time of writing BrookGPU is still in beta phase.

Another interesting project is Compute Unified Device Architecture (CUDA). It has been developed by Nvidia and requires the latest Nvidia graphics hardware, at least GeForce 8 Series. It provides a growing API with algorithms implemented and ready to use. Essential programming tools like compilers and debuggers are also provided. This helps a lot since debugging on the GPU is a very tedious job. At the time of writing a version 1.1 and 2.0 Beta was available for download.

According to the Roadmap of ITK[1] for 2007/2008, GPU programming is planned, at least a portion of the pipeline.

In the current application an implementation with OpenGL and Cg shader programming was realized. Mainly because the above mentioned frameworks are still in early stages whereas Cg has been around for a while.

6.2.2 Cg

Cg [11] stands for “C for graphics”. It enables developers to use a C-like programming language to write code that can be executed directly on the GPU. The developer does not need to possess any knowledge about the graphics hardware assembly language. One difference to ANSI C is for instance the predefined data types *float2*, *float3* and *float4* which are equivalent to the typedef structs in C:

```
typedef struct {
  float x;
  float y;
} float2;
```

For the current application only the fragment processor of the graphics pipeline is needed for the calculations. A general graphics pipeline is depicted in Fig. 6.1. Fragment processors provide mathematical functionalities similar to the CPU as well as read and write memory capabilities. It provides the developer with everything needed to implement a computer program. In graphics applications the fragment processor is used to calculate pixel colors based on texture information. For HMRF regularization the fragment processor is used to optimize the displacement vector field image based on energy calculation of the texture information. Cg programs are loaded onto the fragment processor and executed.

[1] ITK is an open-source, object-oriented software system for image processing, segmentation, and registration.

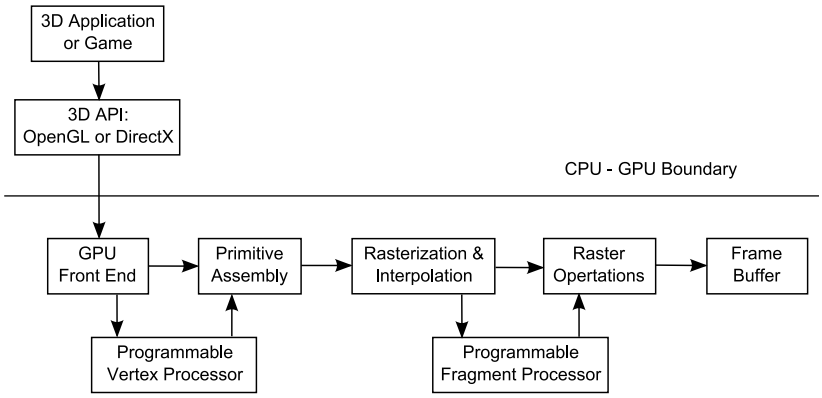


Fig. 6.1. Graphics pipeline showing the interaction between CPU and GPU. The implementation of HMRF is done on the fragment processor.

6.2.3 OpenGL

The Cg language and compiler environment allows the developer to develop and compile a program for the fragment processor. The actual loading and execution of the fragment shader code is done by calling OpenGL functions. The OpenGL draw function triggers the graphics pipeline which in turn calls the Cg program on the fragment processor to write the results to the framebuffer. In graphics applications the content of the framebuffer is shown to the user on the computer screen. In HMRF regularization the content of the framebuffer is copied back to an C/C++ array.

6.2.4 Restrictions

The speed of GPU computation comes with two major restrictions. First, the speed advantage can only be gained by executing many operations at the same time, which requires the algorithm to be parallizable. Second, the texture memory is read-only and the framebuffer is write-only. These restrictions require certain adjustments of the HMRF regularization. In Chapter 5 experiments are conducted to evaluate whether or not the restriction influences the quality of the simulations.

Total Field Energy

The total field energy needs to be computed for convergence checks. To be able to calculate the entire energy field, a copy of the current image back to a C/C++ array in the working memory would be required, this of course is very time consuming. Consequently, the number of iterations need to be predefined before the simulation is started.

Pixel Update During Iteration

Only write-only access to the output buffer means that no update of the pixel during the iteration process can be realized. Updating during iterations is very common in the optimization of MRF models because usually the field converges faster.

Random Access

Because all pixels are calculated at the same time no random access is possible. The technique of random access of pixels during the MRF optimization process is important to prevent the propagation of patterns.

Maximum Arithmetic Operations

Some graphics hardware set a maximum of arithmetic operations that can be executed by the fragment shader program. In the case of the used ATI card the maximum was set to 512 and the actual number of operations needed for the HMRF regularization came very close to this value. With the latest generation of graphics hardware this problem can be neglected because either no restriction is defined by the manufacturer or the maximum is so big that it will never be reached.

6.2.5 OpenGL Extension - EXT_framebufffer_obect

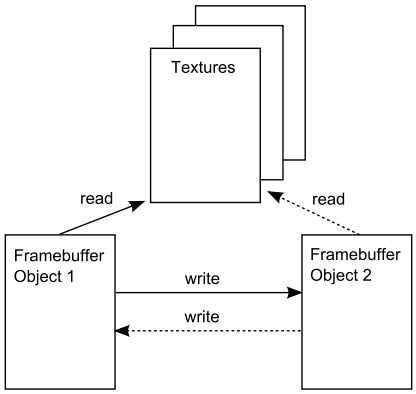


Fig. 6.2. Framebuffer object and texture memory interaction

In [25, 24] the general way of MRF on graphics hardware is presented. For the current implementation a new extension for OpenGL, EXT_framebuffer_object [17, 16], was used. The EXT_framebuffer_object enables alternating between read and write buffers directly without copying the buffers back and forth after one iteration has been completed. In Fig. 6.2 the interaction between framebuffer objects and texture memory is shown. One way is illustrated by plain lines and the other by dashed lines. Only one way is active at a time.

6.2.6 Fragment Shader Program

The shader program has the following function declaration:

```
float4 energy_function(in float2 pos : TEXCOORD0,
 uniform samplerRECT image,
 uniform samplerRECT observation,
 uniform samplerRECT confidence,
 uniform float2 dimension) : COLOR
```

In the following, the parameters are described.

Channels

Images that are saved in the framebuffer object and the texture memory have 4 channels. r (red), g (green), b (blue) and a (alpha). This is because it was intentionally used for graphics purposes. In the current application x and y values are saved instead. This makes no difference both can be represented by *float* values. In Tab. 6.2 the occupation of the channels is described. Channels that are not used in the current implementation are reserved for a future 3D implementation.

Table 6.1. Parameter description of energy shader program

Parameter	Description
in float2 *pos* : TEXCOORD0	The position at which the shader program is loaded. It is of type float2 which is a 2 dimensional array of floats
uniform samplerRECT *image*	Vector displacement image
uniform samplerRECT *observation*	Boundary condition image and mechanical properties image
uniform samplerRECT *confidence*	Confidence image
uniform float2 *dimension*	Dimensions of the vector displacement field image, is used to set boundary values correctly

Table 6.2. Occupation of framebuffer object and texture channels

Entity	Channel	Regularization image
image (framebuffer object)	r	x component of displacement vector field
	g	y component of displacement vector field
	b	not used
	a	not used
observation (texture 1)	r	x component of boundary condition image
	g	y component of boundary condition image
	b	Young‘s modulus
	a	not used
confidence (texture 2)	r	x component of confidence image
	g	y component of confidence image
	b	not used
	a	not used

Boundary values

In imaging frameworks like ITK, access to pixel outside of the image space are handled by designated objects. In the GPU implementation this has to be done manually. For this purpose a parameter *dimension* is passed to the fragment shader program. The program then checks if the current pixel is at the border of the image and assigns appropriate values to pixels that are outside:

```
// left
if(floor(pos.x) == 0) {
 leftPixel.rg = centerPixel.xy;
 leftObs.b = centerObs.b;
}
// right
if(floor(pos.x) == dimension.x-1) {
 rightPixel.rg = centerPixel.xy;
 rightObs.b = centerObs.b;
}
```

6.2.7 Pseudo Code

In Alg. 4 a pseudo code implementation of one scale level of the HMRF regularization is shown. The propagation of scale levels is the same as in the CPU implementation.

Algorithm 4: HMRF using EXT_framebuffer_object

Data: boundary condition image $\boldsymbol{y}$, confidence image $\boldsymbol{c}$, mechanical property image $\boldsymbol{m}$, number of iterations n
Result: displacement vector field $\hat{\boldsymbol{d}}$

1 create two framebuffer objects and label one as read-only and the other as write-only
2 compile, link and load the ICM shader on the GPU
3 initialize $\boldsymbol{d}$ to null vector field
4 copy initial $\boldsymbol{d}$ into framebuffer read-only object
5 specify shader parameters $\boldsymbol{m}$, $\boldsymbol{y}$ and $\boldsymbol{c}$ as texture memory
6 **for** *i=0* **to** n **do**
7 render a rectangle covering a window of size $N \times M$ to write-only framebuffer object
8 switch pointers of write- and read-only framebuffer objects
9 copy the write-only framebuffer object into a C/C++ array

Alg. 4 will be called for each scale level until the finest scale level is reached. To optimize performance the initialization, line 1 to 2, is only done once per scale level.

6.2.8 GPU Debugging

For the development process to be efficient it very important to have debugging tools. On the GPU this is more difficult than on the CPU. There are a few debugging tools available, e.g. [38], but at the time of writing, they are still in a very early phase. The actual debugging was done by means of printing out the values received for every pixel to a text file. At any point of the fragment shader program the following line can be inserted:

```
float4 debug;
...
debug.r = ...;
debug.g = ...;
debug.b = ...;
debug.a = ...;
return debug;
```

This allows for a very simple debugging, and was sufficient for the current program.

6.2.9 Floating-Points Rounding Errors

When results obtained with both implementations are compared, a difference of the order of 10^{-2} can be detected. This can be explained by different floating point arithmetic standards between GPU and CPU [22]. All major CPU manufacturer follow the IEEE 754-1985 standard for floating point arithmetic whereas there is no such agreement between the GPU manufacturer.

6.2.10 Speed-Up

In Chapter 5 speed-up factors are calculated for four different simulations. The GPU implementation is between 50 to 75 times faster than the CPU implementation. This factor can most likely be further improved by using the latest generation of graphics hardware.

6.3 Comparison of CPU and Graphics Hardware Implementation

On a pseudo code level the only difference between the two implementations is the ICM part. In the CPU version ICM is done until U_{total} stabilizes and in the GPU version ICM is done until a fixed number of iteration is reached. On the source code level the energy function implementations show minor difference due to the fact that Cg is an extension of ANSI C code.

7

Field of Application

In this chapter three potential applications of HDM are presented.

7.1 Non-Rigid Registration of Anatomical Structures on Region of Interest

As mentioned in Chapter 2, conventional image registration methods do not take biomechanical properties into account. Therefore, one possible application for HDM is to support existing registration methods in areas of the image where the lack of biomechanical information results in unrealistic prediction of the deformation. In Alg. 5 pseudo code is presented. The confidence image is constructed prior to the registration process. The confidence image can be constructed manually or automatically. When constructed automatically, a certain segmented area is chosen to be the region of interest (ROI). The mechanical property image of the source image needs to be available prior to the registration as well. On line 1 a conventional non-rigid registration method is executed, which is then used as input for HDM.

Algorithm 5: HDM on ROI

Data: source image s, target image t, confidence image $\boldsymbol{c}$ for s (representing ROI), mechanical property image $\boldsymbol{m}$ of s
Result: displacement vector field $\hat{\boldsymbol{d}}$

1 $\mathbf{y} \leftarrow$ non-rigid registration with conventional method
2 initialize $\boldsymbol{d}$ to null vector field
3 run HDM($\boldsymbol{y}$,$\boldsymbol{c}$,$\boldsymbol{d}$,$\boldsymbol{m}$)

7.2 Iterative Non-Rigid Registration of Anatomical Structures

The second proposed application is the registration of an entire image. The idea is to use an iterative process, in which new estimates about the displacement are made and compared with corresponding points, e.g. edges from the segmented image. In Alg. 6 a pseudo code implementation is shown. In contrast to Alg. 5 no boundary conditions and confidence image are available, an iterative method is implemented to estimate the right boundary conditions and its confidence. The deformation conform to the biomechanical properties of the source image.

Algorithm 6: HDM for entire image registration

Data: source image s, target image t, mechanical property image $\boldsymbol{m}$ of s
Result: displacement vector field $\hat{\boldsymbol{d}}$

initialize boundary condition image $\boldsymbol{y}$
initialize confidence image $\boldsymbol{c}$
initialize $\boldsymbol{d}$ to null vector field
repeat
  new estimate $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{e_y}$
  new estimate $\boldsymbol{c} \leftarrow \boldsymbol{c} + \boldsymbol{e_c}$
  $\boldsymbol{d} \leftarrow$ run HDM($\boldsymbol{y}$,$\boldsymbol{c}$,$\boldsymbol{d}$,$\boldsymbol{m}$)
  $\hat{s} \leftarrow$ apply $\boldsymbol{d}$ to s
  check correspondence of $\hat{s}$ and t
until correspondence of $\hat{s}$ and t converged

7.3 Iterative Tumor Detection

Structural changes in soft tissue can provide valuable information and imply medical consequences. Tumors are not alway visible in MRI images. One approach of finding invisible tumors is by taking multiple images of the same patient over time. With the presented Alg. 7 it is then possible to identify hidden tumors that cause differences in underlying material properties.

A concrete scenario is presented for illustration purposes: A brain MRI image s of a patient is taken. Another MRI image t of the same patient is taken 6 month later. A registration of both images is performed with Alg. 5. The registration quality is not satisfactory. This indicates a change of material properties not reflected by pixel intensity and thus the mechanical properties image need to be corrected. Alg. 7 optimizes the correspondence between image s and image t by correcting the mechanical property image.

The validation of this application could be done with multi-sequence MRI segmentation methods presented in [9] and [10].

Algorithm 7: HDM tumor detection on ROI

Data: source image s, target image t, confidence image $\boldsymbol{c}$ for s (representing ROI), mechanical property image $\boldsymbol{m}$ of s
Result: corrected mechanical property image $\hat{\boldsymbol{m}}$

$\boldsymbol{y} \leftarrow$ registration with Alg. 5
initialize $\boldsymbol{d}$ to null vector field
repeat
  new estimate $\boldsymbol{m} \leftarrow \boldsymbol{m} + \boldsymbol{e}$ within ROI
  run HDM($\boldsymbol{y}$,$\boldsymbol{c}$,$\boldsymbol{d}$,$\boldsymbol{m}$)
  $\hat{s} \leftarrow$ apply $\boldsymbol{d}$ to s
  check correspondence of $\hat{s}$ and t
until correspondence of $\hat{s}$ and t converged

8

Conclusion

A method to simulate soft tissue deformations in the image space using physical-based concepts and image processing techniques was presented. The method combines mechanical concepts into a Bayesian optimization framework which has been defined and solved under a HMRF approach and implemented for the CPU and the GPU. The main advantages of the proposed methodology over previous ones is its mesh-free characteristic, which is normally needed to perform accurate mechanical simulations of tissues. The use of HMRF proves to be an appealing technique to solve the proposed stochastic problem. The reasoning is twofold: On one hand it was found that local minima are avoided by using a hierarchical approach, which in turn allows for a speed-up since it is now possible to use local optimizers rather than slow global optimizers (e.g. Markov Chain Monte Carlo Simulated Annealing optimization). Secondly, the nature of the HMRF approach resulted in a straightforward implementation in the GPU, which has been remarked by others [25, 24].

The simulations in Chapter 5 presented five scenarios. On the one hand, different geometries are tested. In the first three simulations square shapes are deformed and in the other two circular shapes, which resemble a brain MRI. It can be stated that for these particular cases circular shapes show better results than squared shapes, but more experiments are required to investigate the reasons of this. On the other hand, different types of boundary conditions are applied to the images. One type of boundary condition describes displacements at the border of the image. Interesting for real world application is the type used in the last experiment, where boundary conditions are also defined inside of the image. The aim was to see whether the model could handle the information provided inside of the image. One concern was that the model would over-smooth the solution, because the hierarchical approach would interpolate the non-boundary condition area too strongly. It was shown that this is not the case. The comparison with non-rigid registrations presented the major advantage of using biomechanical information during the registration process. At the interfaces of tissues the non-rigid method results were not realistic whereas the interfaces were preserved with HDM. Overall the results obtained with HDM over-smoothed at the interfaces, but not as much as the non-rigid registration. However, the HDM deformations correlated with the ABAQUS results.

The computation times compared to FEM models are of the same order of magnitude for simple shapes. In cases where more complex structures are modeled, HDM computation times will not differ much from the computations on simple structures, because the number of elements are the number of pixels. In more complex structures FEM models need more elements resulting in more computation time.

In Chapter 7, a short overview of possible application was given. HDM is certainly interesting for the application in image registration. But also other fields of application could be thought of, maybe even the replacement of FEM for cases were only approximation are needed. This is the case in real-time application. When the latest graphics

hardware can be used the computation time of the model could most likely be reduced to less than one second for images of size 300×300 pixel.

In Chapter 9, restrictions were discussed and ideas to resolve them were presented. One of the most crucial steps that need to be taken in the future is the validation of HDM on non-synthetic data. For this task, it is important to have a reliable ground truth to compare the results, which is not trivial.

By using HRMF at the GPU level, stable results can be attainable in a very reasonably computation time, simplifying the soft-tissue-deformation simulation pipeline. Making this method very appealing for medical applications.

9

Future Work

The model is implemented to provide a good starting point for the integration of future research projects. It is easy to extend and to build upon. However, there are restrictions in the current model that need to be addressed. In the following, restrictions and possible future research projects to resolve them are described. In addition, future extensions to improve the quality of the model are presented.

9.1 Non-Synthetic Data

In the future, experiments with more complex geometry and non-synthetic image data need to be conducted. A source for MRI brain images and corresponding manual segmentation is provided by the Center for Morphometric Analysis at Massachusetts General Hospital and is available at http://www.cma.mgh.harvard.edu/ibsr/. An example of such an MRI and segmented image is shown in Chapter 3 in Fig. 4.2.

9.2 3D Simulations

In the current implementation only 2D models are considered. The developed model is theoretically expandable to 3D without making major changes. Special attention for the transition are to be set on the performance and possible quality optimization between layers.

9.3 Porting of Cg Code

An implementation with OpenGL and Cg shader programming was realized. In the future a transfer to the Brook, project from Stanford University, or Cuda, NVIDIA, framework is to be considered.

9.4 Variable Number of Hierarchical Levels

The results in Chapter 5 showed that a strong influence of the number of hierarchical levels on the number of iterations and the quality of the output exists. To account for this, a model to automatically set the optimal number of hierarchy levels is desirable. The use of sparse information of the image to determine the quality of hierarchy scale levels [40] could be interesting. When the quality does not reach a predefined constant the scale level is skipped. This should make the model more robust and could prevent over-smoothing.

9.5 Initial Displacement Vector Field

In the current model the initial displacement vector field is a null vector field. The creation of an initial guess based on mutual information [34], consistence image registration [7] and Riemann spaces [32] should be subject for future research projects.

9.6 Resolve Restrictions on Graphics Hardware

The massive parallel structure of graphics hardware comes with certain restrictions.

9.6.1 Total Field Energy

To establish a dynamic stop criteria the computation of the total field energy is needed. One approach to investigate is to solve the iteration issues by introducing convergence checks after a certain number of iterations. This would require to copy the image to an C/C++ array. On the one hand, the copying slows down the computations but on the other hand, it reduces the number of iterations needed to converge to the solution. The check interval can further be optimized by a scheduling algorithm, which would schedule the checks according to some sophisticated learning or statistical model.

At the moment only 2D simulations are done so the number of pixels involved for a simulation is rather small and the time gain achieved by intermediate checks is to small to compensate for the complexity of such an algorithm.

9.6.2 Pixel Update During Iteration

1	2	1	2	1	2
3	4	3	4	3	4
1	2	1	2	1	2
3	4	3	4	3	4
1	2	1	2	1	2
3	4	3	4	3	4

Fig. 9.1. Coding of the image plane for a 8-neighborhood structure

In [28] a coding system is established to divide an image into different sets of pixels. Each set of pixels is then executed separately whereas within each set all elements can be executed simultaneously. In Fig. 9.1 a coding scheme for an $c = 2$ neighborhood system is depicted. The number 1 to 4 label the different sets. The results in Chapter 5 indicate that for an ICM optimization the coding system is not necessary. If the ICM is replaced by an stochastic optimization the coding system would be important based on [3] in which the theoretical convergence properties of parallel versions of stochastic relaxation algorithms are studied.

9.6.3 Random Access

Random access of pixel is only important when pixel update during iteration is used. In the current implementation this is not the case.

9.7 Global Energy Optimization

The implementation of HDM on graphics hardware enables new opportunities to use more computational expensive strategies to solve the energy minimization problem. One of which is to use a stochastic optimizer like MCMCSA method rather than a local deterministic optimizer. Furthermore is the field of stochastic optimization a strong field of research and the consideration of new methods, e.g. the work presented in [23], are interesting. A prerequisite is the realization of a dynamic stop criteria for the GPU implementation, otherwise the computational workload would be enormous.

9.8 Continuous Confidence Images

To account for bad boundary condition image input, e.g. when the boundary conditions are calculated with an automatic registration algorithm that provides bad corresponding points, a continuous confidence image could be realized. The HDM then improves or corrects the provided boundary conditions during the modelling process. Continuous values between 0 and 1 are then taken as a confidence probability at a certain pixel location. This idea could be extended to a distribution over multiple pixels, e.g. a given boundary condition can move along a border, by defining anisotropic probability functions.

9.9 Principle Component Analysis

As it has been shown in Chapter 5 the number of iterations per scale level is very important to optimize speed and quality of the model. At the current state the estimation of the right number of iteration is done with an heuristic approach. To automate this procedure, principle component analysis shows interesting properties. The idea is to learn from a set of geometric shapes. This can be done by computing a database of shapes and note the numbers of iteration per shape. Before a simulation is started, this information can be used to relate the shape under study to the shapes used during the learning process. The weighting of the eigenvectors that are needed to create the shape under study is then used as a parameter to set the optimal number of iterations.

The realization of the explained improvement is only relevant as long as no dynamic stop criteria is available.

9.10 Non-Linear Material Laws

The current model uses the linear elastic material law $E = \sigma/\varepsilon$. Biological tissues usually are explained much better with hyperelastic (e.g. exponential) laws. In order to add non-linear material behavior the basic principles of continuum mechanics need to be adapted to prior energy functions.

9.11 Anisotropy

In the current model the material reacts the same in all directions. Anisotropic materials show different behavior depending on the directions. Anisotropic properties could be introduced by using a vector image for the biomechanical properties instead of the scalar image. Each component can then be assigned different Young‘s modulus values.

A

Software

The software developed for the thesis is realized in C++ with the help of Insight Segmentation and Registration Toolkit (ITK) and Visualization ToolKit (VTK). Plotting of graphs and visualization of the displacement vector field images is done with MATLAB and Paraview. In Tab. A.1 the framework and software versions are listed.

Table A.1. Software and Framework versions

Item	Value
ITK	3.2.0
VTK	5.0.3
MATLAB	R2006b
Paraview	3.2.1

ITK is an open-source, object-oriented software system for image processing, segmentation, and registration. It provides a large variation of state of the art algorithms implemented for optimal speed performance. The framework is implemented for the major operating systems on the market.

VTK is an open source, freely available software system for 3D computer graphics, image processing, and visualization. Its implementation follows the object-oriented principles. The Framework can be used cross platform.

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

ParaView is an open-source, multi-platform application designed to visualize data sets of size varying from small to very large. The goals of the ParaView project include the following: Develop an open-source, multi-platform visualization application; support distributed computation models to process large data sets; create an open, flexible, and intuitive user interface; develop an extensible architecture based on open standards.

In the following sections a list of software programs that have been developed during the thesis is given.

A.1 AbaqusToImage

This software program is developed to help parsing the ABAQUS results to an ITK image. At the moment the conversion is done by meshing the ABAQUS model with a high number of elements. The interpolation is done with a nearest neighbor approximation of the nodes of the mesh to the closest point on the image grid. To account for points

that are not defined in the grid image the concept of binary image was introduced. The binary image is used later when comparison to other results are conducted. The binary image defines whether a pixel on the grid is available or not. In Fig. A.1 the interpolation is shown.

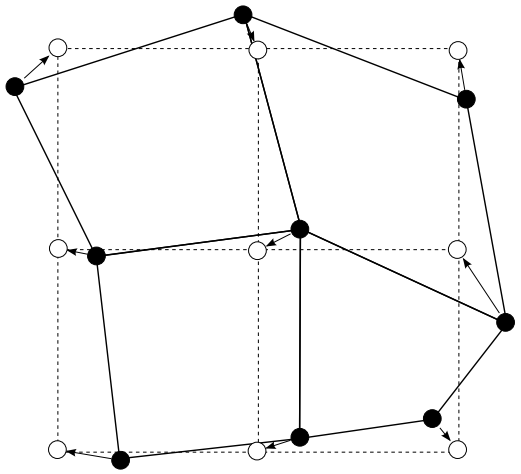


Fig. A.1. Nearest neighbor interpolation for the ABAQUS result (black points) to the regular grid points of an image (white points)

Another way of calculating the displacement vectors at the nodes of the mesh is by using the shape function of FEM.

This is a rough description of the problem and is only provided as a guideline for future work. The following shape functions are used for QUAD4 elements [37]:

$$h_1(r,s) = \frac{1}{4}(1-r)(1-s) \tag{A.1}$$

$$h_2(r,s) = \frac{1}{4}(1+r)(1-s) \tag{A.2}$$

$$h_3(r,s) = \frac{1}{4}(1+r)(1+s) \tag{A.3}$$

$$h_4(r,s) = \frac{1}{4}(1-r)(1+s), \tag{A.4}$$

where (r,s) are coordinates of the unit space. Pixel sites are in the global coordinate system, thus the relation between the global and the unit space is required to calculate unit coordinates for pixel sites:

$$\begin{bmatrix} x(r,s) \\ y(r,s) \end{bmatrix} = \begin{bmatrix} x_1h_1(r,s) + x_2h_2(r,s) + x_3h_3(r,s) + x_4h_4(r,s) \\ y_1h_1(r,s) + y_2h_2(r,s) + y_3h_3(r,s) + y_4h_4(r,s) \end{bmatrix}, \tag{A.5}$$

where (x,y) are coordinates in the global space. In Fig. A.2 an example of unit and global coordinate systems is shown. Once the unit coordinates are calculated they can be inserted into

$$\begin{bmatrix} u(r,s) \\ v(r,s) \end{bmatrix} = \begin{bmatrix} u_1h_1(r,s) + u_2h_2(r,s) + u_3h_3(r,s) + u_4h_4(r,s) \\ v_1h_1(r,s) + v_2h_2(r,s) + v_3h_3(r,s) + v_4h_4(r,s) \end{bmatrix}, \tag{A.6}$$

where (u,v) are the displacements at coordinate (r,s). Both transformation, the distorted global space to unit space transformation, and the transformation to get the displacements at a certain point, are the same. This is because both represent a distortion of the meshing system.

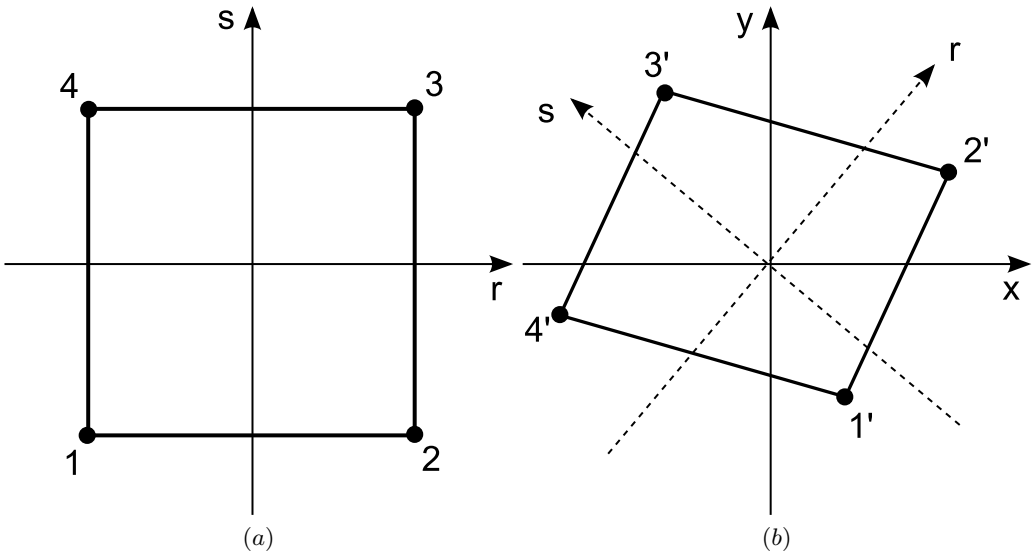


Fig. A.2. QUAD4 elements in (a) unit coordinates system and (b) global cooridnate system

A.2 ImageJToImage

The plugin bUnwarpJ [2] for ImageJ [1] is used as an elastic registration algorithm for comparison. It is possible to extract a displacement vector field from bUnwarpJ. Each point on the image is assigned a point to which it is translated. This is a different format than the one used with ITK. In ITK each site is assigned with a vector which describes the displacement of that site. To overcome the difference this small program converts displacement vector fields obtained from bUnwarpJ to displacement vector fields usable with ITK.

A.3 BCPreperation

BCPreperation is a command line tool to generate boundary condition images. At the moment there are three predefined images that are generated.

A.4 DVFRegularization

Is the actual implementation of the HMRF regularization model. It provides the option of running either the GPU or the CPU version. The following classes have been developed to be integrated to the ITK framework. It allows users of ITK to easily test the developed methods by using the standard ITK pipeline.

A.4.1 itk::MRFRegularizationFilter

This is an implementation of an ITK filter and extends the class *itk::ImageToImageFilter*. To illustrate the usage of this class a code snippet is presented:

```
typedef itk::MRFRegularizationFilter<...> MRFFilter;
MRFFilter::Pointer mrfFilter;
mrfFilter = MRFFilter::New();
```

```
/* set the maximum number of iterations
 * in case stop criteria is not reached */
mrfFilter->SetMaximumNumberOfIterations(50);
// flag to enable pixel update during iteration
mrfFilter->SetDuringIterationUpdate(true);
/* stop criteria is the root mean square deviation of the
 * current and previous energy field */
mrfFilter->SetStopCriteria(0.0001);
// set the energy function
mrfFilter->SetEnergyFunction(...);
// set the segmentation image
mrfFilter->SetLabels(...);
// set the boundary conditon image
mrfFilter->SetObservationImage(...);
// set the condifence image
mrfFilter->SetConfidenceImage(...);
// set the displacement vector field image to be regularized
mrfFilter->SetInput(...);
// call update to excecute the filter
mrfFilter->Update();
```

A.4.2 itk::EnergyFunction

The definition of the energy function for the CPU version of the code is done in the abstract class *itk::EnergyFunction*. The following two functions need to be overwritten when extended:

```
/* abstract functions */
virtual EnergyValueType PriorEnergy(const ImagePixelType newCenterPixel,
  const NeighborhoodIterator<TImage > &dispIter,
  const NeighborhoodIterator<TNLabelImage > &labelIter,
  const NeighborhoodIterator<TImage > &obsIter,
  const NeighborhoodIterator<TImage > &confidenceIter) const = 0;
virtual EnergyValueType ObservationEnergy(const ImagePixelType newCenterPixel,
  const NeighborhoodIterator<TImage > &dispIter,
  const NeighborhoodIterator<TNLabelImage > &labelIter,
  const NeighborhoodIterator<TImage > &obsIter,
  const NeighborhoodIterator<TImage > &confidenceIter) const = 0;
```

A.4.3 itk::YoungPoissonEnergyFunction

The concrete implementation of the presented HMRF regularization based on Young's modulus is done in *itk::YoungPoissonEnergyFunction*.

A.4.4 itk::MRFRegularizationFilterGPU

A second filter for MRF regularization was implemented which executes the computationally expensive code on the GPU. The ITK filter extends from the class *itk:ImageToImageFilter*. A possible configuration of the filter is shown in a code snippet:

```
typedef itk::MRFRegularizationFilterGPU<...> MRFFilterGPU;
MRFFilterGPU::Pointer mrfFilterGPU;
mrfFilterGPU = MRFFilterGPU::New();
// the type of the GPU is set with "ati=0" or "nvidia=1"
```

```
mrfFilterGPU->SetGPUType(0);
// set the stop criteria
mrfFilterGPU->SetNumberOfIterations(50);
// set the segmentation image
mrfFilterGPU->SetLabels(...);
// set the boundary conditon image
mrfFilterGPU->SetObservationImage(...);
// set the condifence image
mrfFilterGPU->SetConfidenceImage(...);
// set the displacement vector field image to be regularized
mrfFilterGPU->SetInput(...);
// call update to excecute the filter
mrfFilterGPU->Update();
```

A.4.5 itk::InterpolateNextStepFilter

The class *itk:InterpolateNextStepFilter* is an extension for *itk:ResampleFilterType* to allow vector images to be interpolated. At the moment ITK only support the interpolation of scalar images. The code snipped to illustrate its usage:

```
typedef InterpolateNextStepFilter<...> NextStepFilter;
NextStepFilter::Pointer nextStepFilter;
nextStepFilter = NextStepFilter::New();
/* set the target number of pixel,
 * at the moment only N x N are allowed */
nextStepFilter->SetNumberOfPixels(currentPixelCount);
// set the image to interpolate
nextStepFilter->SetInput(...);
// call update to execute the filter
nextStepFilter->Update();
```

A.5 EnergyFunctionPlot

To be able to analyze energy functions it is important to visualize a subspace of possible values. The generation of the values is realized with C++ and the visualization is done in MATLAB. In Fig. A.3 a visualization of an image with two pixels is shown. For the two pixels a range of values are calculated with a given energy function. In the visualization it can then be seen whether there exists a local or global minimum. In Fig. A.3 a global minimum is reached when pixel 1 an 2 show an approximate value between 25 and 30. Only scalar images can be investigated with this tool.

A.6 ImagePreperation

In this program synthetic input images are generated. At the moment the input images represent simple geometrical objects like circles and squares. Also the corresponding segmentation images are created.

A.7 ParameterEstimation

A parameter estimation algorithm was implemented to find the optimal parameter α in

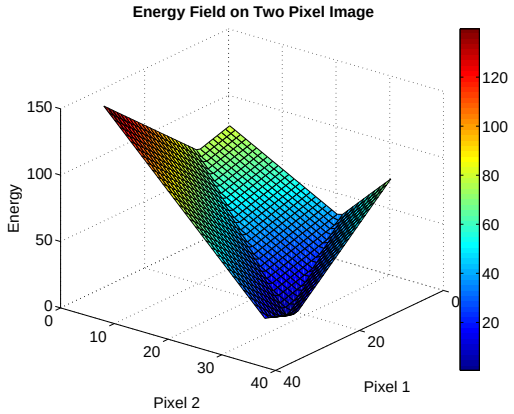


Fig. A.3. Energy function calculations for a range of values for pixel 1 and 2 of an image with two pixels

$$U_{\text{total}} = (1-\alpha)U_{\text{observation}} + \alpha U_{\text{prior}}, \tag{A.7}$$

where α is the weighting between observation and priori energy function. Simplex optimization and MCMCSA method were used to solve the minimization problem

$$\hat{\alpha} = \arg\min_{\alpha} \sqrt{\frac{1}{|S|}\sum_{s\in S}|\mathbf{a_s} - \hat{\mathbf{d}}_\mathbf{s}(\boldsymbol{\alpha})|^2}, \tag{A.8}$$

where $\hat{\mathbf{d}}_\mathbf{s}(\boldsymbol{\alpha})$ is the result of a MRF regularization with α for U_{total} and $\mathbf{a_s}$ is the ground truth vector obtained from ABAQUS.

A.7.1 Markov Chain Monte Carlo Simulated Annealing

Represents an implementation of the minimizer illustrated with Alg. 1. In this case S is not a set of sites that represent an image with pixels, but it is a 1D chain of numbers. The minimization can be implemented in the same way, there are only semantical differences.

A.7.2 Simplex

itk::SingleValuedCostFunction was implemented with the MRF regularization, then the *itk::AmoebaOptimizer* optimizer was used to find the minimum. *itk::AmoebaOptimizer* uses Nelder-Meade downhill simplex method, for most problems, it is not the fastest minimizer provided by ITK, but it can perform much better on noisy error functions. Because time is no issue for the parameter estimation, this algorithm was chosen.

A.8 ScaleInputImages

For experiments to compare the performance between the GPU and CPU implementation for different images sizes, a program was written to scale images. Input image, biomechanical property image, boundary condition image, confidence image and ABAQUS result images are scaled to the appropriate size.

B

Specification of Hardware and Driver

The implementation was developed and tested on a computer equipped with the following hardware and driver software:

Table B.1. Graphics hardware

Item	Value
Graphics Card Manufacturer	ATI
Graphics Chipset	Radeon X1300/X1550 Series
Bus Type	PCI Express
Memory Size	256 MB
Memory Type	DDR2
Core Clock	594 MHz
Memory Clock	396 MHz

Table B.2. Graphics driver

Item	Value
Driver Packaging Version	8.471-080225a1-059746C-ATI
Catalyst©Version	08.3
Provider	ATI Technologies Inc.
2D Driver Version	6.14.10.6783
Direct3D Version	6.14.10.0567
OpenGL Version	6.14.10.7412

Table B.3. CPU hardware

Item	Value
Manufacturer	Intel©
Type	Core^{TM}2 Duo Desktop Processor E6300
CPU Speed	1.86 GHz
Bus Speed	1066 MHz

Table B.4. RAM hardware

Item	Value
Manufacturer	Kingston©
Type	ValueRAM
Speed	800 DDR2
Size	2 × 1 GB

References

1. M. D. Abramoff, P. J. Magelhaes, and S. J. Ram. Image processing with ImageJ. *Biophotonics Int*, 11(7):36–42, 2004.
2. Ignacio Arganda-Carreras, Carlos Sorzano, Roberto Marabini, José Carazo, Carlos Ortiz-De-Solorzano, and Jan Kybic. Consistent and elastic registration of histological sections using vector-spline regularization. In *Computer Vision Approaches to Medical Image Analysis*, pages 85–95. 2006.
3. R. Azencott. *Simulated Annealing: Parallelization Techniques*. Wiley-Interscience, 1992.
4. F. Bachtar, X. Chen, and T. Hisada. Finite element contact analysis of the hip joint. *Medical and Biological Engineering and Computing*, 44(8):643–651, 2006.
5. S. K. Boyd and R. Müller. Smooth surface meshing for automated finite element model generation from 3D image data. *Journal of Biomechanics*, 39(7):1287–1295, 2006.
6. Kup S. Choi, Hanqiu Sun, and Pheng A. Heng. Interactive deformation of soft tissues with haptic feedback for medical learning. *Information Technology in Biomedicine, IEEE Transactions on*, 7(4):358–363, 2003.
7. G. E. Christensen and J. He. Consistent nonlinear elastic image registration. In *IEEE Workshop on Mathematical Methods in Biomedical Image Analysis*, pages 37–43, 2001.
8. S. de Putter, F. N. van de Vosse, F. A. Gerritsen, F. Laffargue, and M. Breeuwer. Computational mesh generation for vascular structures with deformable surfaces. *International Journal of Computer Assisted Radiology and Surgery*, 1(1):39–49, 2006.
9. G. Dugas-Phocion, M. A. Gonzalez, C. Lebrun, S. Chanalet, C. Bensa, G. Malandain, and N. Ayache. Hierarchical segmentation of multiple sclerosis lesions in multi-sequence MRI. In *International Symposium on Biomedical Imaging: From Nano to Macro (ISBI'04)*, Arlington, VA, USA, 2004. IEEE.
10. G. Dugas-Phocion, M. A. Gonzalez, G. Malandain, C. Lebrun, and N. Ayache. Improved EM-based tissue segmentation and partial volume effect quantification in multi-sequence brain MRI. In *Proc. of MICCAI'04*, volume 3216 of *Lecture Notes in Computer Science*, pages 26–33, Saint-Malo, France, 2004. Springer.
11. Randima Fernando and Mark J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
12. A. Frisoli, L. F. Borelli, C. Stasi, M. Bellini, C. Bianchi, E. Ruffaldi, G. Di Pietro, and M. Bergamasco. Simulation of real-time deformable soft tissues for computer assisted surgery. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 1(1):107–113, 2004.
13. D. Geman. Random fields and inverse problems in imaging. *In Saint-Flour Lectures 1988, Lecture Notes in Mathematics*, pages 113–193, 1990.
14. D. Geman and G. Reynolds. Constrained restoration and the recovery of discontinuities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(3):367–383, 1992.
15. Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.
16. D. Göddeke. GPGPU basic math tutorial. Technical report, FB Mathematik, Universität Dortmund, 2005.

17. S. Green. The OpenGL framebuffer object extension. *NVIDIA Corporation, GDC*, 2005.
18. M. A. Greminger and B. J. Nelson. Modeling elastic objects with neural networks for vision-based force measurement. In *IROS 2003*, 2003.
19. J. Gummaraju, M. Erez, J. Coburn, M. Rosenblum, and W. J. Dally. Architectural support for the stream execution model on general-purpose processors. In *Parallel Architecture and Compilation Techniques, 2007. PACT 2007. 16th International Conference on*, pages 3–12, 2007.
20. Jayanth Gummaraju and Mendel Rosenblum. Stream programming on general-purpose processors. In *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, Barcelona, Spain, 2005.
21. D. L. G. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes. Topical review: Medical image registration. *Physics in Medicine and Biology*, 46(3):R1–R45, 2001.
22. K. Hillesland and A. Lastra. GPU floating-point paranoia. In *ACM Workshop on General Purpose Computing on Graphics Processors*, pages C–8, 2004.
23. L. Ingber. Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics*, 25, 1996.
24. Pierre M. Jodoin and Max Mignotte. Markovian segmentation and parameter estimation on graphics hardware. *Journal of Electronic Imaging*, 15(3), 2006.
25. Pierre-Marc Jodoin, Jean-Franois St-Amour, and Max Mignotte. Unsupervised Markovian segmentation on graphics hardware. *Pattern Recognition and Image Analysis*, pages 444–454, 2005.
26. Charles Kervrann and Fabrice Heitz. A hierarchical Markov modeling approach for the segmentation and tracking of deformable shapes. *Graphical models and image processing: GMIP*, 60(3):173–195, 1998.
27. J. B. A. Maintz and M. A. Viergever. A survey of medical image registration. *Medical Image Analysis*, 2(1):1–36, 1998.
28. E. Mémin, F. Heitz, and F. Charot. Efficient parallel nonlinear multigrid relaxation algorithms for low-level vision applications. *J. Parallel Distrib. Comput.*, 29(1):96–103, 1995.
29. A. Mohamed, E. I. Zacharaki, D. Shen, and C. Davatzikos. Deformable registration of brain tumor images via a statistical model of tumor-induced deformation. *Med Image Anal*, 10(5):752–763, October 2006.
30. V. Murino, U. Castellani, and A. Fusiello. Disparity map restoration by integration of confidence in Markov random fields models. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, volume 2, pages 29–32 vol.2, 2001.
31. Rasmus R. Paulsen. *Statistical Shape Analysis of the Human Ear Canal with Application to In-the-Ear Hearing Aid Design.* PhD thesis, Technical University of Denmark, 2004.
32. X. Pennec. Left-invariant Riemannian elasticity: a distance on shape diffeomorphisms. *Proceedings of International Workshop on the Mathematical Foundations of Computational Anatomy*, pages 1–13, 2006.
33. J. N. Provost, C. Collet, P. Rostaing, P. Pérez, and P. Bouthemy. Hierarchical Markovian segmentation of multispectral images for the reconstruction of water depth maps. *Comput. Vis. Image Underst.*, 93(2):155–174, 2004.
34. D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes. Non-rigid registration using free-form deformations: application to breast MR images. *IEEE Transactions on Medical Imaging*, 18(8):712–721, 1999.
35. I. A. Sigal, M. R. Hardisty, and C. M. Whyne. Mesh-morphing algorithms for specimen-specific finite element modeling. *Journal of Biomechanics*, 2008.
36. C. O. S. Sorzano, P. Thevenaz, and M. Unser. Elastic registration of biological images using vector-spline regularization. *IEEE Transactions on Biomedical Engineering*, 52(4):652–663, 2005.
37. Rolf Steinbuch. *Finite Elemente - Ein Einstig.* Springer, 1998.
38. M. Strengert, T. Klein, and T. Ertl. A hardware-aware debugger for the OpenGL shading language. In *Proceedings of the ACM Siggraph/Eurographics conference on Graphics Hardware*, pages 81–88. Eurographics Association, 2007.
39. Gerhard Winkler. *Image Analysis, Random Fields and Markov Chain Monte Carlo Methods.* Springer, second edition, 2006.
40. Jianchao Yang, John Wright, Yi Ma, and Thomas Huang. Image super-resolution as sparse representation of raw image patches. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2008.*

41. Yongmin Zhong, Bijan Shirinzadeh, Gursel Alici, and Julian Smith. Soft tissue modelling through autowaves for surgery simulation. *Medical and Biological Engineering and Computing*, 44(9):805–821, 2006.
42. B. Zitová and J. Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977–1000, 2003.