

SITE-SPECIFIC MEASUREMENT OF MICRO-STRUCTURAL BONE ADAPTATION AND PREDICTION OF ADAPTING BONE ARCHITECTURE

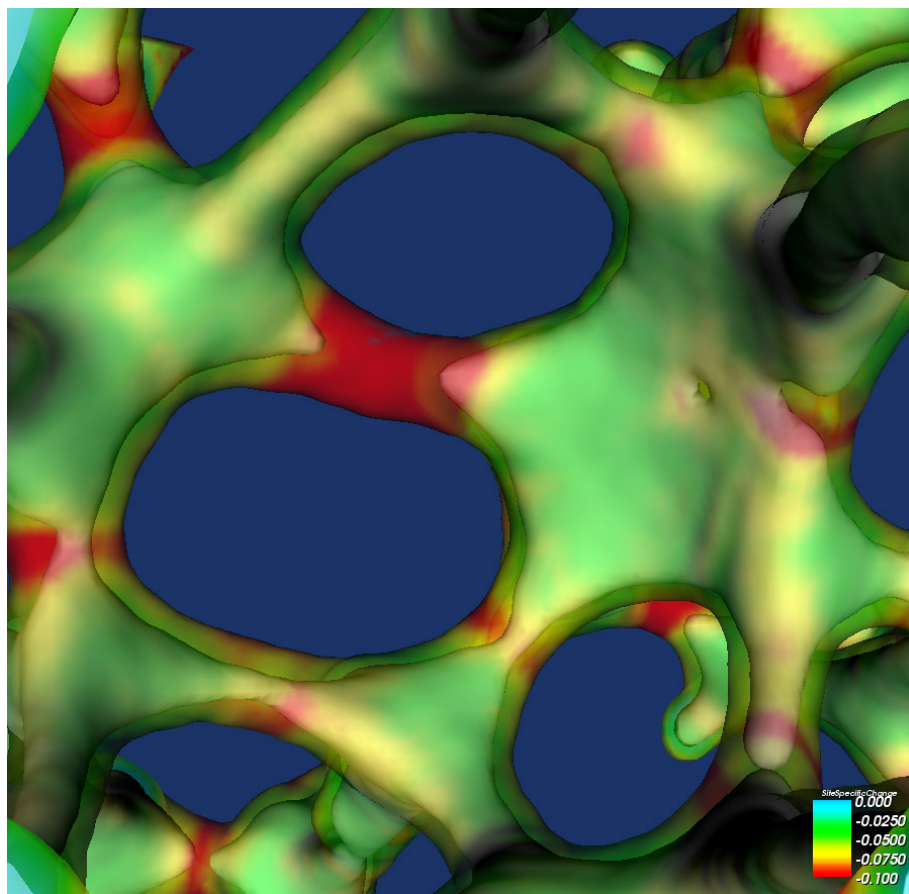
Diplomarbeit: October 2003 to May 2004

Authors:

Corinne Mattmann
Andreas Kuhn

Advisors:

Dr. Steven Boyd
Prof. Dr. Ralph Müller



SITE-SPECIFIC MEASUREMENT OF MICRO-STRUCTURAL BONE
ADAPTATION AND PREDICTION OF ADAPTING BONE ARCHITECTURE

1. INTRODUCTION

Osteoporosis (OP) is a debilitating disease characterized by low bone mass and the deterioration of bone tissue micro-architecture. The disease is most prevalent in women over the age of 50 where, due to a reduction in architectural integrity, osteoporotic bone is prone to spontaneous fracture. Osteoporotic fractures have a devastating effect on morbidity and mortality, and there is a large body of research that focuses on understanding the etiology of the disease. A fundamental goal of this research is to determine methods of fracture prevention, and this can be achieved by a number of means including: (i) early detection of fracture risk, (ii) the development of interventions (i.e., pharmaceutical or exercise), or (iii) the development of preventative measures (i.e., supplemental exercise and diet early in life).

Central to all of these OP studies is the evaluation of bone quality, and this necessitates not only monitoring the apparent density of bone (which is closely related to bone strength), but also the bone micro-architecture. Micro-computed tomography (μCT) is a relatively new technology that plays a central role in quantifying 3D bone micro-architecture, thus allowing a non-destructive, quantitative assessment that has become a crucial component of OP research. Although until recently μCT has been limited to in vitro analysis, new in vivo systems are now emerging. To fully exploit the use of in vivo μCT , appropriate quantitative analysis methods need to be developed that take full advantage of the sequential measures of bone micro-architecture. The purpose of this project is to develop a quantitative method for assessing site-specific bone adaptation, and then to use those time-related data to predict future bone architecture. In vivo μCT provides 3D bone microarchitecture at multiple time points. By using intra-modality image registration, the bone micro-architecture can be quantitatively assessed on a time-based analysis, with site-specific measures. Trends in these site-specific measures can be used to predict future bone architecture. These new measurement tools will be used in future studies using a murine model for OP to investigate the preventative role of exercise and diet in juvenile mice in the long term development of OP.

2. PROJECT TASKS

- o Set up a detailed time-table for the duration of the project.
- o Establish data sets representing serial 3D μCT measurements undergoing resorption and apposition.
- o Determine methodology for measurement of site-specific bone changes (i.e., non-rigid registration, computer graphics methods).

- o Analyze appropriate methods for generalization of site-specific changes (i.e., tensors, localization of change).
- o Develop method to extrapolate future site-specific adaptations; and a method using computer graphics to apply those extrapolated adaptations to bone micro-architecture to generate future bone morphology.
- o Evaluate site-specific bone adaptations (i.e., performance criteria) and prediction of future morphology on (i) artificial data sets (with or without necessary image registration, and (ii) in vivo data sets (murine, rat).
- o Prepare final report and feasibility of performing image registration on live mice and rats.
- o Discussion of results, writing of final report.

3. LITERATURE

1. Cranney A, Ě, Rosen C 2002 Meta-analyses of therapies for postmenopausal osteoporosis. IX: Summary of meta-analyses of therapies for postmenopausal osteoporosis. *Endocr Rev* 23(4):570-8.
2. David V, Laroche N, Boudignon B, Lafage-Proust MH, Alexandre C, R  egsegger P, Vico L. 2003 Noninvasive in vivo monitoring of bone architecture alterations in hindlimb-unloaded female rats using novel three-dimensional micro-computed tomography. *J Bone Miner Res.* 18(9):1622-31.
3. Hajnal JV, Hill DLG, Hawkes DJ 2001 Medical image registration, CRC Press, New York.

Diplomarbeit, WS 2003/2004

Start of Thesis:	27. October 2003
End of Thesis:	7. Mai 2004

Thesis Supervisor ETH:

Prof. Dr. R. M  ller
Institut f  r Biomedizinische Technik

Thesis Advisor:

Prof. S. Boyd, Ph.D.
Department of Mechanical and Manufacturing Engineering
University of Calgary, Canada
skboyd@ucalgary.ca

Z  rich, den 22. Oktober 2003

Abstract

Bone is a living tissue that is highly organized and changes over time. Many of the changes happen inside the trabecular bone. The changes are highly localized. It is essential to understand the evolution of these changes to understand the pathology of diseases. Until today just quantitative measures that give a mean over the whole bone are in use. For example Tb.Th. [7] or the BV/TV ratio. These measurements are very important but they do not give any information about local changes. They cannot answer questions as for example: Where occur the biggest changes in the bone? Where are trabeculae lost? Is there even any local apposition? How does the future bone shape look like? One reason why only these static parameters are available is because until now measurements could only be performed in-vitro. Lately, in-vivo μ CT scanners came on the market and opened a new research field. Now it is possible to trace local changes.

The goal of this project was to derive new methods in order to approach the answers of the questions above. One very important step is to measure the distance between subsequent in-vivo scans. Prior to this measurement, a very important pre-processing is the 3D registration of these datasets because local changes can only be traced if subsequent images are superimposed perfectly. During the development of the methods, SIBA [10] was an often used tool in order to exclude the issues of the registration.

Two very different approaches were chosen in order to find the local changes (site-specific changes). On the one hand, geometric methods were implemented that use either the normals of a surface or a closest point search. On the other hand, a deformable registration was used. This non-rigid registration was then used to predict future bone shapes by following the vectors in the deformable fields of subsequent images. The geometric approach was used to generalize the results. Different generalization approaches were used. Very straight forward mean calculations were implemented in order to get a first impression of the changes. However, as a lot of information gets lost in these mean measurement other ideas as a principal component analysis, a percentage change (compared to local thickness) or the spacial localization of lost trabeculas have been implemented.

The new methods were evaluated with various datasets in order to see if the algorithms work as expected. Finally, they were applied to in-vivo μ CT data which was provided by Novartis, Switzerland.

Contents

1	Introduction	1
2	Overview	3
3	General remarks	4
3.1	Image data, surface data and thresholding	4
3.2	Process for a site-specific in-vivo measurements over time	5
I	Calculation of the site-specific changes	6
4	Introduction	7
5	Geometric methods in order to find the site-specific bone change	8
5.1	Introduction	8
5.2	Calculate site-specific change using normals	9
5.2.1	General idea of this method	9
5.2.2	Starting point and critical situations	9
5.2.3	Algorithm	12
5.2.4	Search depth	14
5.2.5	Memory consideration	15
5.3	Calculate site-specific change using the closest point method	15
5.3.1	General Idea of this method	15
5.3.2	Starting Points	15
5.3.3	Algorithm	16
5.3.4	Search depth	17
5.3.5	Memory consideration	17
5.4	Statistics	17
5.4.1	Which points are included in the statistics?	19
5.4.2	Values provided by "vtkSiteSpecificChangeFilter"	19
5.4.2.1	Counting numbers	19
5.4.2.2	Statistical numbers	20
5.4.2.3	Others	21
5.4.3	Histogram	21
5.5	Evaluation of the site-specific change filter	21
5.5.1	Computation time of the filter	22
5.5.1.1	Computation time as a function of the number of points	23
5.5.1.2	Influence of vtkCellLocator on the computation time	24

5.5.1.3	Computation time using real data	25
5.5.2	Accuracy of the results determined with geometric objects . . .	26
5.5.2.1	Two spheres	26
5.5.2.2	Using a cone and a cylinder	26
5.5.2.3	Using Cylinders to show resorbtion of trabecula	30
5.5.2.4	Show not considered points	32
5.5.3	Test the filter with μ CT data	32
5.5.4	Test with real in vivo μ CT data)	34
5.5.5	Reproducibility test	36
5.5.6	Sensitivity to Registration (translation)	38
5.6	Comparison of the normals and the closest point method	39
5.6.1	Speed	39
5.6.2	Sensitivity to changing bone structures	40
5.6.3	Flexibility	40
5.6.4	Rough surface	40
5.6.5	Sensitivity to Registration	40
5.6.6	Bone apposition	40
5.6.7	Huge changes	41
5.6.8	Summary	41
5.7	Recommendations	41
6	Deformable Registration to find the site-specific bone change	43
6.1	Introduction	43
6.2	Calculate a global deformation pattern using Demons Deformable Registration	44
6.2.1	General idea of this method	44
6.2.2	Advantages and disadvantages of the demons registration method	46
6.2.3	Speeding up the algorithm: Multi-resolution pyramid	47
6.2.4	Standard deviation	48
6.2.5	Number of iterations	49
6.2.6	Memory consideration	49
6.3	Calculate a global deformation pattern using FEM Deformable Registration	50
6.4	Evaluation of "DeformableRegistrationFilter"	50
6.4.1	Evaluation Method	50
6.4.2	Performed Tests	52
6.4.3	Accuracy of the Demons Deformable Registration with Simulated Bone Atrophy	52
6.4.4	Accuracy of the Demons Deformable Registration with the Asymmetrical Formula	57
6.4.5	Accuracy of the Demons Deformable Registration with Real Bone Atrophy	59
6.4.6	Comparison of the Site-Specific Change calculated with the Geometric Approach compared to the one extracted from the Deformable Field	60
6.4.7	Influence of the standard deviation on the registration	62
6.4.8	Computation Time	65
6.4.9	Computation Time for multi-resolution techniques)	67

6.4.10	How does the deformable field look like	69
6.4.11	FEM Registration	72
6.5	Discussion	74
II	Generalization	76
7	Introduction	77
8	Basic measurements	78
9	Relative site-specific change	79
9.1	Concept	79
9.2	Discussion	80
10	Principal components of site-specific change	82
10.1	Concept	82
10.2	Fitting of an ellipsoid	82
10.3	PCA	83
10.4	Discussion	86
10.4.1	Different gaussian smooths	87
10.4.2	Size	87
10.4.3	Translation	87
11	Identify resorbed trabeculae	89
11.1	Procedure	89
11.2	Discussion	90
12	Thickness measurement	92
12.1	Introduction	92
12.2	Concept of the surface based approach	92
12.3	Algorithm	94
12.4	Weighting of the mean	95
12.4.1	Surface weighted mean	95
12.4.2	Unweighting the mean	96
12.5	Example	97
12.6	Boundary points	98
12.7	Closest Point instead of sphere	98
12.8	Evaluation	98
12.8.1	Cylinder with a known diameter	98
12.8.2	Compare to Tb.Th and Tb.Sp	98
12.8.3	Change accuracy to speed up the calculation	101
12.8.4	Computation time	104
III	Extrapolation	106
13	Extrapolation with Deformable Registration (Prediction of the site-specific bone change)	107

13.1	Introduction	107
13.2	Important Considerations	107
13.2.1	Choice of the Deformable Fields Used to Extrapolate	107
13.2.2	Forward versus Backward Field	109
13.2.3	Finding the Belonging Vectors to Extrapolate	110
13.2.4	Inversion of the Field	111
13.3	Different Approaches for the Extrapolation	113
13.4	Evaluation	117
13.4.1	with simulated data	117
13.4.2	with real data	120
13.5	Discussion	124
IV	In vivo measurements	125
14	Volume and thickness	126
15	PCA	128
16	Visual inspection	130
	Outlook	131
	Bibliography	133
	Appendix	135
A	Implemented Applications	137
A.1	AppSiteSpecificChange	137
A.2	AppSiteSpecificThickness	140
A.3	AppDetectResorbedTrabeculae	141
A.4	AppDeformableRegistration	142
A.5	AppAutomatedTestDeformableRegistration	143
A.6	AppExtrapolation	144
B	Implemented Filters	147
B.1	vtkSiteSpecificChangeFilter	147
B.1.1	Input/ Output	147
B.1.2	How to visualize the result of the filter (site-specific change) . .	149
B.1.3	Settings of the filter	150
B.1.4	Useful functions included in vtkUtility	151
B.2	vtkSiteSpecificThicknessFilter	153
B.2.1	Accuracy	154
B.2.2	Points with no thickness	155
B.2.3	Settings of the filter	156
B.2.4	Returned values (report)	156

B.3	vtkReportPCA	157
B.4	vtkConvertAttachedVectorsToPolyData	157
B.4.1	Settings of the filter	157
B.5	vtkScalarOperation	158
B.5.1	Settings of the filter	158
B.6	vtkDetectResorbedTrabecula	158
B.6.1	Settings of the filter	159
B.7	itkDeformableRegistrationFilter	159
B.7.1	Input/ Output	159
B.7.2	Settings of the filter	160
B.8	itkSymmetricForcesDemonsRegistrationFilter and itkSymmetric- ForcesDemonsRegistrationFunction	162
B.9	itkExtrapolateDeformationFieldFilter	162
B.9.1	Input/ Output	162
B.9.2	Settings of the filter	162
B.10	itkIterativeInverseDeformationFieldImageFilter	163
C	Writing an itk-filter	164
C.1	Templates	164
C.2	Necessary Functions	164
C.3	Set up the pipeline	165
C.4	How to enable syntax highlighting for txx-files in Visual C++	165
D	Table of the used test data	166

Chapter 1

Introduction

Osteoporosis is a very widespread disease in today's society. Simplified, it is a loss of bone mass over time. Mostly elderly women are affected by this complaint. They are afraid of getting a fracture because the bone is less sturdy than before. The reason for the loss of bone is not clear at all at this moment.

Today it is known that not only the density of the bone but also the structure of the bone has a big influence on the strength of the bone. Therefore, it is possible that two bones with the same density do not have the same strength. Nevertheless, it is the most common clinical treatment to only measure the density of the bone at the moment because it is still a matter of research to also take the structure of the bone, quantitatively, into account.

An important step forward was possible if not only the density of the bone could be measured but also the structure of the bone. This became possible by μ CT scanners. These scanners measure the density of the bone in one plane. If various slices are measured, it is possible to reconstruct the bone in three dimensions with a resolution that is big enough to enable a view of the trabecular structure. That enabled to look at temporal changes in the bone. Two examples are for example [\[14\]](#) or [\[13\]](#).

The drawback of this method is that it is only possible to measure the bone in-vitro. The next step forward will be to measure the bone in-vivo. With this method of scanning the bone in the living animal or human it is possible to make pictures of the same part of a bone over time. Therefore it will be possible to see the change of the structure of the bone over time.

The first goal of this work is to quantitatively measure the change of the bone over time. The found variation of the bone is called site-specific bone adaptation because it is related to the microarchitecture of the bone. To measure this adaptation in a three dimensional manner is a completely new concept. There is only one publication that goes in the same direction [\[12\]](#). However, that work is limited in the resolution because only thresholded data has been used. The changes will be measured by using computer graphics methods with no thresholding. Thereafter it is important to generalize this site-specific change measurement in such a way that it is possible to describe the change of bone with a few numbers.

With the help of this quantitative and time related measurement of bone changes it should also be possible to predict the trends of the changes into the future. A prediction of the future bone change would be great in helping to anticipate the development of the bone of an osteoporotic person.

Chapter 2

Overview

This report is divided into four parts. The first part discusses two different ways of measuring the site-specific changes. On the one hand we used geometric methods in order to find the changes and on the other hand a deformable registration. The geometric methods are based on surface data (polydata) and get the change by using for example the normals or a closest point search. The non-rigid registration operates on voxel data (imagedata) and uses the demons registration method in order to find the vector field that represents the site-specific changes.

The second part presents generalization methods which simplify the site-specific changes to a representation that is as tight as possible and extract the important statements out of the variety of information. The generalization was based on the geometric site-specific measurements.

The third part talks about the extrapolation of the changes to the future. These predictive methods are based on the deformable registration and different extrapolation schemes.

The fourth part discusses some points about the in vivo data we have received from Novartis, Switzerland.

Prior to the first part some general remarks that are important for the whole project are given.

The appendix contains descriptions of the filter and applications that have been implemented and also the description of the test data that has been used for evaluation purposes during the project.

Chapter 3

General remarks

3.1 Image data, surface data and thresholding

In image processing it is always possible to work on voxel data (image data) or surface data (poly data). The surface data is normally constructed from the voxel data (slices) by using for example the Marching Cubes algorithm (Lorensen W.E. (1987) [9]) which has been used during this project. It is important to be aware that the 3D surface on the computer screen only matches the bone if the threshold of the marching cubes algorithm was set at the exact position and therefore the correct isosurface was found. Nevertheless, it is possible to make comparisons of site-specific changes because it is a relative measurement. If the same specimen was measured at different time points and the *same* threshold was used to build the surface you have always the same offset between the surface reconstruction and the surface of the real bone when it is assumed that the scanner has no shift.

The reconstructed surface consists of triangles which are held by points. Figure 3.1 shows an example of an image data slice stack and a surface reconstruction.

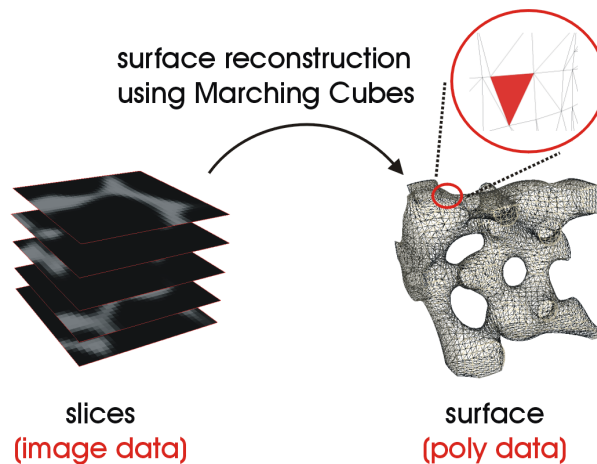


Figure 3.1: surface and poly data are two representations of the information

3.2 Process for a site-specific in-vivo measurements over time

Figure 3.2 shows the whole process of an in vivo assessment over time and also where the actual site-specific change measurement comes into play. The figure also shows that the geometric methods were used for the generalization and the deformable registration in order to predict the adaptation of the bone.

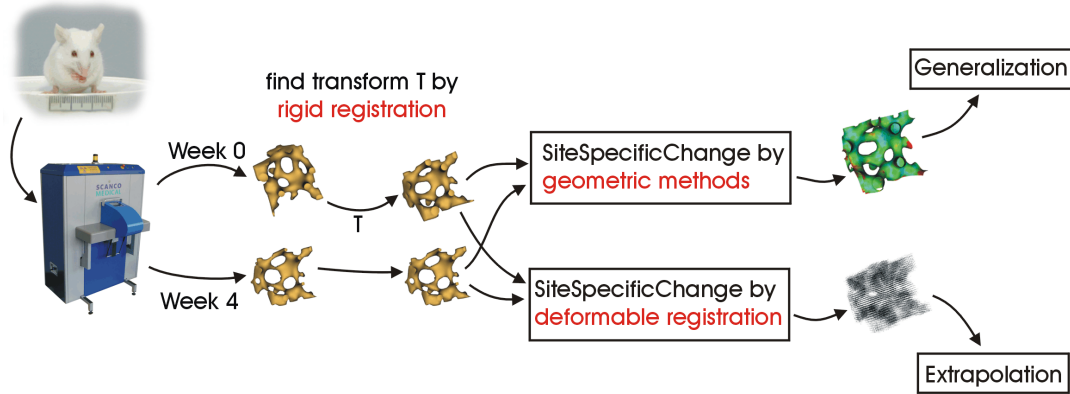


Figure 3.2: this figure shows the process for measuring bone changes (in-vivo) over time

Part I

Calculation of the site-specific changes

Chapter 4

Introduction

Site-specific measurements are a concept that hasn't been done before, in the way it is presented here. Therefore we would like to describe this expression in a little more detail.

For a site-specific change measurement always two scans of the bone microstructure are needed. These scans are normally related to different points in time. In general site-specific means a measurement that is performed between these two bones and returns the change in all points. In this context we mean with surface not only the surface reconstruction because such measurements can also be done on image data.

It is important to be aware that the site-specific change measurement of a bone microstructure needs preprocessing. Above all, this preprocessing includes the rigid registration of the two bones (see also figure 3.2). It is important that this registration is very exact. After the registration the site-specific change measurement can be started. The result of this measurement are vectors on one of the two bone surfaces. These vectors represent the site-specific changes in every point.

For the development of the measurements which are presented in this part it was assumed that the two bones are perfectly registered. Therefore a good tool for the evaluation was the Simulated bone atrophy (SIBA) [10]. This is a technique that simulates the loss of bone by applying a gaussian filter and a threshold to the data. SIBA can be applied several times to the same data set and simulate in this way a bone loss over time. The advantage of using SIBA for the evaluation is that it is always perfectly registered. Therefore a bad registration prior to the site-specific measurement has not to be taken into account.

Chapter 5

Geometric methods in order to find the site-specific bone change

5.1 Introduction

This chapter discusses the geometric methods to calculate the site specific change of the bone. We used two different geometric approaches. The methods are discussed in sections 5.2 (normals method) and 5.3 (closest point method). In these two sections it is also explained why we calculate all site-specific changes beginning from *surface no. 1* where surface no. 1 is defined as earlier time point and *surface no. 2* is the one later in time. We used numbers for the surface names because it is easy to remember that the scan taken earlier in time has also a smaller number.

Each of the methods has certain advantages under certain circumstances. This pros and cons are discussed in section 5.6.

To generalize the results of the measurement we need statistical data which describes in a good way the change occurred to the bone. These values are introduced in section 5.4.

The implementation of the two methods into a VTK-Filter is described in section B.1.

In section 5.5 we evaluate the implemented filter by using real μ CT data but also geometric objects.

At the end of the chapter in section 5.7 we give some recommendations in connection with the geometric measurements.

5.2 Calculate site-specific change using normals

5.2.1 General idea of this method

If you are looking at two surfaces and one surface is shrinking or growing in certain regions it is possible to use the normals to calculate the distance between the two surfaces in each point. The general idea is therefore to build the normal in every point of one surface and to intersect this normal with the other surface (see figure 5.1). Out of these two points we calculate the Euclidean distance which results in the site specific change. We assign this site specific change to the starting point of the normal.

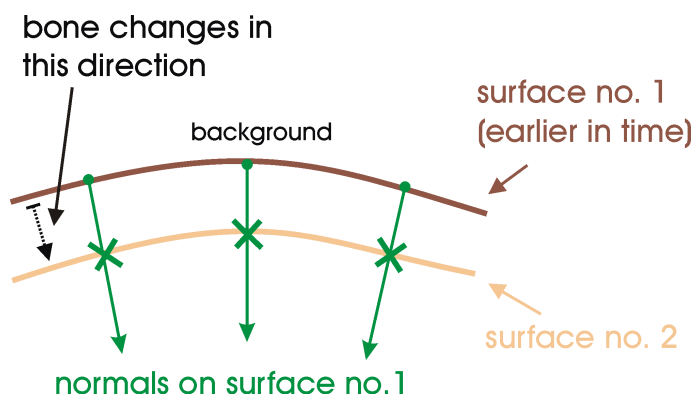


Figure 5.1: sketch of normals method

The algorithm implemented out of this idea is described in section 5.2.3.

5.2.2 Starting point and critical situations

As mentioned in the introduction the picture taken earlier in time is defined as surface no. 1. We define the starting point of the normals on this surface and also the measured site-specific change between surface no. 1 and surface no. 2 is assigned to this point on surface no. 1. We decided to define the starting point of the normals on surface no. 1 because of geometric considerations which will be discussed in the following and the fact that we are looking normally at bone atrophy (bone loss). Yet the method is not limited to bone atrophy. It works also for bone apposition. Only that the critical situations are designed for bone loss.

Figures 5.2 to 5.6 show certain incidents happening in changing trabecular bone. The figures always contain one or two sketches of the situation and a surface reconstruction of μ CT data. All CT pictures contain two surfaces. The outer surface is always brown and transparent and the inner surface is white. The sketch always contains an example of a critical normal on surface no. 1 and one on surface no. 2. This normal shows why we chose to take the normals on surface no. 1.

Figure 5.2 shows the absorption of a trabecula. In such a case we do not find an

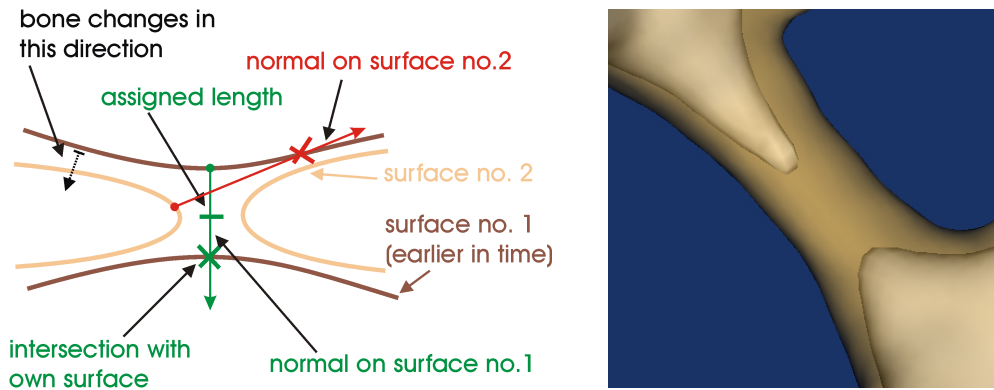


Figure 5.2: resorption of trabeculae

intersection with the inner surface. Therefore we intersect the normal with its own surface and define the site specific change as half of the Euclidean distance. This is a good estimation of the real site specific change in these points.

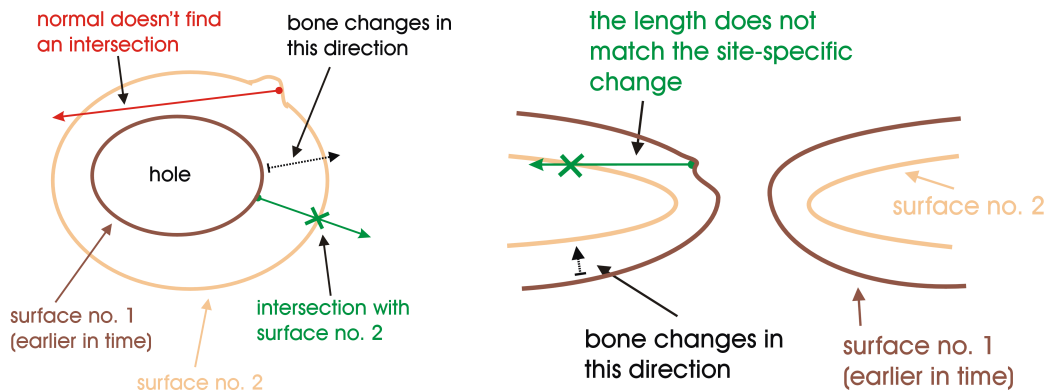
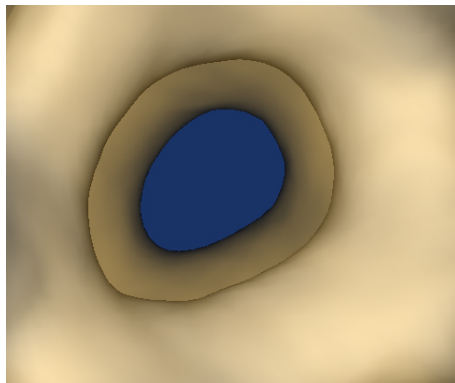


Figure 5.3: hole shaped formation of the bone

In Figure 5.3 you see the situation of a hole shaped formation of the bone. Because of bone loss the hole gets bigger. In such cases it is again better to define the starting point of the normals on surface no. 1 because of small bumps in the bone as it is

illustrated in figure 5.3 on the bottom left. The bottom right sketch shows another cut plane of the same situation. There it is worse to use the mentioned normals but the first mentioned case happens more often because the surface in the second case is bigger and includes therefore more points.

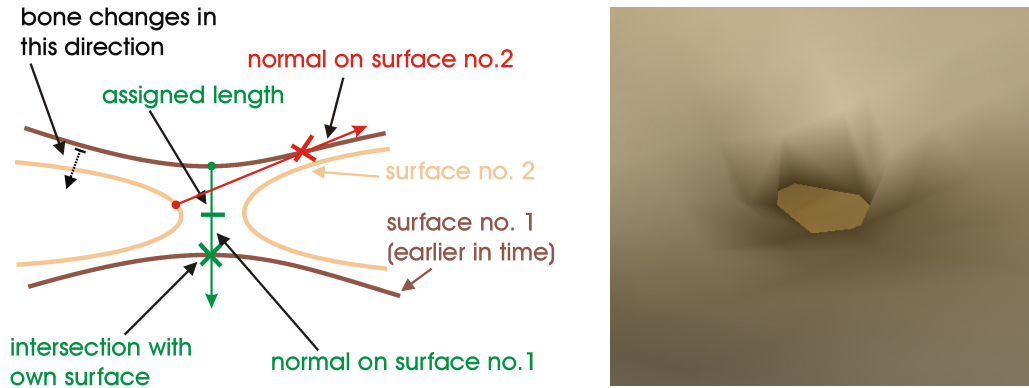


Figure 5.4: new hole arises

If a new hole arises in the bone (see figure 5.4) it will be treated equal to the case of a resorbed trabecula. Therefore it can be considered as the case described before.

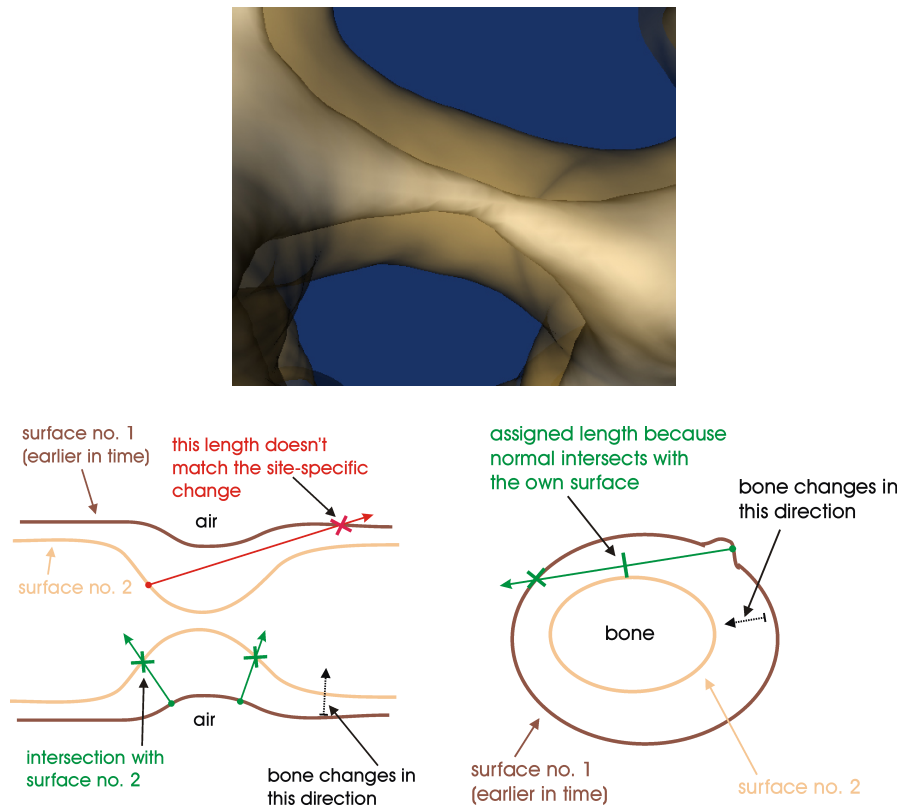


Figure 5.5: bone atrophy (rod like structure)

Figure 5.5 shows an example of a trabecula which gets thinner. The sketch on the bottom left indicates that again it is better to set the root point of the normals on surface no. 1. The bottom right one illustrates that if the normal does not hit the inner surface the half of the length will be assigned because the algorithm thinks that there is a trabecula disappearing. This estimation is near to the real site-specific change.

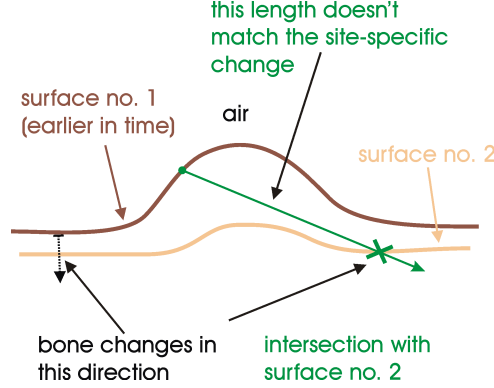


Figure 5.6: shrinking bump

In figure 5.6 you see a picture of a shrinking bump. In this case surface no. 1 is the worse choice as you can see in the figure. There is not a μ CT picture posted because we have never found such a case in our data and therefore the decision was near to use anyway surface no. 1 as root point for the normals because it can be expected that this case is not too frequently.

5.2.3 Algorithm

A flow chart of the normals method is shown in figure 5.7 in this chapter.

As mentioned before we are always looking at two surfaces and try to derive the site specific change in every point. The algorithm goes through every point on surface no. 1 and uses the scheme according to 5.7. Using the surface normals in every of these points we can first search in one direction for an intersection and then in the other direction if no intersection was found (these two cases are colored in yellow in figure 5.7). Because the surface normals are always oriented outwards we know if a loss or apposition of bone arises. We define the site-specific change as Euclidean length from the root point of the normal to the intersection with the other surface. This change is assigned to the point on surface no. 1. Bone loss is defined as negative change and a bone apposition as positive number.

Mostly either bone apposition or bone loss dominates the considered data. That is the reason why we always search first the direction where already the most correct intersections were found. With this method we can reduce the calculation time in the best case by half.

If no correct intersection in either of the two directions was found we try to inter-

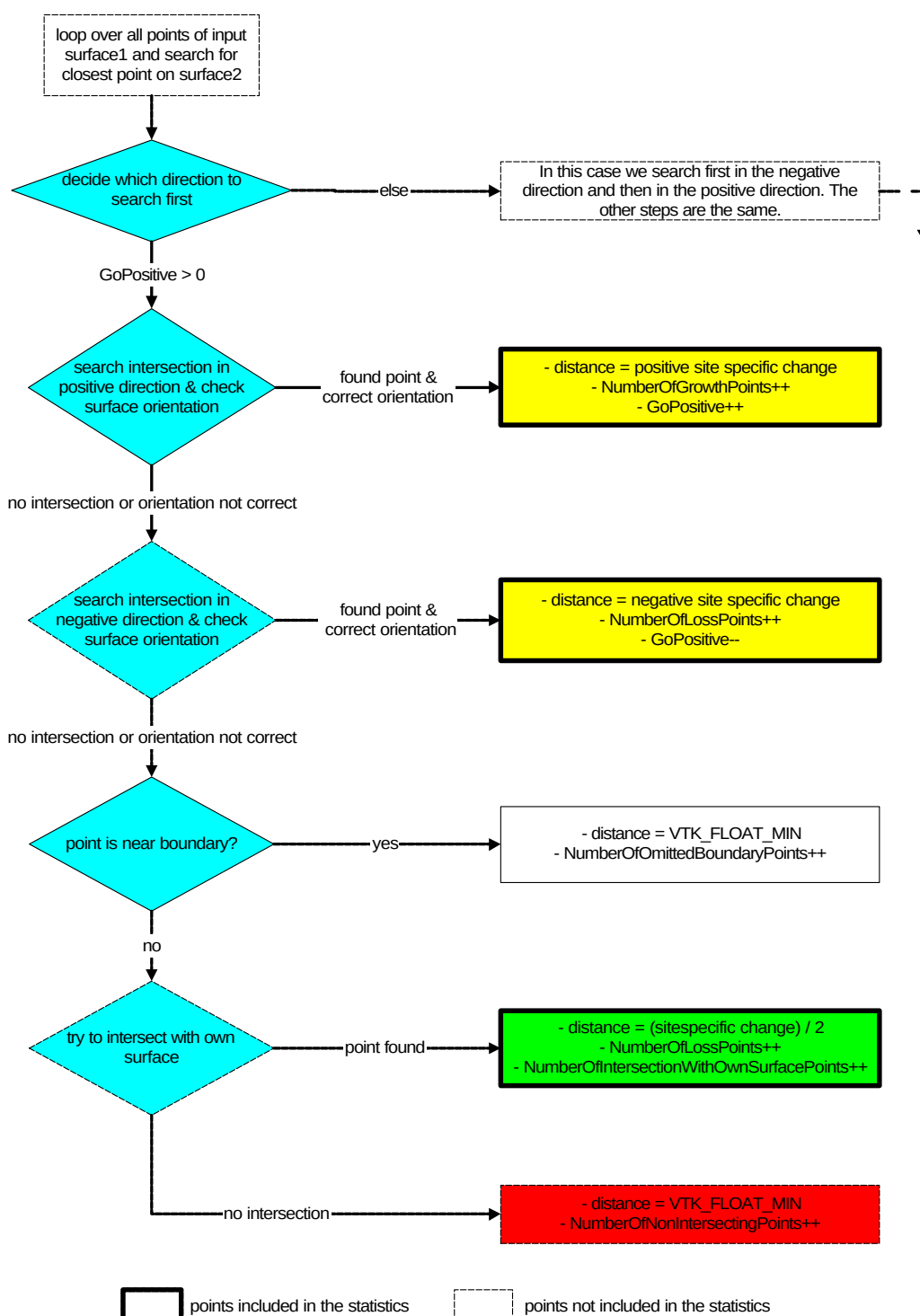


Figure 5.7: flow chart of the normals method

sect the surface with its own (this case is colored in green in figure 5.7). Mostly an

intersection is not found in the case of a resorbed trabecula.

Orientation of the two surfaces to each other

When we are searching for an intersection either on the other surface or on the same surface as mentioned in the subsection before there arises the problem that we have to check the surface orientation (see also 5.7). In the case of intersecting with the other surface (surface no. 2) the normals of both points (starting point of the normal and the intersection point) have to point in the same direction. The same direction is defined as a positive dot product of the two surface normals.

If the orientation is wrong the other direction will be searched.

Boundary points

Boundary points are treated specially because there are often artifacts on the border of the considered surfaces. For example it can happen that just one surface is in the image; the belonging points on the other surface have been cut away. But be also aware that these errors get smaller if bigger data sets are used.

Because the thickness of the bone could be very different the possibility to define the size of this boundary exists (see chapter B.1.3 (Settings of the filter)).

The flow chart in figure 5.7 shows in which case boundary points are omitted. The idea is that we only omit points on the boundary that do not find a corresponding intersection.

5.2.4 Search depth

An important parameter is the search depth of the normals method. This distance defines the length of the normals and therefore the maximum site specific change which can be measured. That is why it is important to define the search depth big enough. You have also to take into account that we sometimes have to intersect with the own surface (resorbed trabecula). Therefore the depth should be adapted to the thickness of the trabeculae.

If the search depth is chosen too large normally you should not run into problems. However, it can happen, that because of artifacts on the boundary or also in the middle of the bone the algorithm finds intersections that are not rejected because the normal in this point was calculated wrongly (less than 0.01%). Therefore it is more safe to estimate the search depth by using the function `GenerateNormals` which is located in `vtkUtility` (see also B.1.4).

5.2.5 Memory consideration

The normals method assigns in every point of the input data the site-specific change and the site-specific change vector. We calculate this change as 4 Byte float type. Accordingly we get per one million surface points 16 MByte of additional data.

If the vector isn't needed it is possible to not output the vectors (see chapter [B.1.3](#) "settings of the filter") because there is redundant information between the scalars, vectors and normals. They are attached anyway to be consistent with the closest point method.

5.3 Calculate site-specific change using the closest point method

5.3.1 General Idea of this method

The general idea of this second geometric approach is to traverse all points on one surface and to find the closest point on the other surface. Between these two points we calculate the Euclidean distance and define this distance as site specific change.

Be aware that the distances (site-specific-change) of the normals- and the closest point method are not the same.

Figure [5.8](#) shows a picture of the closest point method where all the closest point lines are visualized (lines between starting point on surface no. 1 and closest point on surface no. 2). The inner surface (surface no. 2) is white the outer surface (surface no. 1) is transparent and colored according to the site-specific change using the implemented filter *vtkSiteSpecificChangeFilter*.

Figure [5.9](#) shows two sketches of the closest point method.

5.3.2 Starting Points

As mentioned in section [5.1](#) we define the picture taken earlier in time surface no. 1. To be consistent with the normals method (see [5.2.2](#)) we define the starting points of the closest point search also on surface no. 1 and therefore the measured site-specific change is also assigned to the points on surface no. 1. The consistency between the normals method and the closest point method is important to have the possibility to compare the results of the two methods.

The closest point method is less sensitive to errors than the normals method because you do not need special definition for example for a resorbed trabecula (compare to subsection [5.2.2](#)). You can see how the closest point method reacts in the case of a resorbed trabecula in figure [5.8](#). The lines in this figure represent the closest point vectors between surface no. 1 and surface no. 2.

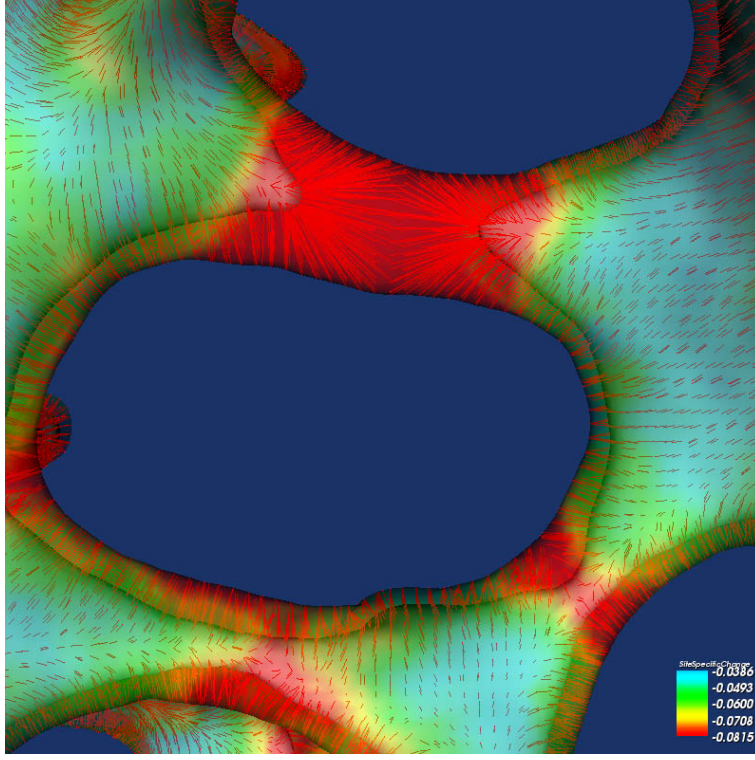


Figure 5.8: visualizing the closest point lines

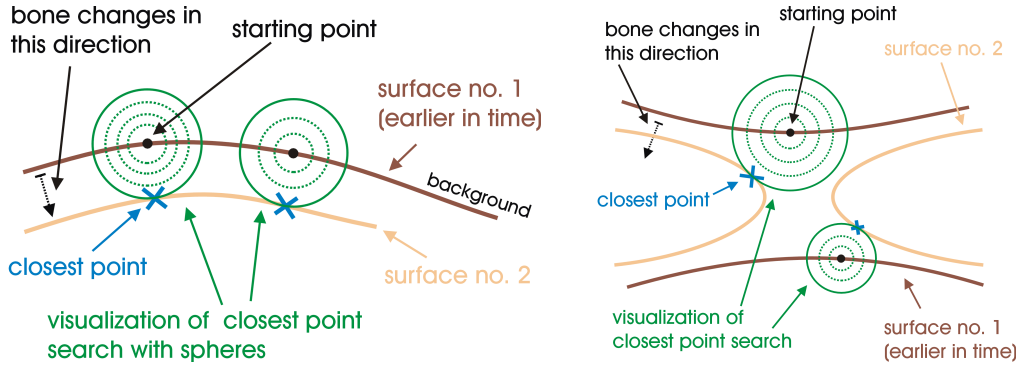


Figure 5.9: two sketches of the closest point search

5.3.3 Algorithm

Figure 5.10 shows the flow chart of the closest point method.

In every point on surface no. 1 we start searching for a closest point on surface no. 2. This search is limited by the search depth which has to be specified. As we did in the normals method the orientation of the two surfaces has to be checked. For this purpose we use the normals in the two points (starting point on surface no. 1 and the closest point found on surface no. 2) and inspect the dot product between the two normals. If the dot product is positive we accept the point found with the closest

point method.

If a point was accepted you have to detect if there happened a bone loss or bone apposition in this point. We decide this by looking at the dot product between the vector from the starting point to the closest point and the normal in the starting point. If this dot product is positive we define the change in bone as apposition and if the dot product is negative as loss.

As we used to treat boundary points specially in the normals method so we do here. The idea is that points on the boundary where no closest point was found or the surface orientation was wrong are not treated in the statistics (see also section statistics 5.4).

5.3.4 Search depth

As we described in the section talking about the normals method (section 5.2) it is also important to select the search depth big enough. There should not be a problem if the depth is chosen too big because we always check the surface orientation. But it is more secure to set it adapted to the magnitude of change in the bone.

The search depth can be estimated by using the function `GenerateNormals` which is located in `vtkUtility` (see also B.1.4). Be aware that the normals generated by this function do not match the direction of the closest point method but it gives a good estimate for the search depth.

5.3.5 Memory consideration

The closest point method calculates two new values. On the one hand there is the vector from the starting point on surface no. 1 to the closest point on surface no. 2, on the other hand we provide the site specific change as a scalar value. Per one million points 16 MByte of memory is needed.

Because often only the scalar value is needed there exists the possibility to not output the vectors (see chapter B.1.3 "settings of the filter") because the length of the vector and the scalar value is redundant information. We decided to provide both outputs because often there is only need for the scalar value (magnitude of the vector) and not the direction of the change.

5.4 Statistics

The statistics included in the *vtkSiteSpecificChangeFilter* are a form of generalizing the result. For big datasets it is not possible anymore to look at the data as suggested in B.1.2 by visual inspection because you cannot see through the bone. Therefore it is important to provide numbers which can describe what is happening in the bone.

In the following it will be described which points of the input data are included in

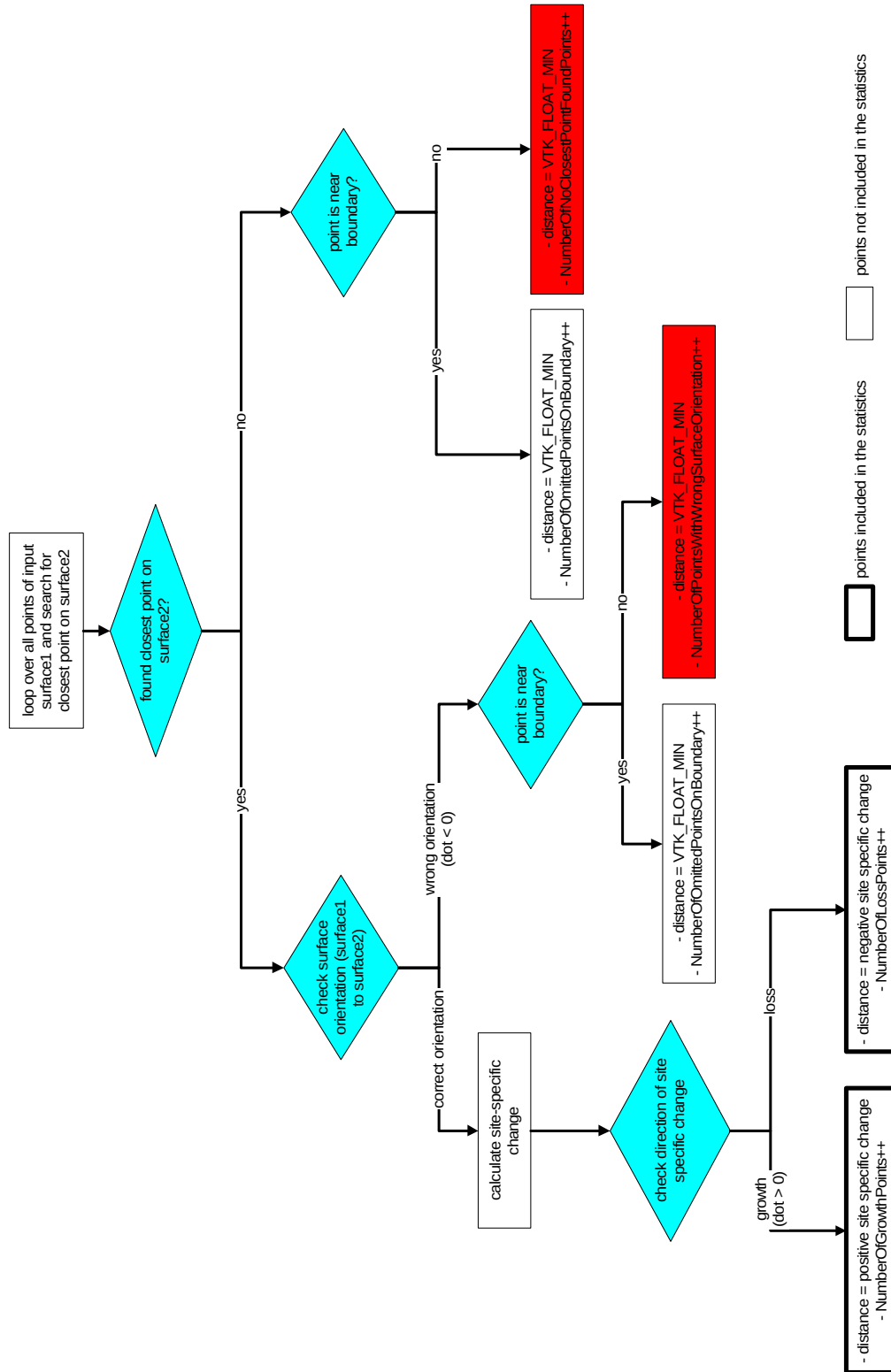


Figure 5.10: flow chart of the closest point method

the statistics and which are left out. Thereafter the different numbers provided by the report (statistics) of the *vtkSiteSpecificChangeFilter* are introduced.

5.4.1 Which points are included in the statistics?

The statistics only includes points with a real bone loss or bone apposition. We do not consider wrong points (e.g. wrong surface orientation, no intersection, no closest point, boundary points) for the calculation of the statistics. The two flow charts 5.7 and 5.10 show which cases are included in the statistics and which are not. The included cases are shown by a thicker box.

The number of included points in the statistics is returned by *NumberOfPointsConsideredInStatistics*.

To distinguish between considered and not considered points we assign the value `VTK_FLOAT_MIN` as distance for the site-specific change. Before calculating the statistics the function *MaskScalars* (see B.1.4) is used to delete temporarily the not considered scalars.

5.4.2 Values provided by "vtkSiteSpecificChangeFilter"

The following titles always correspond to the name which was used in the C++ program (member variables in the class *vtkSiteSpecificChangeFilter*).

Relative numbers are relative to the total number of points of surface no. 1 and provided as value in percentage. Values representing a length are in millimeter.

5.4.2.1 Counting numbers

NumberOfGrowthPoints and NumberOfLossPoints

Returns the absolute and relative number of points on surface no. 1 which have experienced an apposition or loss respectively.

NumberOfNoClosestPointFoundPoints_ClosestPoint

If the closest point method does not find a closest point this variable is incremented (compare to figure 5.10).

If this value is large the search depth is possibly too short (see also 5.3.4).

NumberOfOmittedPointsOnBoundary

When a point lies on the boundary and did not find an intersection or closest point this variable is incremented (compare to figures 5.10 and 5.7).

NumberOfPointsWithWrongSurfaceOrientation_ClosestPoint

If the surface orientation of the found point is wrong (closest point method) this variable is incremented (compare to figure 5.10).

If this number is big (compared to the number of points of surface no. 1) the images could be registered badly.

NumberOfIntersectionWithOwnSurfacePoints_Normals

If a trabecula is resorbed the normals method intersects with its own surface. In such a case this variable is incremented. Compare to figure 5.7 (the green box).

NumberOfNoIntersectionPoints_Normals

If the normals method does not find any intersection at all this number is incremented.

Large numbers could mean that you have to increase the search depth (see also 5.2.4).

5.4.2.2 Statistical numbers

NumberOfPointsConsideredInStatistics

This number shows how many points are considered in the statistics. We only consider apposition and loss points and do not consider wrong points (e.g. no intersection, no closest point, ...).

MeanOfAllSiteSpecificChangeVectors

In every point on surface no. 1 there is a site-specific change vector which points into the direction of the closest point or into the direction of the normal, depending on which method was used. This vector has the length of the site-specific change. If the point is considered in the statistics (see 5.4.1) it is added up. At the end this sum is divided over the *NumberOfPointsConsideredInStatistics* to get the mean site-specific vector.

The resulting vector shows if the whole dataset was shifted into one direction. The reason for a shift could be a bad registration or bone apposition (loss) in the macroscopic sense.

Mean change (MeanOfLossGrowth)

This number returns the mean of the site-specific change (all loss and apposition points). It shows if resorption or apposition dominates the volume of interest.

Standard Deviation of change(StandardDeviationOfLossGrowth)

Returns the standard deviation of the site-specific change. This value could be skewed if the search depth is chosen too high because few big or small values affect the standard deviation.

Median change (MedianOfLossGrowth)

This number returns the median of the site-specific change.

5.4.2.3 Others**ComputationTimeOfFilter**

This variable holds the computation time of the filter in seconds. The time includes also the passing of the data from the input to the output and the calculation of the statistics.

5.4.3 Histogram

VtkSiteSpecificChangeFilter includes the possibility to show the histogram of the calculated site specific change as xy-plot (see [B.1.3 settings of the filter](#)). Figure [5.11](#) contains an example of such a histogram. The x-axis represents the site-specific change in mm and the y-axis the number of points with the same site-specific change. It can be seen that the histogram gives a very nice overview of the site-specific changes.

5.5 Evaluation of the site-specific change filter

This chapter has five main sections.

- **Computational performance ([5.5.1](#))**
This section talks about the computation time of the two implemented methods as a function of the number of points.
- **Accuracy of the results determined with geometric objects ([5.5.2](#))**
This sections wants to show the correct functionality of the filter with different geometric objects. The idea using geometric objects is to prove the successful operation of the filter in these ideal and well controlled cases.
- **Application to real μ CT data ([5.5.3](#))**
In this section we used real μ CT data but with simulated bone atrophy (SIBA [\[10\]](#)) to evaluate the algorithms. The test data is summarized in chapter [D](#).
- **Application to real in vivo μ CT data ([5.5.4](#))**
In this section the filter gets tested with real in vivo data.

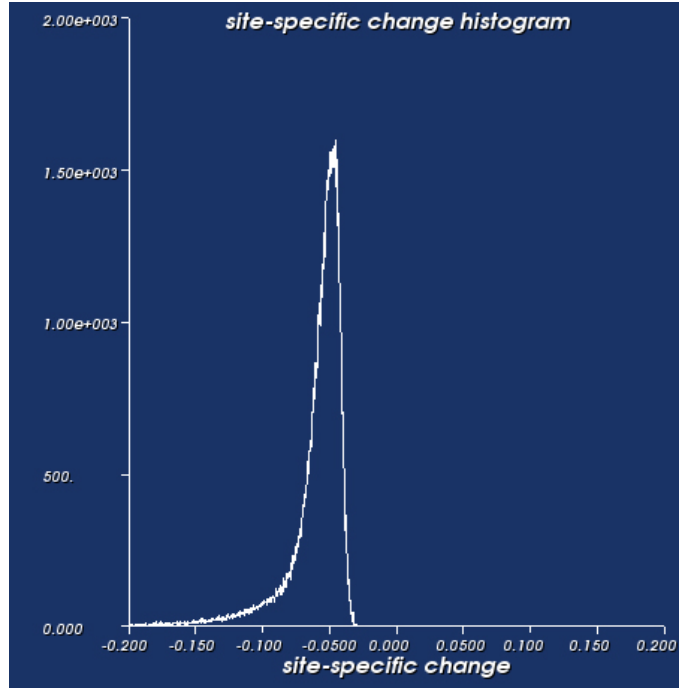


Figure 5.11: example of a Histogram

- **Reproducibility of the scans** (5.5.5)

This test applies the methods to a reproducibility scan of the μ CT scanner. It wants to show the accuracy of the whole process from scan until site-specific measurement.

- **Sensitivity to Registration (translation)** (5.5.6)

This section analyzes the constraints due to translations of the data sets. Translated data sets simulate badly registered images.

5.5.1 Computation time of the filter

It is very important to have an estimation of the computation time of the filter for a given dataset. Therefore we measured the time that elapses during the calculation of the filter. The test was running on a Windows XP PC with 2.17GHz and 1GB of RAM.

The described filter handles surface data. This data is composed of points. That is the reason why the speed for the filter is measured with different number of surface-points and also for the different methods implemented in the filter (normals method and closest point method).

5.5.1.1 Computation time as a function of the number of points

In this test the inputs to the filter were two spheres. Surface no. 1 was a bigger sphere and surface no. 2 a smaller one (see figure 5.15 on the right). The computation time of the filter was measured for different number of points of surface no. 1 (bigger sphere). It is surface no. 1 which changes the speed of the algorithm because the filter traverses the points of this surface with the method set.

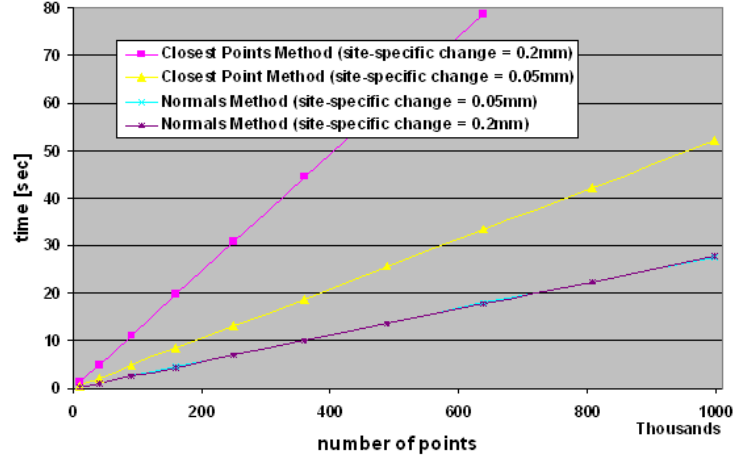


Figure 5.12: computation time of the implemented methods

Figure 5.12 shows the elapsed time compared to the number of points which is linear in time for both methods. It can also be seen that the normals method is faster.

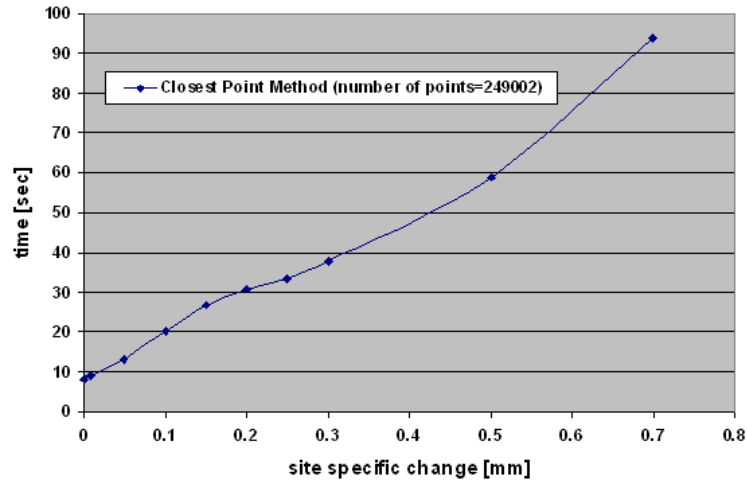


Figure 5.13: computation time closest point method with varying site-specific change

Another insight you get of this chart is that the closest point method is strongly

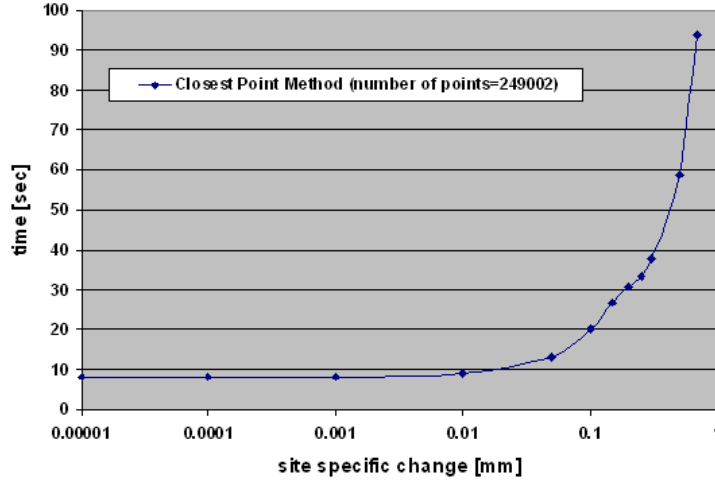


Figure 5.14: computation time closest point method with varying site-specific change (logarithmic)

depending on the amount of the site-specific change. If this change gets bigger the method gets slower. This correlation can be seen in figure 5.13 which shows the time compared to the site-specific change for the closest point method. 5.14 shows the same but with a logarithmic scale. With the logarithmic scale you see that for small site specific changes the speed converges to a certain value. For small site-specific changes the speed of the closest point method comes close to the normals method.

This test was accomplished by using again the two spheres. The computation time was measured for different radii of the inner sphere. Figure 5.15 shows an example of the spheres. The inner sphere on the left side has a radius of 0.95, the one on the right a radius of 0.8. The outer sphere has a radius of 1. Therefore the site-specific changes are 0.05 and 0.2 in this example.

The normals method does not suffer from the changing computation time depending on the amplitude of the site-specific change as can be seen in figure 5.12.

5.5.1.2 Influence of vtkCellLocator on the computation time

The vtkCellLocator is used to search for the closest point (closest point method) and to intersect the normals with the surface (normals method). In [6] you can read that vtkCellLocator is a spatial search object to quickly locate cells in 3D. There is also mentioned that many other types of spatial locators have been developed, such as variable depth octrees and kd-trees. These are often more efficient for the operations described here. VtkCellLocator has been designed for subclassing; so these locators can be derived if necessary. In our project we used only the vtkCellLocator because at this time we did not have problems with the computation time. But we are sure that an improvement is possible by implementing a faster search method.

The measurements in subsection 5.5.1.1 were done by using the vtkCellLocator. As

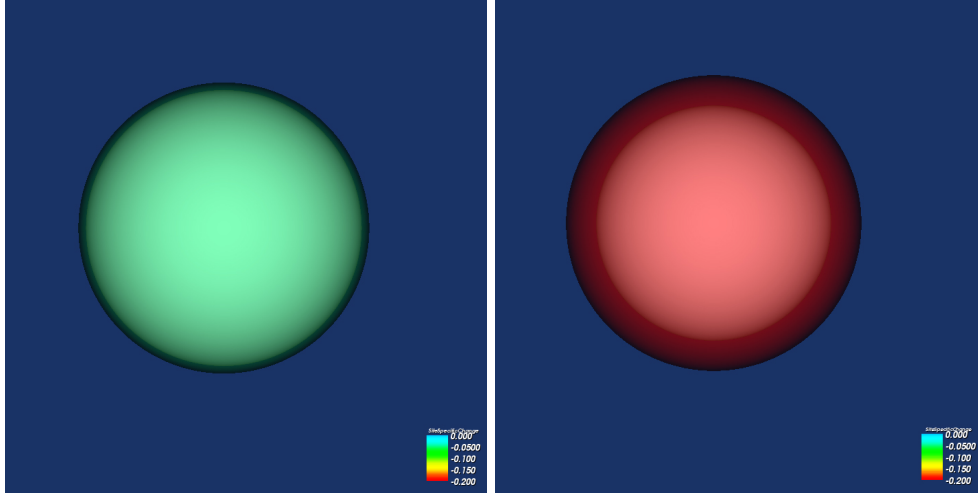


Figure 5.15: computation time for the closest point method with different inner sphere radii

mentioned before it is a tree structure for searching objects in 3D. It is possible to adapt the depth of the tree and also the number of buckets used to build the tree. For the measurements in subsection 5.5.1.1 we used the setting *AutomaticOn()* and did not change these parameters. We also made measurement with different data sets and different tree depths and number of buckets. The results were not deterministic because the speed of the search is strongly depending on the topology of the data. That means that there is a best choice for the parameters but it is depending on the topology of the data. Therefore we used for all tests the *AutomaticOn* setting of the *vtkCellLocator* so that the results are comparable.

5.5.1.3 Computation time using real data

Figure 5.16 shows the same chart as 5.12 (computation time with two perfect spheres) but now with added points for the time measurement of real μ CT data. The red points show the measured points with the normals method. All measurement show a longer computation time. The reason for this is the *vtkCellLocator* which is used to intersect the normals with the surface. Depending on the topology of the data the cells of the surface can be found more or less efficient (see also 5.5.1.2). Figure 5.12 can still be used as an estimate for the computation time using the normals method.

The green rhomb represents the time measurement with the closest point method. Next to each rhomb the mean site-specific change is given because the computation time strongly depends on this change (see 5.5.1.1). Because the topology of the data affects the speed of the closest point function it is even harder to predict the computation time. But also in this case the results of the sphere can be considered as an estimation of the computation time.

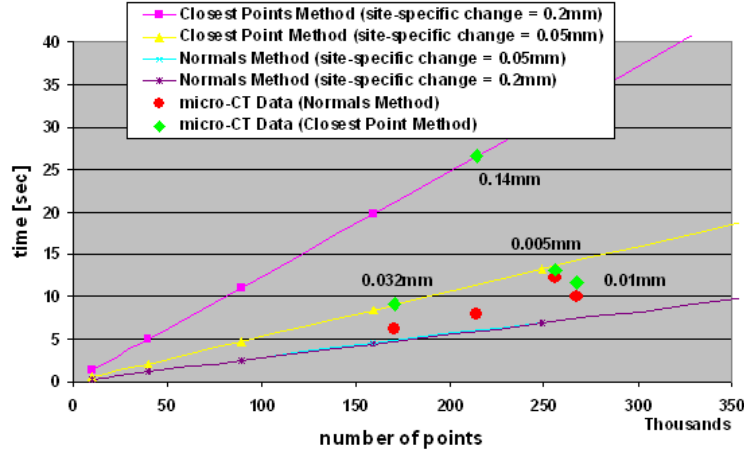


Figure 5.16: Computation time using real data

5.5.2 Accuracy of the results determined with geometric objects

5.5.2.1 Two spheres

This test uses two concentric spheres to show the correct operation of the normals and the closest point method of the filter. In all figures the outer sphere is transparent that the inner one is visible. The bigger sphere has a radius of 1 and the smaller one a radius of 0.8. Therefore the site specific change should be 0.2 in every point. Because we want to simulate a loss, the outer sphere is passed as surface no. 1 (SetInput) and the inner sphere as surface no. 2 (SetSource) to the filter. The results would be the same if a bone apposition were simulated. The outer surface (transparent) is colored according to the site-specific change. We see by looking at the scalar bar and the color of the sphere (figure 5.17) that the change is calculated correctly over the hole sphere. The same result is shown by the statistics in tables 5.1 and 5.2. Important values are the *mean of loss* and the *number of loss points*. The statistics are generated by *vtkSiteSpecificChangeFilter* and returned as report as mentioned in section 5.4.

Figure 5.18 shows again the two spheres but with the inner sphere shifted by 0.1 in z-direction and the radius changed to 0.6 for the closest point and the normals method. You can see that the site specific change shows a progression over the different colors of the scalar bar as expected. Looking at the tables 5.3 and 5.4 it can be seen that the mean vector shows the shift in z-direction. Be aware that the length of the vector does not represent the amount of the shift.

5.5.2.2 Using a cone and a cylinder

Figure 5.19 shows a scene of a cone and a cylinder between which we measure the site specific change. The cylinder is passed as surface no. 1 (SetInput) and the cone as

5.5. EVALUATION OF THE SITE-SPECIFIC CHANGE FILTER

number of points inputSurface1 (reference picture)	9802
number of points inputSurface2	9802
number of apposition points	0 (0%)
number of loss points	9802 (100%)
number of omitted points on boundary	0 (0%)
number of points which were intersected with own surface	0 (0%)
number of points with no intersection	0 (0%)
mean of all site specific change vectors	(-8.22619e-009, -4.96623e-009, -7.97503e-010)
magnitude of this mean vector	9.64208e-009
number of points considered in statistics	9802 (100%)
mean of loss/apposition	-0.2
standard deviation of loss/apposition	4.72686e-008
median of loss/apposition	-0.2

Table 5.1: statistics using the normals method (inner sphere radius = 0.8; outer sphere radius = 1; spheres are shown in figure 5.17)

number of points inputSurface1 (reference picture)	9802
number of points inputSurface2	9802
number of apposition points	0 (0%)
number of loss points	9802 (100%)
number of omitted points on boundary	0 (0%)
number of points where no closest point was found	0 (0%)
number of points with wrong surface orientation	0 (0%)
mean of all site specific change vectors	(-6.78416e-009, -2.62239e-009, -3.04043e-011)
magnitude of this mean vector	7.27342e-009
number of points considered in statistics	9802 (100%)
mean of loss/apposition	-0.2
standard deviation of loss/apposition	4.60793e-008
median of loss/apposition	-0.2

Table 5.2: statistics using the closest point method (inner sphere radius = 0.8; outer sphere radius = 1; spheres are shown in figure 5.1)

mean of all site specific change vectors	(-1.93617e-008, -1.76706e-008, 0.0494996)
magnitude of this mean vector	0.0494996
number of points considered in statistics	9802 (100%)
mean of loss/apposition	-0.40428
standard deviation of loss/apposition	0.0704217
median of loss/apposition	-0.410038

Table 5.3: statistics of two shifted spheres using the normals method (inner sphere radius = 0.6; outer sphere radius = 1; shift in z-direction = 0.1; spheres are shown in figure 5.18)

mean of all site specific change vectors	(-1.45502e-008, 2.73382e-009, 0.0509415)
magnitude of this mean vector	7.27342e-009
number of points considered in statistics	9802 (100%)
mean of loss/apposition	-0.40256
standard deviation of loss/apposition	0.0702917
median of loss/apposition	-0.406579

Table 5.4: statistics of two shifted spheres using the closest point method (inner sphere radius = 0.6; outer sphere radius = 1; shift in z-direction = 0.1; spheres are shown in figure 5.18)

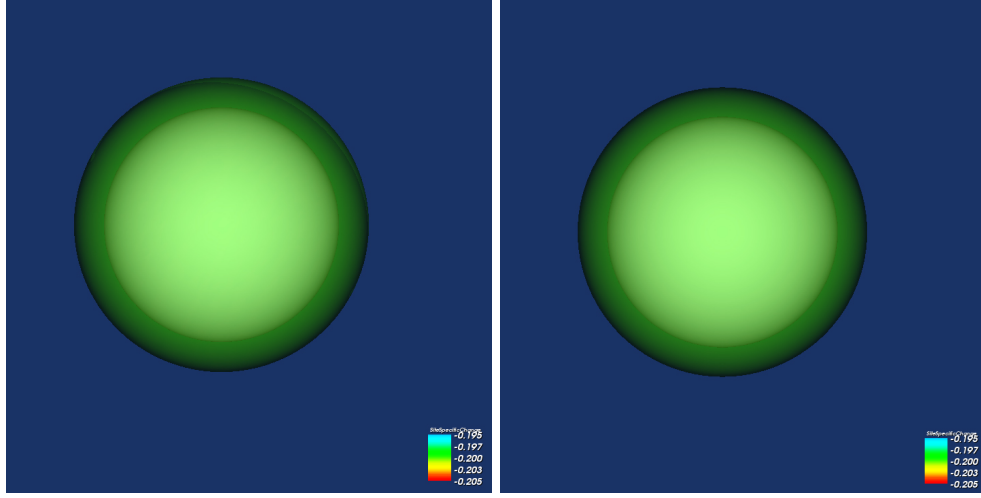


Figure 5.17: evaluation of the normals (left) and the closest point method using two spheres (inner sphere radius = 0.8; outer sphere radius = 1;)

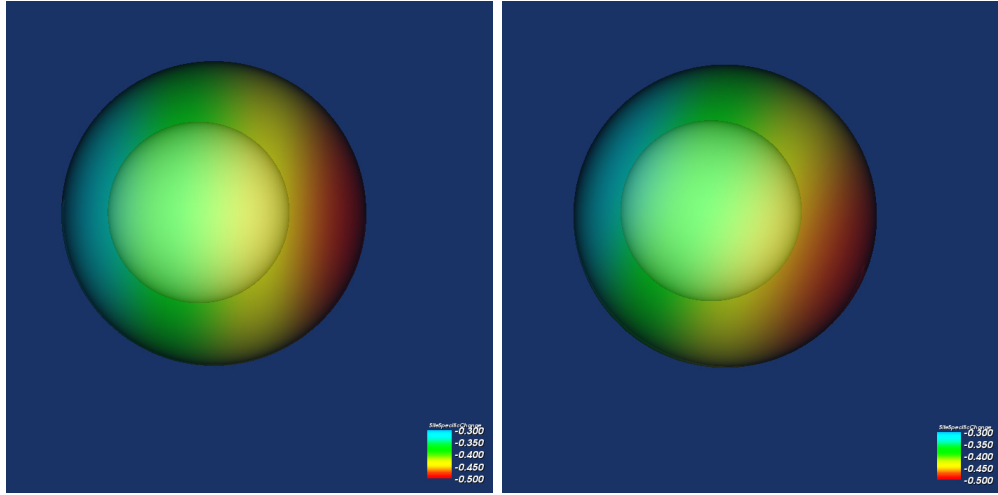


Figure 5.18: evaluation of the normals (left) and the closest point method using two translated spheres (inner sphere radius = 0.6; outer sphere radius = 1; shift in z-direction = 0.1)

surface no. 2 to the filter. That means that we measure the site-specific change from the cylinder to the bone. This test shows that we can measure positive and negative changes in a dataset. As expected the change is positive at the position where the cone is bigger than the cylinder (bone apposition) and the change is negative when the cylinder is smaller (bone loss).

Figure 5.19 shows the histogram of the site-specific change. The histogram shows the expected distribution. When looking at the vertical line at point -1 you can see that it does not intersect with -1. The reason for this is that the histogram is calculated just at a finite number of values (x-axes) (number of points divided by 100).

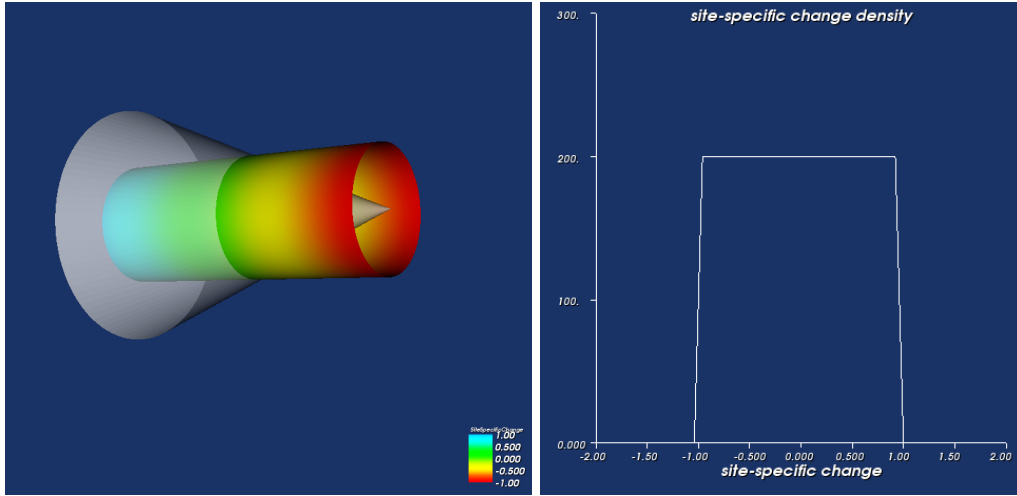


Figure 5.19: positive and negative changes using a cone and a cylinder

5.5.2.3 Using Cylinders to show resorbtion of trabecula

The purpose of this test is to show that the filter is working with bone that loses trabeculae. It also shows how the filter handles such situations using the different methods.

Figure 5.20 shows the scene used for this check. It consists of a big transparent cylinder (radius = 1) which illustrates the trabecula and two smaller cylinders (radius = 0.8) which show the resorbed trabecula. Therefore the big cylinder is surface no. 1 (earlier in time) and the two smaller cylinders surface no. 2 if we are looking at bone loss.

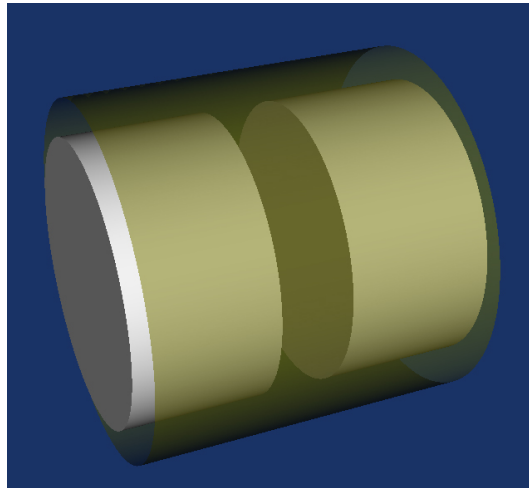


Figure 5.20: scene for testing the case of resorbed trabecula

Figure 5.21 shows the output of the filter using the normals method. The big peak at 0.2 in the density chart shows the difference between the cylinders where they are

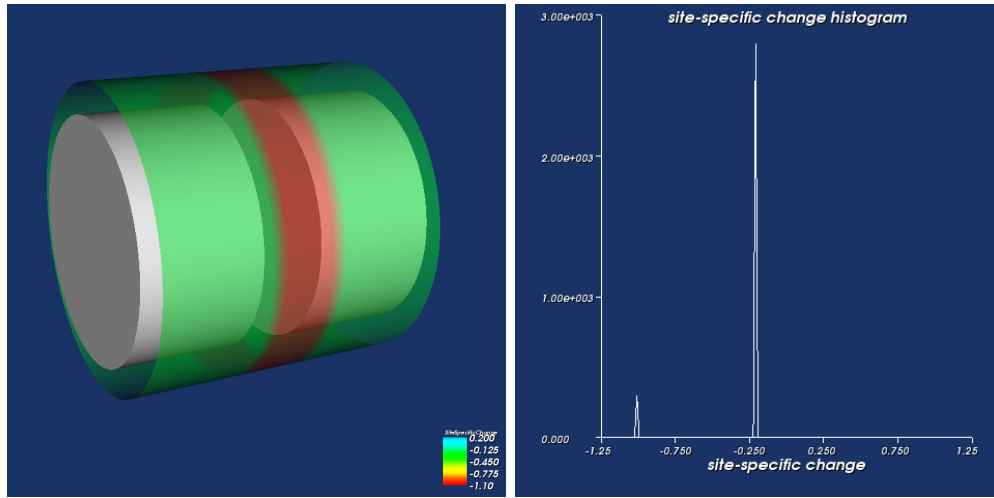


Figure 5.21: resorbed trabecula using the normals method and the histogram of the site-specific change (compare to 5.23 where the closest point method is shown)

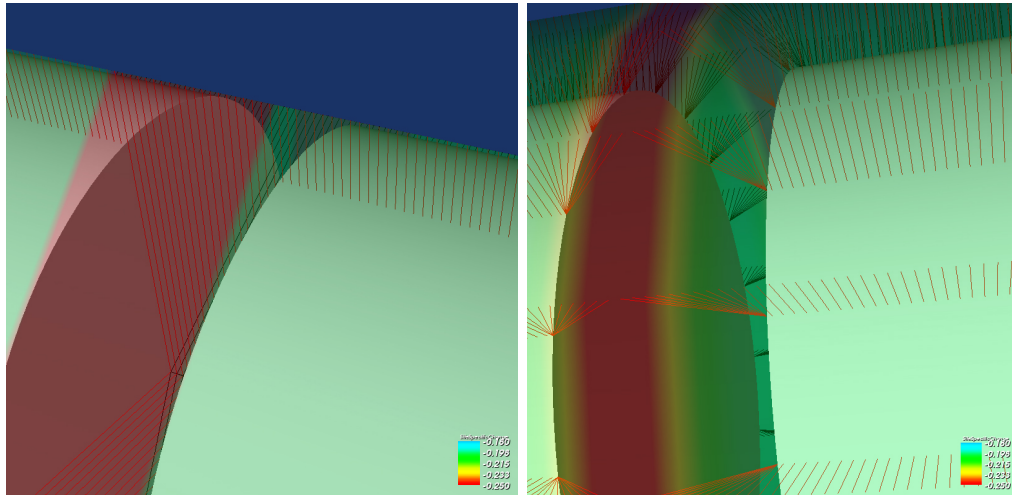


Figure 5.22: visualization of the normals method (left) and the closest point (the red lines visualize the normals with starting point on the outer surface)

overlapping and the small peak at 1 is due to the resorbed trabecula. The normals method intersects with surface no. 1 if there is not an intersection with surface no. 2 as described in 5.2.3 and takes the half of the distance as site-specific change. Because the diameter of the big cylinder is 2 the site specific change at the position of the resorbed trabecula is 1 (the red band on the big cylinder in figure 5.21).

Figure 5.23 shows the output of the filter using the closest point method. You recognize that there is only one peak in the density chart. The reason for this is that the closest point method uses another definition of the site-specific change at a resorbed trabecula. Figure 5.22 visualizes the two different approaches by showing lines in the direction of the site-specific change.

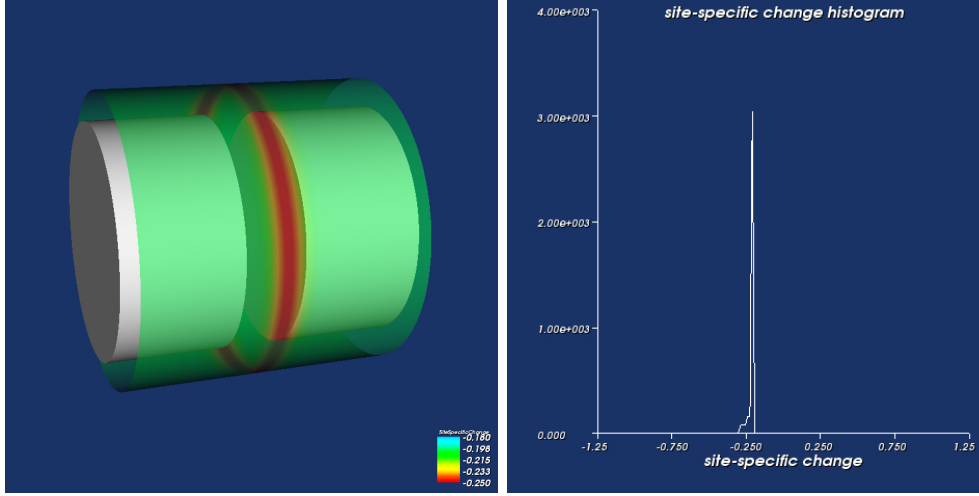


Figure 5.23: resorbed trabecula using the closest point method and the histogram of the site-specific change (compare to 5.21 where the normals method is shown)

Be aware that you cannot compare the result of the filter if not the same method was used.

5.5.2.4 Show not considered points

If the filter does not find a closest point or an intersection for a certain point this point is excluded from the statistics.

As described in B.1.3 there exists the possibility to visualize these points. Figure 5.24 shows a scene of the resorbed trabecula introduced in section 5.5.2.3 by using a search depth that is too short. That is why all points in the middle cannot find an intersection using the normals method.

5.5.3 Test the filter with μ CT data

In this subsection we use the μ CT data mentioned in chapter D to evaluate the filter. This evaluation has the main focus to show that the algorithms of the two methods (see subsections 5.2.3 and 5.2.3) work for various μ CT datasets and different amount of site specific changes. It cannot be checked if the site-specific change is calculated accurate because we do not know the amount of bone change only the correct activity of the algorithm is demonstrated. The accuracy of the method was tested by using the geometric objects in subsection 5.5.2.

Therefore the important parameter in this check is the *number of points considered in statistics* (compare to 5.4.2). If most of the points are used for the statistics you know that the algorithm worked fine. Otherwise it has to be checked what points cause the problems. If these points are *omitted points on the boundary* it is because of artifacts on the boundary, what is not caused by the algorithm. In the other cases

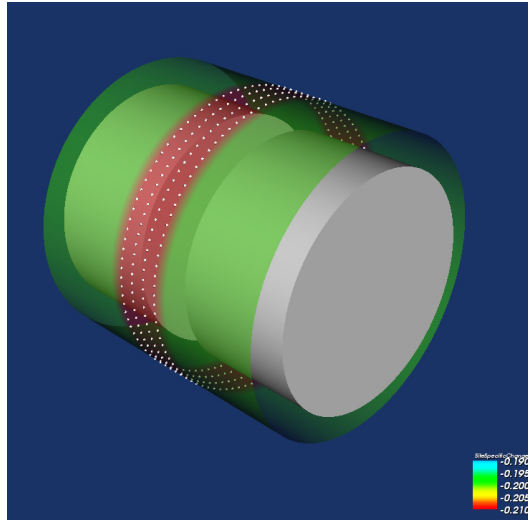


Figure 5.24: example with a too small search depth (the small white spheres show the points where no intersection was found)

(name of statistics entry)	value
number of points inputSurface1 (reference)	140075
number of omitted points on boundary	6105 (4.35838%)
number of points with no intersection	5 (0.00356952%)
number of apposition points	3 (0.00214171%)
number of loss points	133962 (95.6359%)
number of points considered in statistics	133965 (95.6381%)

Table 5.5: evaluation of normals method using aluminum foam (siba1_gfilt_sub.aim; siba0_gfilt_sub.aim)

(no intersection, no closest point, wrong surface orientation) the points have to be checked. It was used the setting *GenerateNotConsideredPoints* (B.1.3) of the filter to find these points in the scene. In the following we show two examples of the tests that have been done.

Aluminum foam

In this check aluminum foam scans were used and SIBA [10] to simulate a bone loss over time.

Figure 5.25 shows the visualization of the result and the histogram of the site-specific change. As usually there are shown two bones. The outer one is transparent and colored according to the site-specific change between the two bones. The picture shows on the right side the loss of a trabecula. Tables 5.5 and 5.6 contain the mentioned numbers of the statistics. You see that very few points didn't find an intersection or were assigned wrong (much less than 1%).

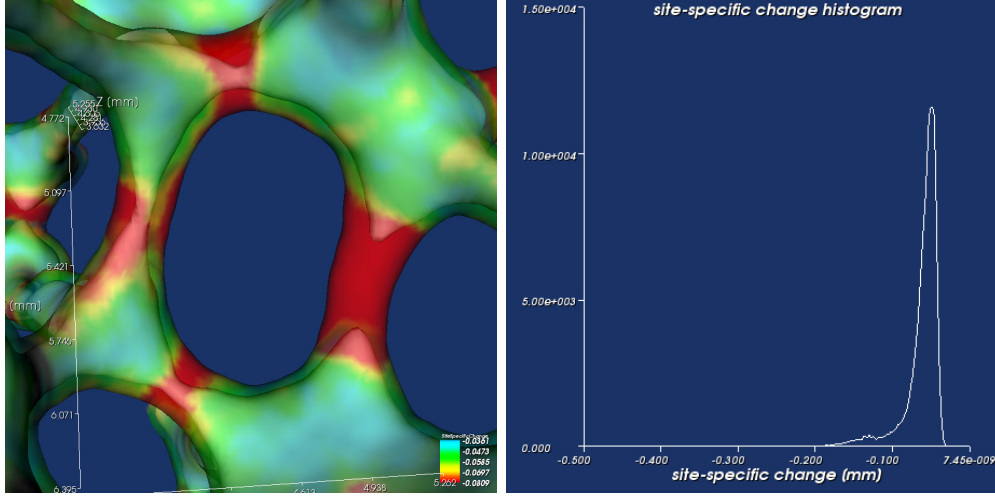


Figure 5.25: site-specific change in an aluminum foam sample (siba1_gfilt_sub.aim; siba0_gfilt_sub.aim)

(name of statistics entry)	value
number of points inputSurface1 (reference)	140075
number of omitted points on boundary	204 (0.145636%)
number of points where no closest point was found	0 (0%)
number of points with wrong surface orientation	5 (0.00356952%)
number of apposition points	32 (0.0228449%)
number of loss points	139834 (99.8279%)
number of points considered in statistics	139866 (99.8508%)

Table 5.6: evaluation of closest point method using aluminum foam (siba1_gfilt_sub.aim; siba0_gfilt_sub.aim)

Canine bone

In this check canine bone scans were used and SIBA [10] to simulate a bone loss over time.

Again figure 5.26 shows the visualization and tables 5.7 and 5.8 the important parameters of the statistics (the calculation was made without closing the boundaries; it was only used to visualize the picture).

5.5.4 Test with real in vivo μ CT data)

In this subsection we give two examples with OVX data and one with SHAM data to show that the methods also work with real data.

- Figure 5.27 shows a small sample of SHAM data. Because the rats have been sham operated the changes are small. This example shows also that the registration of the data is very good.

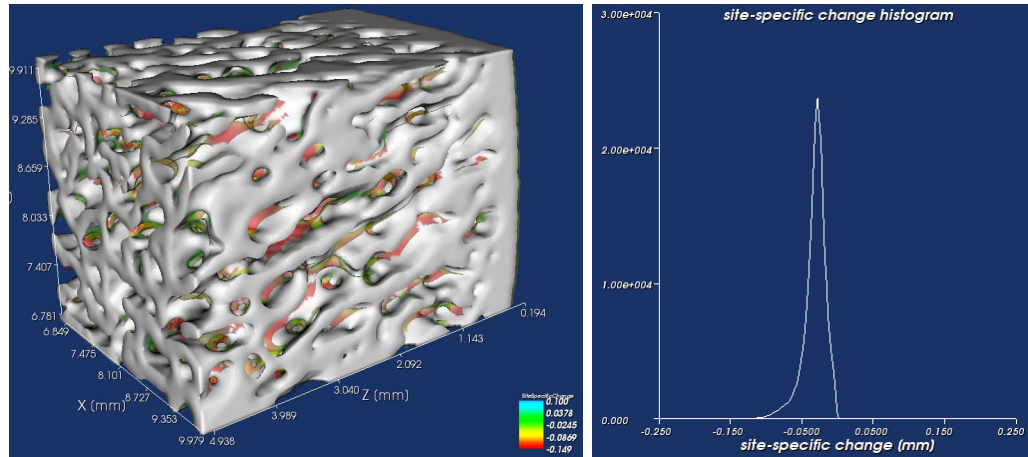


Figure 5.26: site-specific change in a canine sample (dog02_gauss.aim; dog04_gauss.aim)

(name of statistics entry)	value
number of points inputSurface1 (reference)	267469
number of omitted points on boundary	5811 (2.17259%)
number of points with no intersection	138 (0.0515948%)
number of apposition points	56 (0.020937%)
number of loss points	261464 (97.7549%)
number of points considered in statistics	261520 (97.7758%)

Table 5.7: evaluation of normals method using canine bone (dog02_gauss.aim; dog04_gauss.aim)

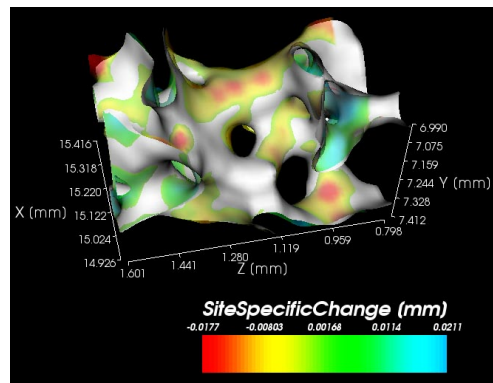


Figure 5.27: example of SHAM rat where the changes are small (C0000475_sub_gfilt_sub.aim and C0000706_sub_gfilt_regist12_sub_regist12.aim)

- Figure 5.28 shows very nicely the case of a resorbed trabecula.
- Figure 5.29 shows an example where big parts of the trabecular structure get resorbed. In such a case the question is, how to define the site-specific changes,

(name of statistics entry)	value
number of points inputSurface1 (reference)	267469
number of omitted points on boundary	214 (0.0800093%)
number of points where no closest point was found	0 (0%)
number of points with wrong surface orientation	181 (0.0676714%)
number of apposition points	147 (0.0549596%)
number of loss points	266927 (99.7974%)
number of points considered in statistics	267074 (99.8523%)

Table 5.8: evaluation of closest point method using canine bone (dog02_gauss.aim; dog04_gauss.aim)

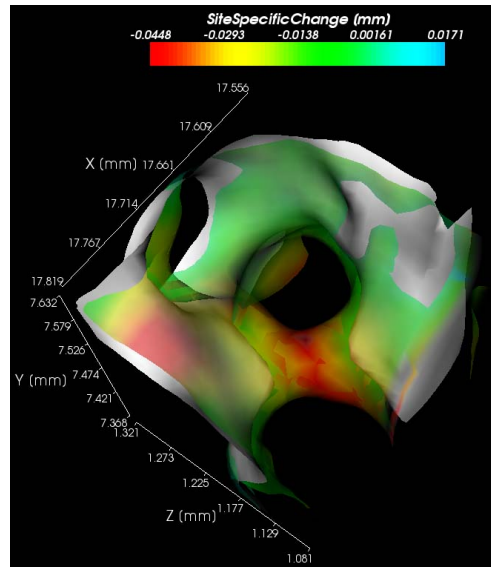


Figure 5.28: this picture shows a resorbed trabecula of an OVX rat (C0000711_sub_gfilt_regist12_sub_regist12.aim and C0000978_sub_gfilt_regist123_sub_regist123.aim)

when they are actually no more site-specific. Using the closest point method will result in huge changes. If the normals method is used the half of the thickness of the resorbed trabecula will be assigned. We think that in this case the definition of the normals method makes more sense.

5.5.5 Reproducibility test

When the same sample is scanned several times with the μ CT scanner without touching we wanted to see if the implementation is enough accurate to measure any changes. The results demonstrate that the measurements show movement of the sample without operator interference. It has to be checked where this inaccuracy of the scans come from (i.e. scanner malfunction or conversion from bmps to aim files).

Figure 5.30 shows one of the measurements that show the translation of the bone. The

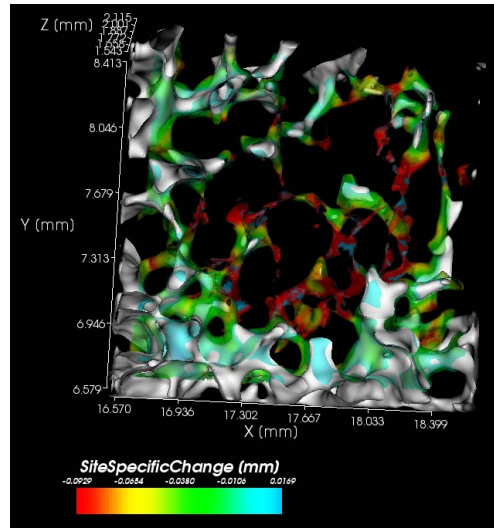


Figure 5.29: this picture shows huge structural changes and therefore it is hard to define site-specific changes (C0000976_sub_gfilt_regist123_sub_regist123.aim and C0001086_sub_gfilt_regist1234_sub_regist1234.aim)

translation can be seen best in the histogram. One observes positive and a negative maxima. That is because after a translation on one side of the bone a positive change and on the other side a negative change is measured. It is even possible to say that the translation was approximately $20\text{ }\mu\text{m}$ (distance from 0 to the maxima). The direction of the change could be found out by using the mean vector of the site-specific changes (see 5.4.2).

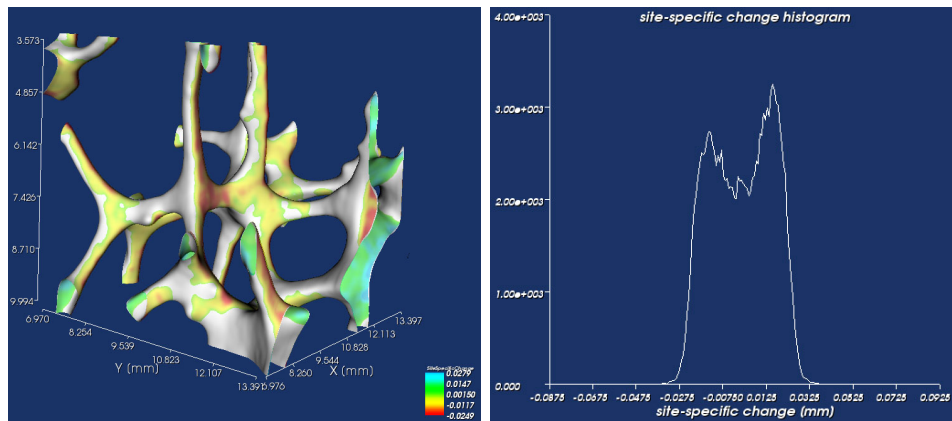


Figure 5.30: no change of the sample measurements (REGIST_ALU_FIX_01_sub.aim REGIST_ALU_FIX_02_sub.aim)

5.5.6 Sensitivity to Registration (translation)

All tests until this point assumed a perfect rigid registration of the two data sets. This is a very nice assumption to check if the methods and algorithms are working fine but in reality there are displacements of the bone. These translations or rotations are normally due to a bad registration prior to the site-specific change measurement. Out of this fact there arise a couple of questions.

1. Can we see in the results if the data is badly registered?
2. Can we still rely in the results we get from the statistics?

To answer these questions different data sets were used. A translation was applied to see the effect of a bad registration. Both methods (normals and closest point method) were used.

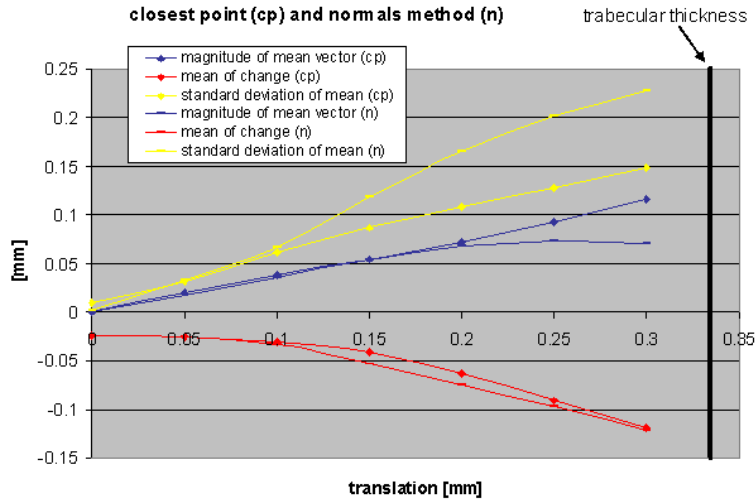


Figure 5.31: statistical parameters while applying a translation to one dataset (siba2_gfilt_sub.aim; siba1_gfilt_sub.aim)

Figure 5.31 shows a representative example of the measurements that were done. Figure 5.32 illustrates the displacement of one surface by 0.3mm in positive x-direction. Out of these tests a couple of conclusions can be drawn:

- It is not deterministic if the mean of the site-specific change changes in positive or negative direction when a translation is applied.
- The trabecular thickness has a big influence of how big the displacement may be. The displacement should not be bigger than 1/4 of the trabecular thickness as a rule of thumb. If it is smaller than this value the mean is still representative for the real change in the bone. The trabecular thickness can be calculated by using the *vtkSiteSpecificThicknessFilter* described in 12.

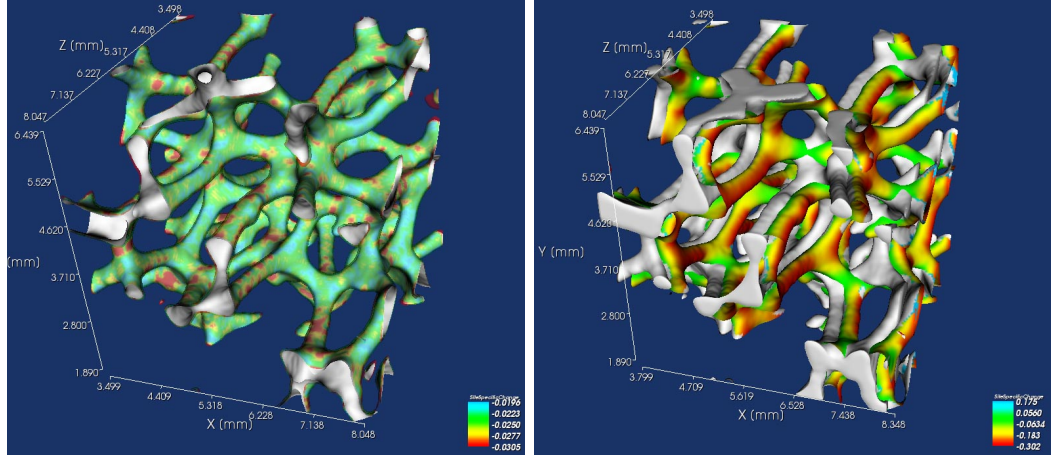


Figure 5.32: visualization without and with shift of 0.3mm (siba2_gfilt_sub.aim; siba1_gfilt_sub.aim)

- The closest point method is less affected by translations than the normals method.
- As mentioned in 5.4.2 the mean vector indicates a displacement of the two surfaces. The magnitude of this vector gives an impression of how far the dataset is shifted. We saw that the magnitude lies always in between 1/2 and 1/3 of the real displacement when using the closest point method.
- The number of apposition and loss points (see 5.4.2) change when translating one surface . If it is known that there was particularly loss or apposition in the VOI these numbers can be used to check if the anticipated change happened.

5.6 Comparison of the normals and the closest point method

5.6.1 Speed

The normals method is usually faster than the closest point method because it has only to search in one direction. The difference in the computation times between the methods depends on the amount of the site-specific change. If this change gets bigger the closest point method needs more time to find the points. The normals method does not suffer from the amount of the site-specific change. On the other side the methods have nearly the same computation time if you are measuring changes which are below 0.01mm (compare to figure 5.14). The topic of the computation time of the two methods is discussed in more detail in subsection 5.5.1.

5.6.2 Sensitivity to changing bone structures

The closest point method is less sensitive to changing structures than the normals method. For example you have to treat points where a trabecula is resorbed differently using the normals method (we intersect with the own surface as described in subsection 5.2.2) what makes the method a little bit more error-prone.

5.6.3 Flexibility

The normals method is more flexible than the closest point method. If the closest point method finds a point that has, for example, a wrong surface orientation you cannot search in an other direction for another point, because it will always find the same closest point. If this closest point is the wrong one you cannot search further, you have to omit this point (see flow chart 5.10). The normals method on the other hand can first find a surface with wrong orientation in positive direction and can then search in negative direction to find possibly the right point.

With the test data we were using this problem did not happen and therefore these comments are theoretically important but perhaps of little practical consequence.

5.6.4 Rough surface

If the surfaces are very rough this could cause a problem for the normals method because the normals could point in a wrong direction. The site-specific change will be overestimated mostly in such a case. The closest point method in contrast finds the nearest point even if the surface is rough.

5.6.5 Sensitivity to Registration

If the registration is insufficient a translation or rotation of the to data sets is possible. In these cases it is better to use the closest point method than the normals method as can be seen in subsection 5.5.6.

5.6.6 Bone apposition

It was mentioned a couple of times that the algorithms work either if there is bone apposition or bone loss. The only problematic point happens in the case of joining trabecula or similar events. In such a case the normals method has problems to define the site-specific change whereas the closest point method do not. Therefore it is better to use the closest point method for bone that experiences notable apposition.

	Normals method	Closest point method
Speed	+	
Sensitivity to disturbances		++
Flexibility	+	
Rough surface		+
Bad registration		+
Bone apposition		+
Huge changes	+	

Table 5.9: Summary of the comparison of the two methods

5.6.7 Huge changes

If there are huge changes occurring (whole trabecular structures disappear as in figure 5.29 was shown) in the dataset a site-specific change can hardly be defined. In these cases the closest point method assigns too big changes. Yet the normals method used the half of the thickness what gives the more realistic results.

5.6.8 Summary

Table 5.9 summarizes the following subsections. Overall we recommend to use the closest point method.

5.7 Recommendations

This section contains some important recommendations in connection with the geometric site-specific change measurements.

- As default the **search depth** is set to 1mm that all changes are found for sure. In connection with the closest point method we did not find a problem with setting the search depth large. Yet if you are using the normals method be aware that very few points (less than 0.01%) are assigned with wrong intersections because of the interpolation of the normals during the orientation checking if the search depth is large. Therefore try to use an adequate search depth in combination with the normals method looking at the histogram or using the *generateNormals* setting (see also 5.2.4).
- If the **search depth** is too short not in all points a site-specific change can be assigned.
- Be aware that the results of the closest point and the normals method are not the same because the definition of the site-specific change is different.
- Usually the closest point method should be chosen (see also 5.6).
- For bone apposition use the closest point method because the normals method doesn't work for forming trabeculas.

- For small site-specific changes the closest point and the normals method have very similar computation times. For bigger changes (more than 0.01mm) the normals method is faster (compare to 5.5.1).
- If you plan a statistical analysis of the results do not close the boundaries of the bone prior to the site-specific calculation. For example when performing a site-specific measurement on a bone that has closed boundaries there would be additional measured values on the boundary which are all zero. In such a case the calculated statistics would be skewed.
- The mean vector (5.4.2.2) of the site-specific change can be used to identify a shift of the data sets under consideration. You should visually check the data if the magnitude is bigger than 0.01mm.
- Be aware that the mean change (5.4.2.2) contains positive and negative changes of the bone. Compare to the number of apposition and number of loss points 5.4.2.1.
- Always use the `SetInput()` function of the *vtkSiteSpecificChange* for the surface taken earlier in time and the `SetSource()` function for the surface later in time (see also B.1.1). Otherwise apposition and loss are switched. We recommend to use numbers for the surfaces where smaller numbers represent scans taken earlier in time.
- Use the histogram to get an idea of the distribution of the site-specific changes (see also 5.4.3). If the peak is very broad you should visually check the changes and look for translations of the data.
- If the shift of the two data sets under consideration is more than $\frac{1}{4}$ of the trabecular thickness you should not rely on the results of the statistics any more.
- Be aware that a bad registration always has an influence on the results.
- Macroscopic changes of the bone (e.g. movement of individual trabecula) may lead to misinterpretation of the results.

Chapter 6

Deformable Registration to find the site-specific bone change

6.1 Introduction

As described in [16], the registrations can be separated in rigid and non-rigid categories. If you apply only rigid transformations to an object (translation, rotation), it does not change its shape whereas by applying non-rigid (non-linear) transformations, the shape changes. A deformable registration allows all kind of transformations and it therefore is categorized as non-rigid transformation.

The idea behind using a deformable registration to determine the bone change is to assume that bone is not a static material. It changes its shape over time which makes it a candidate for deformable registration. The advantage of a deformable registration is that with this method the elasticity of the material can be taken into account.

The result of a deformable approach is a deformation field consisting of a vector at each voxel position representing the local shift between the two images at that point. Obviously, it is much more difficult to extract the effective site-specific change from a deformation field than from the resulting vectors of the geometric approach (see chapter 5). This is one reason why we do not calculate the site-specific change with this method (although the information is there and it would be possible). Another reason is that the geometric approach is much faster.

The main reason for implementing this method is the possibility to easier extrapolate the bone change into the future and to actually apply the calculated bone change in order to get an image of the future structure of the bone (prediction).

The jargon of the registration field includes the names "fixed and moving image" to express that one image is shifted and changes its shape while the other one keeps always the same shape (reference). In the following chapter, we use these two terms too whereas the moving image always corresponds to the one taken earlier in time and the fixed image to the later one.

6.2 Calculate a global deformation pattern using Demons Deformable Registration

6.2.1 General idea of this method

One approach to calculate the deformable registration is to use the demons method [2]. This method is very fast what is a very important aspect when dealing with large image data.

The principal idea behind this method is to push the moving image in the direction of the largest gradient of the fixed image (normal to the fixed surface). The orientation (+/-) of this force depends on the fixed image point lying inside or outside the moving image. This tends to sort the inside and outside points of the moving image. The name "demon" arose from "Maxwell's demons" [2] because those two principles are quite similar.

The demons approach is different than most other deformable models which define the force in the direction of the closest point with the same intensity and which are often computationally quite complex.

The result of the demons deformable registration is a "deformation field" containing at each voxel a vector with the shift from the moving to the fixed image (from the one earlier in time to the later one).

To get this "demons force" in a certain voxel, one has to first calculate the gradient of the fixed image in that point which leads to the normal vector. In a second step, the orientation of the theoretical force has to be defined by comparing the two intensity values (fixed and moving image) at that point. If the fixed image point lies inside the moving image (voxel value of the fixed image is smaller than the one of the moving image), the direction of the force is in the direction of the gradient, weighted by the difference in the intensities. Otherwise, the force is in the opposite direction of the gradient. This behaviour is illustrated in figure 6.1 whereas f represents the fixed image and g the moving image. The gradients ∇f are shown by the blue arrows and the resulting demons force v by the red arrows. In the left image, the fixed image lies outside the moving image ($f(p) > g(p)$) and therefore the force has to be in the opposite direction of the gradient. The right image shows a situation where the fixed image lies inside the moving one and therefore the direction of the force and the gradient are the same.

The basic formula describing this behaviour looks like this:

$$\vec{v} * \vec{\nabla f} = g - f \quad (6.1)$$

whereas $\vec{v}$ is the deformation, $\vec{\nabla f}$ the gradient at a certain point and $g - f$ is the difference in the image intensities between the moving and the fixed image. In figure 6.2 this formula is visualized.

It can be seen that if the two images have a constant gradient over a large part of the image, the registration is perfect. Unfortunately, it is never like this in a real image.

To get good results nevertheless, the algorithm has to be performed several times. This means that you have to iterate several times over the image to get good results.

Equation (6.1) leads to the following formula (formula (4) in Thirion's paper [2]) to calculate the deformable field:

$$\vec{v} = \frac{(g - f) * \vec{\nabla} f}{\vec{\nabla} f^2 + (g - f)^2} \quad (\text{formula (4), asymmetrical formula}) \quad (6.2)$$

in which f is a voxel intensity of the fixed image and g the value of the same voxel in the moving image. The term $(g - f)^2$ has been added to the denominator to limit the result if $\vec{\nabla} f$ is a small number. This formula was implemented in ITK ([3] and [4]) (although it took us quite a while to realize that there was an error in this filter).

However, this formula has a problem with large bone changes in a rather sharp image: It can happen, that the resulting deformation is small although there should be a large shift. This problem is illustrated in figure 6.3, point p, where the gradient of the fixed image is zero or almost zero. Even though the term $g - f$ is big, the resulting "force" gets small which results in a poor registration at those locations.

This problem is due to the formula (6.2) just considering the gradient of the fixed image. This phenomena is shown in the evaluation part in section 6.4.3.

That's why we decided to implement ¹ the formula (5) out of Thirion's paper [2]. For simplicity reasons, we call this formula "symmetrical demons formula":

$$\vec{v} = \frac{2 * (g - f) * (\vec{\nabla} f + \vec{\nabla} g)}{(\vec{\nabla} f + \vec{\nabla} g)^2 + (g - f)^2} \quad (\text{formula (5), symmetrical formula}) \quad (6.3)$$

¹This filter has been contributed to ITK and is called `itkSymmetricForcesDemonsRegistrationFilter`

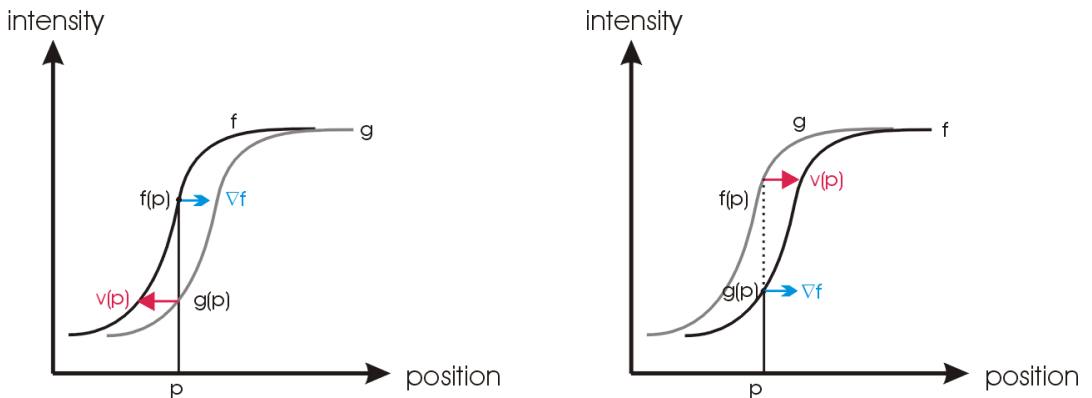


Figure 6.1: The moving image g is pushed by the demons force in the direction of $-\nabla f$ if $f(p) > g(p)$ (left picture) and according to ∇f otherwise.

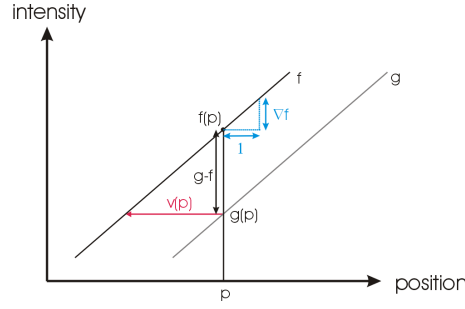


Figure 6.2: Visualization of the basic demons formula described in formula (6.1).

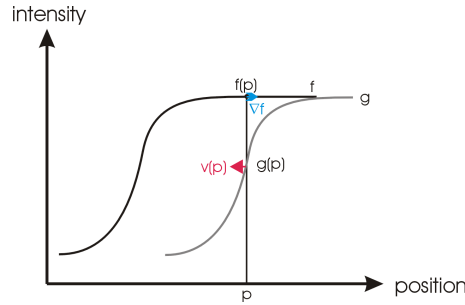


Figure 6.3: Problem with the demons formula (6.2): If you have large bone changes and a rather sharp image, the gradient (and therefore also the "demons force") is very small at positions where there should result a large deformation (point p).

This formula takes - additionally to the fixed gradient - the moving gradient into account, which is large at the above mentioned positions. This results in a force in the right direction at large bone changes where formula (6.2) has problems.

The only small disadvantage about this formula is that, additionally, the moving gradient ($\vec{\nabla}g$) has to be calculated which makes this algorithm about 20% slower.

For further information please consult Thirion's publication [2].

6.2.2 Advantages and disadvantages of the demons registration method

A big advantage of this method is that, compared to other deformable algorithms, it is computationally *very fast* and *fully automated*. This means that neither any initial matching points nor any material parameters have to be set. In fact, you just have to define the number of iterations to be performed and a standard deviation (see section 6.2.4).

Of course this method also has some disadvantages:

- The decision if a point in one image lies inside or outside the object in the other image, is made with the intensity value at that point. For that reason

it is very important that the *intensities of the two images* are *similar*. In our case, for matching two images of the same bone taken at different points in time (intramodal registration), this is not a problem. If you did an intermodal registration, you would have to first match the histograms of the two images before you could register them.

- Another weak point is the *sensitivity to the initial positioning* of the two images to be registered. This means that if you calculate the deformable field twice with two different initial positions of the images, you get two different results. Therefore it is very important that the images are registered well!
- The third restriction is that if the change in the bone is too large, the demons algorithm will not work any more. A "too large change" means that a part of the bone (i.e. a trabecula) does not overlap with the same part in the other image. This could happen if the two images are registered poorly or if there is a large shift in the bone.

6.2.3 Speeding up the algorithm: Multi-resolution pyramid

The most often used method to improve computational efficiency is the multi-resolution pyramid (see figure 6.4). This means that the images are resampled at one or more lower resolutions. The two images with the lowest resolution are then registered and the result is set as the initial deformation field to the next higher level of the pyramid. Like this a smaller number of iterations is needed at the level with the highest resolution - the slowest level.

It is important to realize that it is not useful to have as many pyramid levels as possible: If the resolution is too low, the structure of the bone is not the same anymore (it is possible that some trabeculae disappear because they are too "thin" for the resolution) and the initial deformation field gets completely wrong. This has to be corrected at the higher levels of the pyramid which takes many iterations. At the end, you get an even worse result than with just 2 levels!

It has turned out that for μ CT bone images (with a resolution of $34\mu m$) a multi-resolution pyramid of two levels is appropriate. It can speed up the algorithm by about 40% compared to just one level (see section 6.4.9).

Using three levels is not very useful, at least not with the kind of data we had at the time of writing this report. In the future - with better resolutions - it might be possible to speed up the calculation with three levels. Another reason not to use three levels is that even with two levels it is hard enough to find reasonable parameters, with three levels this gets almost impossible!

Guidelines how to make a good choice of the number of levels and the number of iterations will be given in section 6.5.

A multi-resolution pyramid has been implemented in the `itkDeformableRegistrationFilter`, which is described in section B.7.

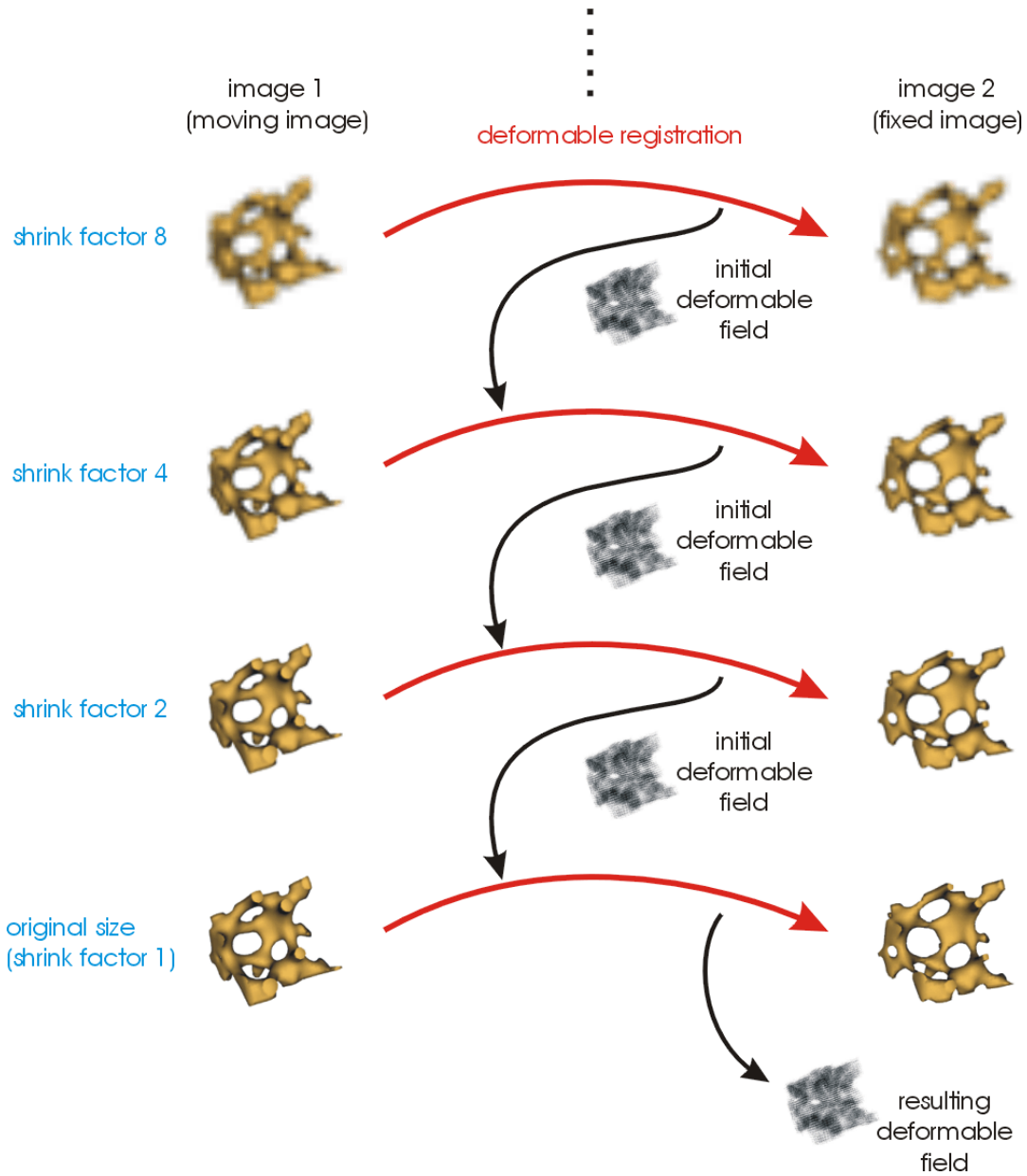


Figure 6.4: Visualization of the multi-resolution pyramid. This picture shows a pyramid with four levels. Note, that it is not useful to use so many levels. Mostly 2 or 3 levels are enough, depending on the image resolution.

6.2.4 Standard deviation

The standard deviation corresponds to the deformability of the bone and is a factor of how much the deformable field is being smoothed between each iteration. This smoothing is done with a gaussian filter whereas the standard deviation to be set corresponds to the standard deviation of the gaussian filter. Therefore, if a large standard deviation is chosen, big changes (may be wrongly calculated) in the deformable field

disappear and you get just very slow and smooth changes in the surface of the bone. This simulates a rather stiff material. On the other hand, by choosing a very small standard deviation, the deformable field gets hardly smoothed at all and you might get wrongly registered areas (big errors). This kind of standard deviation would be chosen for a very elastic material.

Additionally to the standard deviation, a kernel width has to be specified for a discrete gaussian filter. In ITK, this value is calculated with two variables: a maximum kernel width and a maximum error. The kernel width is then chosen as small as possible, however, the error occurring from making the width smaller is not allowed to be bigger than the maximum error. Usually, the values of these two variables do not have to be changed. The default values for the maximum error is 0.1, the one for the maximum kernel width is 30.

A good choice when registering simulated bone is a standard deviation of 0.8 to 1.0. In chapter 6.4.7 it is shown how the resulting deformable field depends on this factor and why a value of 0.8 to 1.0 is recommended. For real bone, it has turned out that a value of 1.0 to 1.5 is appropriate.

6.2.5 Number of iterations

The number of iterations has a big influence on how well the registration is performed. For example if you have just one iteration, the registration is very poor. However, this does not mean that the result gets perfect if you perform a very large number of iterations. It has shown that for small and smooth changes (e.g. SIBA), you get a quite good result after about 3 to 5 iterations. For real bone loss, however, about 10 to 20 iterations are needed (sometimes even more). For more information about the choice of the number of iteration for a single-resolution registration see chapter 6.4.7 and 6.5.

If you would like to use a multi-level registration, a reasonable number of iterations at the first level (lower resolution) is around 2; for the second level 3 for simulated data and 5 to 10 for real bone loss (see also section 6.4.9).

6.2.6 Memory consideration

The output of the algorithm is an image of the same size as the two input images, consisting of a 3-dimensional vector at each point (in the case of 3D-images). The data type of this vector can be chosen at the definition of the function (see chapter B.7).

If the data type is chosen to be *float*, the memory usage of the resulting image will be 12 Bytes times the image size, resulting in 12MB for 1 million image points (100x100x100 image).

6.3 Calculate a global deformation pattern using FEM Deformable Registration

As there is a filter in ITK which calculates the deformable field based on finite element models, we tried this method as well to see how well it works. At the time of writing this report, the result was quite disappointing. On the one side, the algorithm is much slower than the demons registration (by more than ten times) and, on the other hand, the result gets much worse. Another disadvantage is the necessity to look for the best parameters, which is quite time consuming, whereas the demons algorithm requires just two parameters which are not that critical.

In section 6.4.11, the best result we could achieve is shown and compared to the demons approach.

In the beginning, we planned to include the FEM registration in the DeformableRegistrationFilter so that it would be possible to switch from the demons to the FEM algorithm by just setting a variable. However, as it turned out that this algorithm works that much worse (at least at the time of writing), we decided to just include the demons approach.

6.4 Evaluation of "DeformableRegistrationFilter"

6.4.1 Evaluation Method

To evaluate the demons deformable registration method (point 1 in figure 6.5), we applied the deformable field to the moving image (point 2). The application of a deformable field to an image can be done in ITK by using the itkWarpImageFilter.

Then we compared the reconstructed fixed image with the original one (point 3). On one hand we did this with a metric (mean square difference of the intensities between the fixed and the moving image), but then you just get a number telling you how well it is registered. As we needed more information, we decided to use, additionally, the SiteSpecificChangeFilter described in chapter 5. With this filter we know exactly where the registration was poor and can visualize it. Additionally, we get some statistical values like the mean and median errors and the standard deviation.

We decided to use for all measurements the closest point method as this method seems to be better than the normal method.

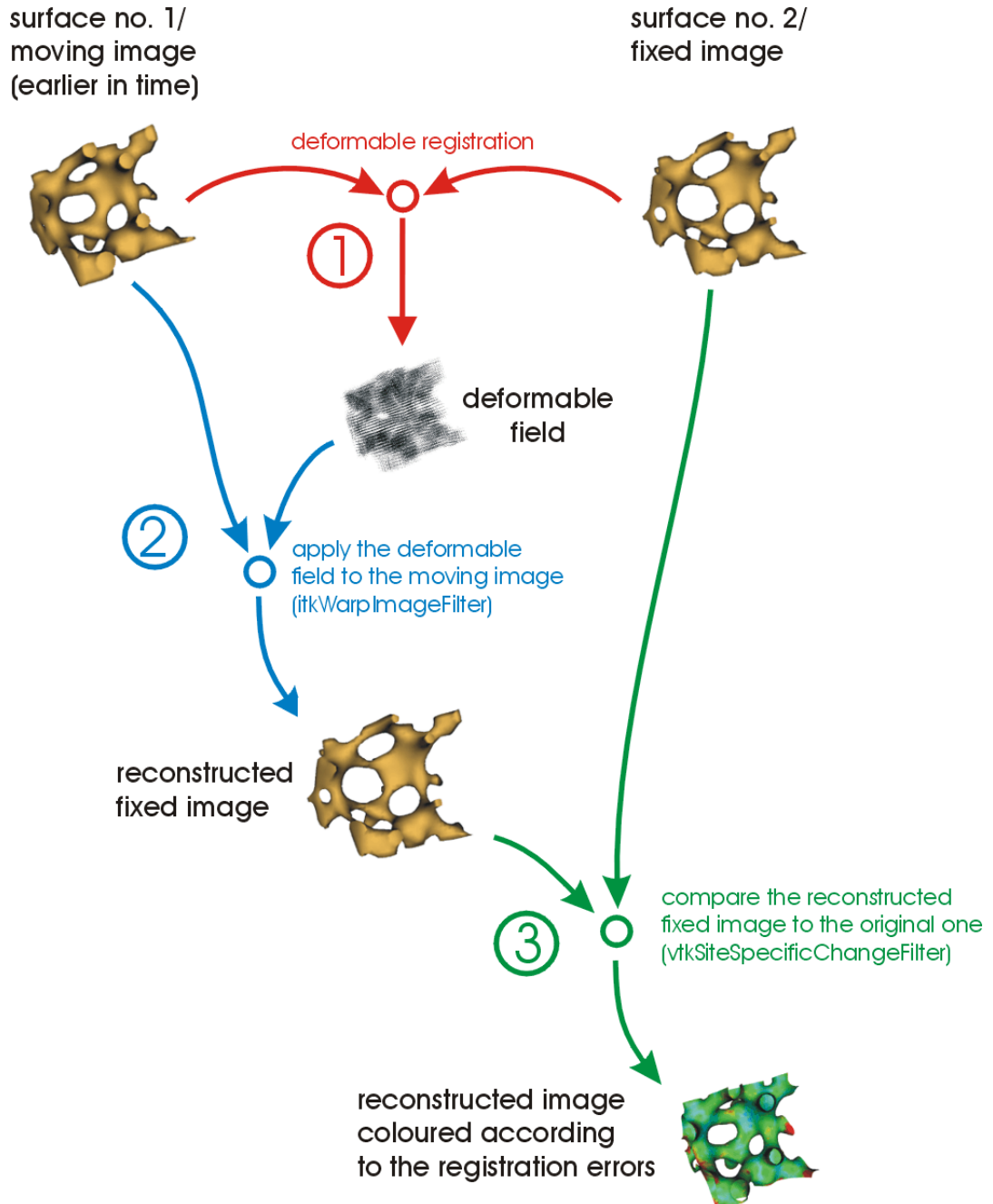


Figure 6.5: This graphics shows how we evaluated the registration.

6.4.2 Performed Tests

This chapter has nine main sections:

- **Accuracy of the Demons Deformable Registration with Simulated Bone Atrophy (6.4.3)**

In a first test, we wanted to see how well the demons registration works for simulated bone atrophy.

- **Accuracy of the Demons Deformable Registration with the Asymmetrical Formula (6.4.4)**

To see the improvements of the symmetrical formula compared to the asymmetrical one, we show one example of the asymmetrical registration.

- **Accuracy of the Demons Deformable Registration with Real Bone Atrophy (6.4.5)**

In this section, the filter is tested with real bone loss.

- **Comparison of the Site-Specific Change calculated with the Geometric Approach compared to the one extracted from the Deformable Field (6.4.6)**

To show that the site-specific change can also be extracted from a deformable field, this section has been added.

- **Influence of the standard deviation on the registration (6.4.7)**

The standard deviation is a parameter, which has to be set in the demons registration filter. As this parameter has quite a big influence on the result, an example is shown in this section.

- **Computation Time (6.4.8)**

This section talks about the computation time of the demons registration as a function of the number of iterations and the image size.

- **Computation Time for multi-resolution techniques (6.4.9)**

As a multi-resolution pyramid gets quite important with larger image sizes, an example of the computation time (2 level pyramid) is shown.

- **How does the deformable field look like (6.4.10)**

To get a insight into how the deformable field looks like, a very simple example has been made to show the field.

- **FEM Registration (6.4.11)**

In this section, an example of the FEM deformable registration is shown.

6.4.3 Accuracy of the Demons Deformable Registration with Simulated Bone Atrophy

To get insight into how well the demons algorithm works, several series of registrations with different number of iterations were done. Two of them are shown in this section:

- The first one is an example which could occur in a real case with small changes in most regions and some large ones where a trabecula is resorbed. Some pictures for different number of iterations are shown in figure 6.6, whereas figure 6.7 shows the appropriate plot containing the mean and median error to a perfect registration as well as the standard deviation of the mean error, the metric value and the RMS Change (root of the mean square change in the deformable field).
- The second example (figures 6.8 and 6.9) is a very extreme one which means that the images are quite different due to many trabeculae which are absorbed.

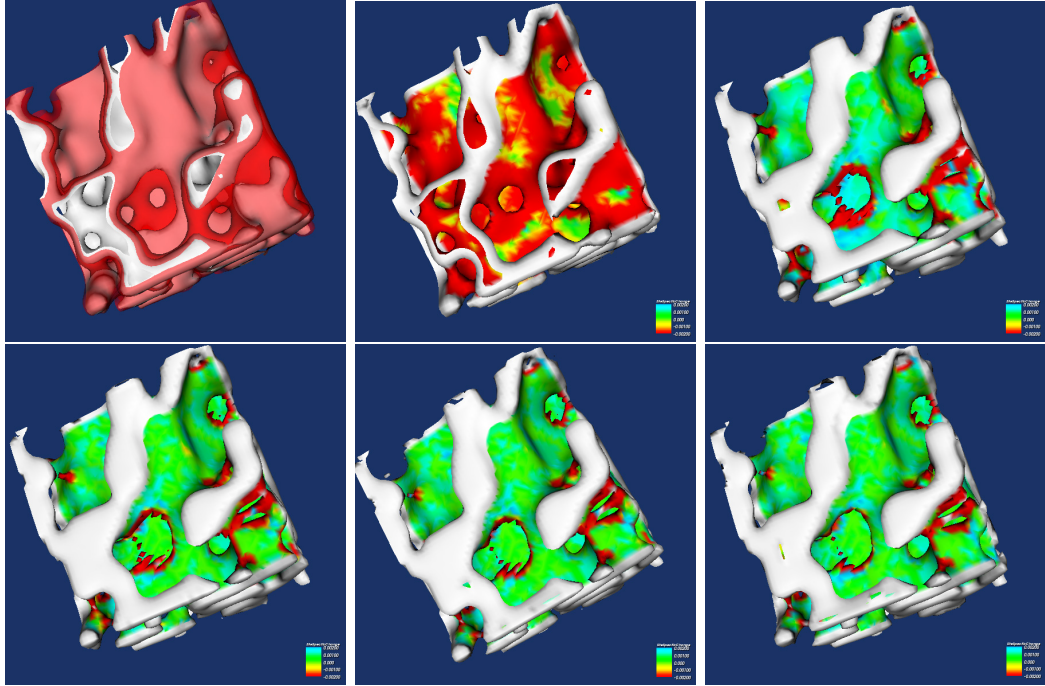


Figure 6.6: Registration at different number of iterations (0, 1, 2, 3, 5, 10) and a fixed standard deviation of 1.0. The colors represent the error to a perfect registration calculated with the closest point method (SiteSpecificChangeFilter) (images: dog03_gauss_sub.aim and dog05_gauss_sub.aim)

The first picture in figures 6.6 and 6.8 respectively (on the upper left) shows the two original images which were the input into the filter. Thereby, the white image is the image later in time and the red, transparent image the earlier one. Therefore, the red image should be mapped onto the white one. The other five pictures are results of the demons algorithm, each of them with a different number of iterations (1, 2, 3, 5 and 10) and a standard deviation of 1.0, respectively 0.5. The colors represent the registration error calculated with the closest point method described in chapter 5.3. The range of the scalar (colors) is always chosen equally ($-2\mu m$ to $2\mu m$) so that the same error in different pictures is always represented by the same color. At locations where there is no error, which means that the original and the reconstructed images are the same, the bone is colored blue. In every picture, there can be seen

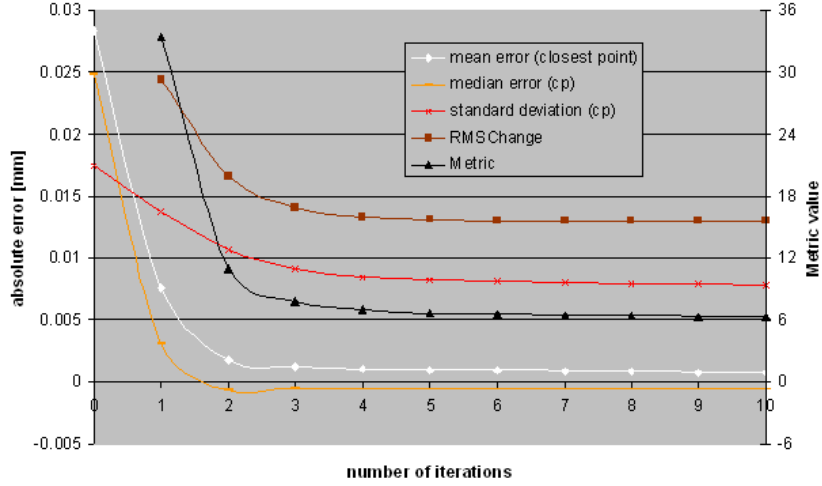


Figure 6.7: Mean error, median error and standard deviation of the registered images showed in figure 6.6, calculated with the geometric approach (SiteSpecificChangeFilter). Additionally, the metric value and RMS Change are shown.

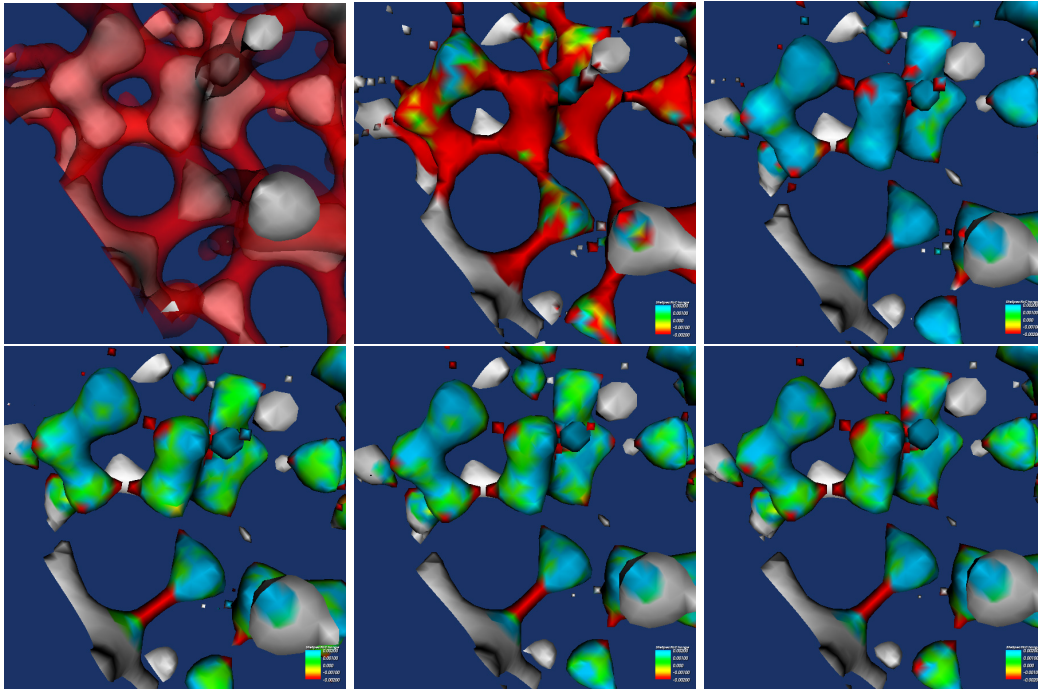


Figure 6.8: Registration at different number of iterations (0, 1, 2, 3, 5, 10) and a fixed standard deviation of 0.5. The colors represent the error to a perfect registration calculated with the closest point method (SiteSpecificChangeFilter) (images: siba1_gfilt_sub_small.aim and siba0_gfilt_sub_small.aim)

a white border. These points were excluded in the SiteSpecificChangeFilter due to border errors which would lead to incorrect statistical values. The pictures in the

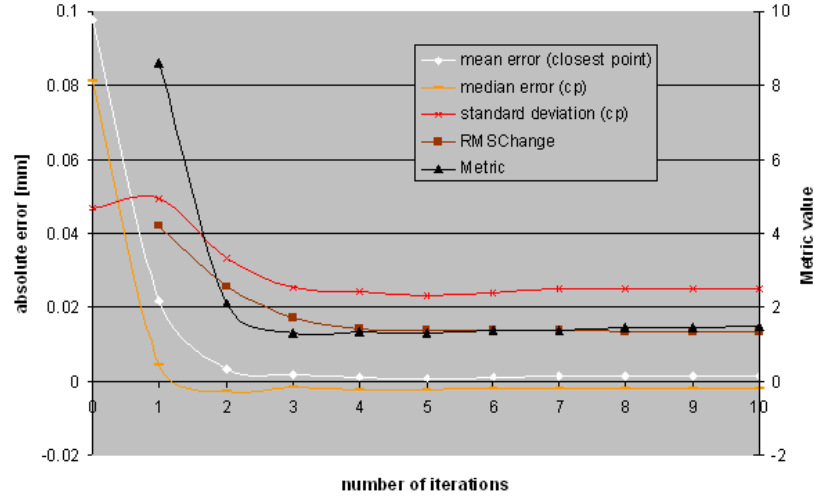


Figure 6.9: Mean error, median error and standard deviation of the registered images showed in figure 6.8, calculated with the geometric approach (SiteSpecificChange-Filter). Additionally, the metric value and RMS Change are shown.

bottom right corner show the resulting images after the algorithm has converged. When performing more iterations, the deformable field does hardly change anymore (except if it gets unstable, e.g. when choosing a small standard deviation).

In the charts 6.7 and 6.9, it can be seen that the registration reaches an asymptote after 3 to 5 iterations. In these plots, the negative values of the mean and the median error have been taken to make the chart more clear. Negative values in the original mean and median error (positive values in the chart) mean that there was a "bone loss" from the reconstructed image to the original one. In other words that the registration went not far enough.

If you look at the mean error, you can see that it is around $-1\mu m$ for the first example (canine) and about $-2\mu m$ for the second one (aluminum). Comparing these values to the mean site-specific change before registration, which is about $-28\mu m$ in the first series (figure 6.6), respectively $-100\mu m$ in the second one (figure 6.8), the mean error in the registration is around 2% to 4%. The standard deviation is about $8\mu m$ and $25\mu m$ respectively. However, one has to be careful with these statistical values: If you calculate the errors of the demons registration with the normal method, you would get slightly different results due to the different approach of this method. However, they are a good approach to see how well the registration works.

The other parameters we looked at are a metric value and RMSChange. The metric value is the mean square difference of intensity between the fixed and the moving image (exclusive a border of two voxel). This value tells you quite good how well the images are registered. The RMSChange is the root of the mean square change in the deformable field. This value never converges to zero because the deformable field is gaussian smoothed between each iteration. Therefore, when the algorithm converges, the change in the deformable field calculated with the demons formula is about the

same as the change resulting from the filtering.

Another thing you have to be very careful about, is the choice of the search depth in both geometric approaches (see chapter 5.2.4 and 5.3.4). If you choose too large a search depth, you get some (not many at all) wrongly calculated site-specific changes which, however, can falsify the statistical values quite strongly; mainly if you measure just very small "changes". By choosing the search depth too small, you might not take into account some poorly registered points!

In table 6.1, the above mentioned statistical values are summarized. The relatively big standard deviations can be explained with the trabeculae which do not disappear and result in quite large errors (red color in figures 6.6 and 6.8). The reason for the mean error being almost zero anyway, is that most of the points are registered a little bit too far (blue and part of the green color in figures 6.6 and 6.8), so that the mean value is almost zero. In figure 6.10, the histograms of the registration errors are shown. As the large changes from disappearing trabeculae cannot be seen in these histograms (on the left side of the peak in the first image; the number is too small compared to the well registered points), they are enlarged around very small changes. Like this, you have a histogram of the errors where no trabecula disappears. It can be seen that in these parts, the mean error is around $1\mu\text{m}$. The positive value means that the bone is about $1\mu\text{m}$ too small after the registration. If we compare this value to the mean site-specific difference between the two images before registration, which is $-28\mu\text{m}$, respectively around $-100\mu\text{m}$, the registration error where no trabecula disappears is smaller than 4%.

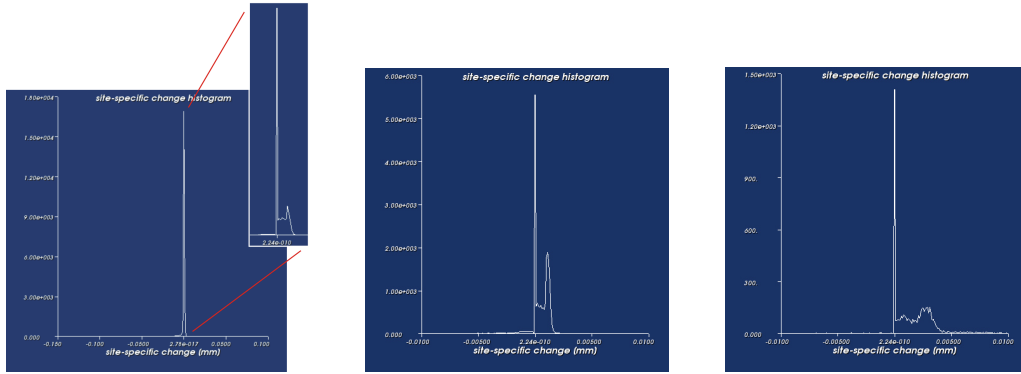


Figure 6.10: Enlarged histogram of the registration shown in figures 6.6 (middle) and 6.8 (right picture) after 3 iterations. On the left side, the original histogram is shown containing all measured site-specific changes.

The very small metric value of the aluminium example is probably due to the very loose bone so that a lot of voxel values are zero in both images and the error in that position is, therefore, zero.

Even though these results seem to be quite good, the absorbed trabeculae cannot be registered correctly. This is also the restriction of the demons registration method. It is sometimes (not always) not capable to register the two images correctly at a disappearing trabeculae or at an appearing hole. Yet there arises at the same

	example 1 (figure 6.7)	example 2 (figure 6.9)
mean error	$> -1\mu m$	$-2\mu m$
standard deviation	$8\mu m$	$25\mu m$
median error	$< 1\mu m$	$2\mu m$
mean site-specific change before registration	$-28\mu m$	$-100\mu m$
Metric	7	1.4
RMSChange	$13\mu m$	$14\mu m$
image spacing	$34\mu m$	$96.5\mu m$

Table 6.1: Statistical values of the difference between the reconstructed and the original image (two examples).

time the following question: How important is a perfect registration at these regions. The main purpose for using the deformable registration is to be able to predict the future structure of the bone. Even if the registration was perfect at the disappearing trabeculae, it would not be easier to predict the structure at these points because the amount of bone loss at these positions will for sure not be the same in the future as at that time step when the trabecula disappears.

Therefore, the most important thing is that the registration is as good as possible where no trabecula is absorbed, which is fully fulfilled by the demons registration.

6.4.4 Accuracy of the Demons Deformable Registration with the Asymmetrical Formula

To show the difference between the asymmetrical and the symmetrical demons formula (equation 6.2 and 6.3), the same measurement as with the symmetrical one in chapter 6.4.3 has been made with the asymmetrical formula. In figure 6.11, it can be seen that mainly at large bone changes (where a trabecula disappears), the registration is not as good as with the symmetrical formula. This is also confirmed by the chart in figure 6.12: The mean error is about $-3\mu m$ with a standard deviation of $15\mu m$. This is quite a bit worse than the registration with the symmetrical demons formula which resulted in a mean error of $-1\mu m$ and a standard deviation of $8\mu m$. Therefore, with the implementation of the symmetrical formula, an improvement of about 100% could be achieved.

This means that the asymmetrical formula is less suitable to register bone with a lot of disappearing trabecula than the symmetrical formula.

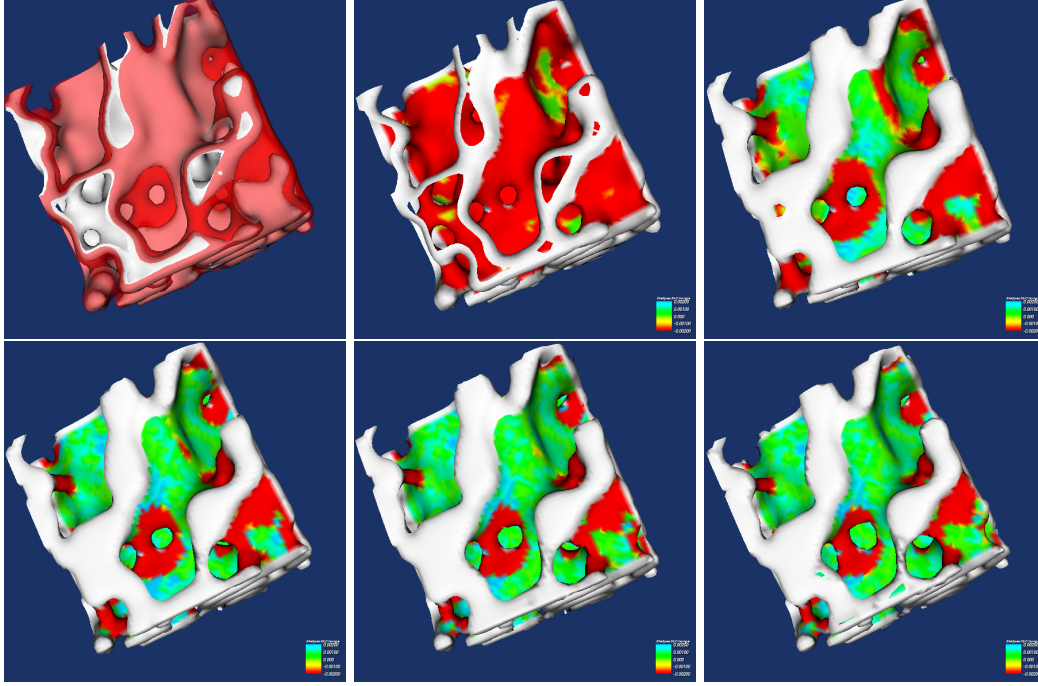


Figure 6.11: Registration with the demons formula 6.2 at different number of iterations (0, 1, 2, 3, 5, 10) and a fixed standard deviation of 1,0. The colors represent the error to a perfect registration calculated with the closest point method (SiteSpecificChangeFilter) (images: dog03_gauss_sub.aim and dog05_gauss_sub.aim)

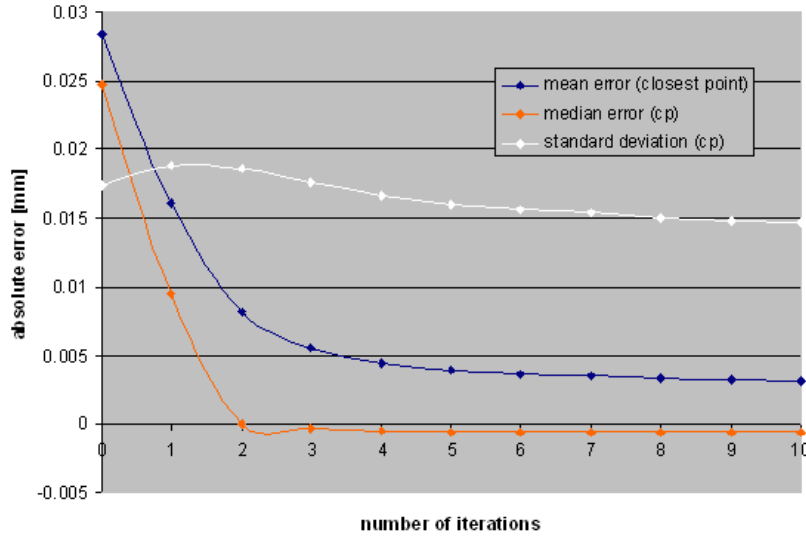


Figure 6.12: Mean error, median error and standard deviation of the registered images showed in figure 6.11, calculated with the geometric approach (SiteSpecificChangeFilter). Additionally, the metric value and RMS Change are shown.

6.4.5 Accuracy of the Demons Deformable Registration with Real Bone Atrophy

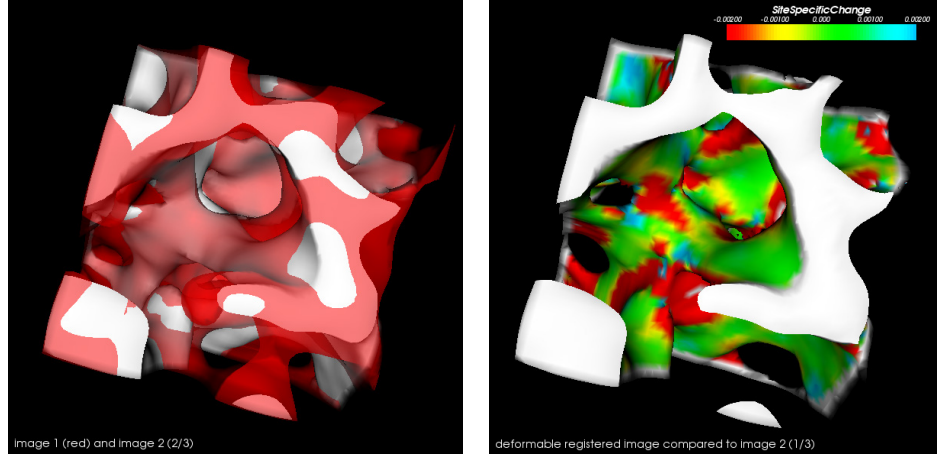


Figure 6.13: left: images to be registered; right: registered image colored according to the registration error (images: C0000711_sub_gfilt_regist12_sub_regist12_sub50.aim C0000978_sub_gfilt_regist123_sub_regist123_sub50.aim)

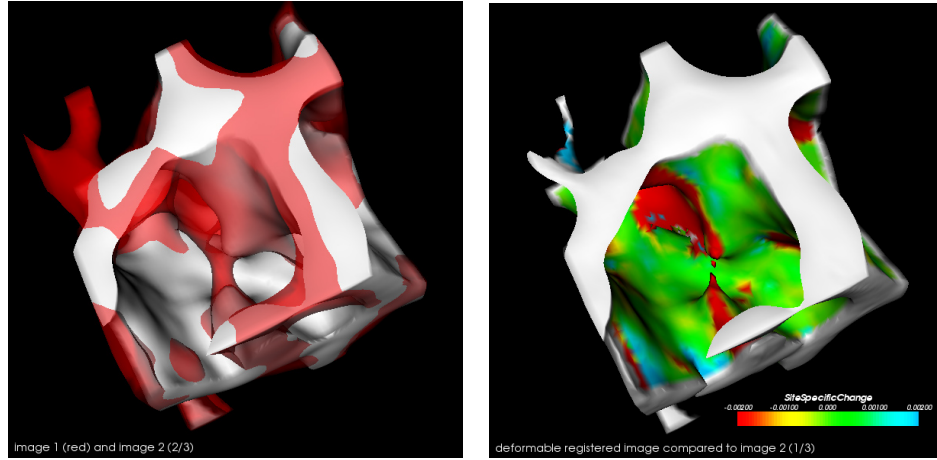


Figure 6.14: left: images to be registered; right: registered image colored according to the registration error (images: C0000751_sub_gfilt_regist12_sub_regist12_sub50.aim C0001018_sub_gfilt_regist123_sub_regist123_sub50.aim)

In figures 6.13 and 6.14, two examples are shown on how the demons deformable registration algorithm works for in vivo data. It can be seen that it works quite well in most parts of the bone. Wherever it is green, the error is less than $1\mu m$ (the scalar bar ranges from $-2\mu m$ to $2\mu m$). However, as already reported in the example with simulated bone loss, we can again recognize that the algorithm has a problem when a lot of bone is resorbed, e.g. the red part in image 6.14 where a trabecula disappears.

In figure 6.15, the error parameters for the example shown in figure 6.14 are shown.

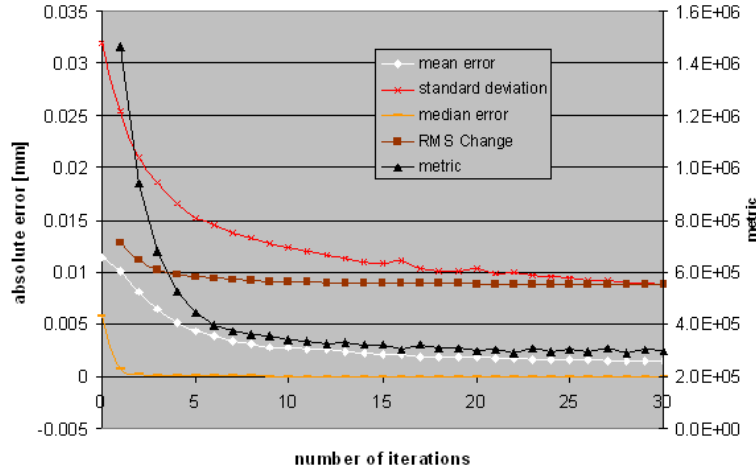


Figure 6.15: Chart showing the mean, median error, standard deviation, the metric value and the RMS change for the registered image shown in 6.14.

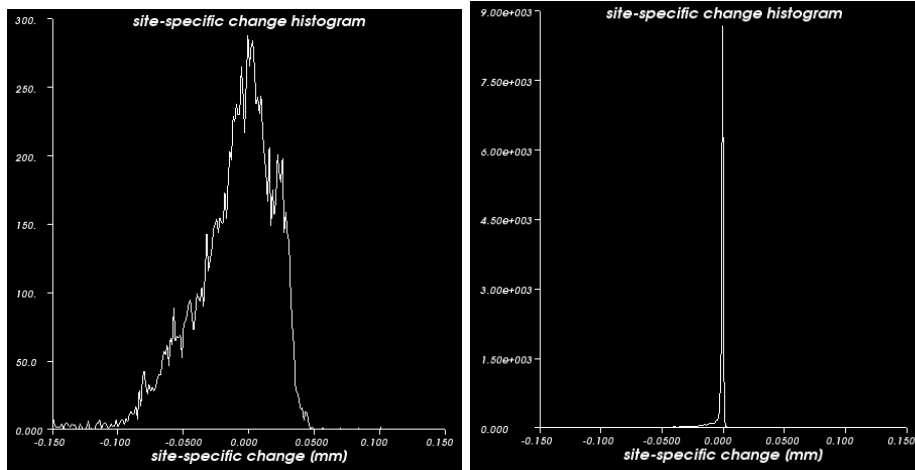


Figure 6.16: Histogram of the site-specific changes. Left: original images, right: registered images shown in 6.14.

It can be seen that the mean registration error goes down to about $2\mu m$ and the standard deviation to $10\mu m$. In the histograms in figure 6.16, it can even better be seen by what amount the initial images have been registered!

6.4.6 Comparison of the Site-Specific Change calculated with the Geometric Approach compared to the one extracted from the Deformable Field

The site-specific change is always calculated with the geometric method described in chapter 5. This is reasonable because this filter is much faster and it is always known what the filter is doing in each situation. But all the information is also available in

	geometric approach	deformable registration	difference
example 1	$-0.0283497mm$	$-0.0294231mm$	$0.0010734mm$
example 2	$-0.0465591mm$	$-0.0441459mm$	$0.0024132mm$
example 3	$-0.153022mm$	$-0.155242mm$	$0.00222mm$

Table 6.2: Comparison of the mean site-specific change calculated with the geometric approach and the deformable registration. (Parameters demons filter: standard deviation = 0.5; 5 iterations; inverse filter: 5 iterations; site-specific change filter: search depth = 0.25)

the deformable field. That's why we tried to extract this information and to compare it to the result from the site-specific change. This gives us an additional check on how well the deformable registration works.

To get the site-specific change from the deformable field, the field has to first be inverted (The reason for doing this will be clear after having read section 13.2.2). Afterwards, the site-specific change at each surface point on image 1 can be calculated by interpolating the inverted field in that point.

For the comparison, we used the mean site-specific change and the visualization of the changes.

In table 6.2, the mean site-specific changes are summarized. It can be seen that they match quite well. In these examples, the difference between the two methods is always smaller than $3\mu m$. As the mean value is just a limited criteria for comparing the two methods, the three examples have also been visualized (see figures 6.17 to 6.19). The first image on the left side always shows the two images between which the site-specific change is calculated. The second image is colored according to the site-specific change calculated with the geometric method. Whereas the image on the right is colored according to the extracted change from the deformable registration. The white border in these two images is because, in both approaches, voxel close to the border are not included as they are often wrong.

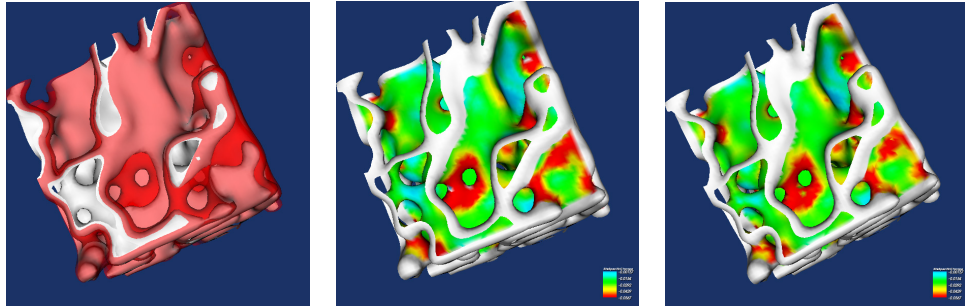


Figure 6.17: Example 1: Comparison of the site-specific change calculated with the geometric approach (middle image) to the extracted one from the deformable field (right) (images: dog03_gauss_sub.aim dog05_gauss_sub.aim).

In the first example, it can be seen that the site-specific change calculated with the

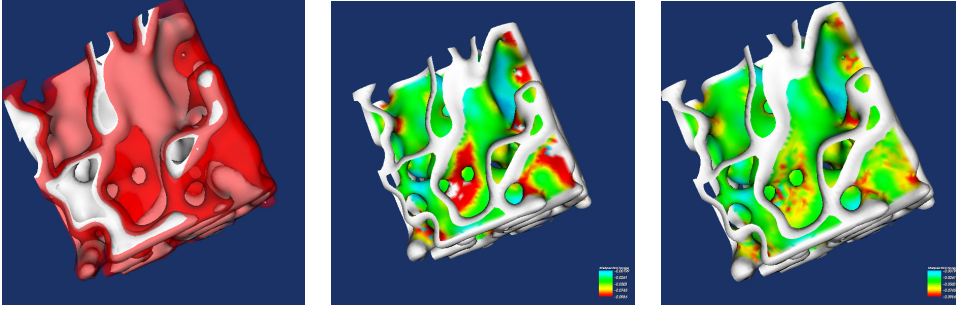


Figure 6.18: Example 2: Comparison of the site-specific change calculated with the geometric approach (middle image) to the extracted one from the deformable field (right) (images: dog03_gauss_sub.aim dog06_gauss_sub.aim)

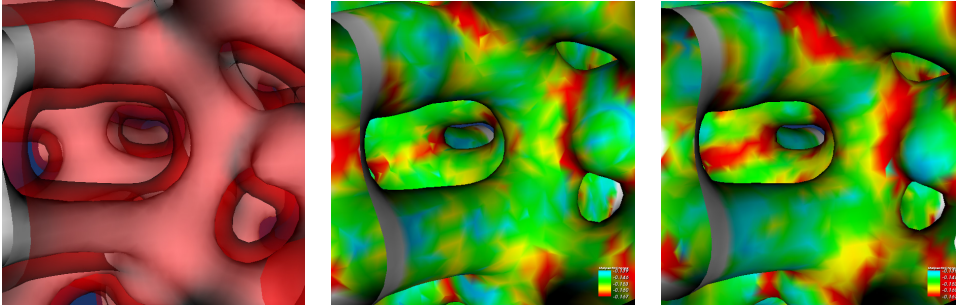


Figure 6.19: Example 3: Comparison of the site-specific change calculated with the geometric approach (middle image) to the extracted one from the deformable field (right) (images: siba4_gfilt_sub_small.aim siba3_gfilt_sub_small.aim)

deformable registration is almost perfect. However, with a larger bone change, the deformable registration gets worse. Again, it can be seen that the algorithm has mainly problems with disappearing trabeculae.

6.4.7 Influence of the standard deviation on the registration

The standard deviation is a very important parameter which has to be chosen when performing a demons deformable registration. This value changes the gaussian filter used to smooth the deformable field between each iteration and, therefore, represents in a way the elasticity of the material: A small value for the standard deviation results in a very elastic material whereas a big value represents a rather stiff model.

In figures 6.20 to 6.22, the mean error, the associated standard deviation and the median value on one hand and the metric and RMSChange on the other hand have been plotted for different standard deviations. The first three numbers have been calculated with the SiteSpecificChangeFilter using the closest point method.

When looking at figures 6.20 and 6.21, you would expect a suggestion for the standard deviation to be around 1.0. However, when looking at the metric value, which gives

6.4. EVALUATION OF "DEFORMABLE REGISTRATION FILTER"

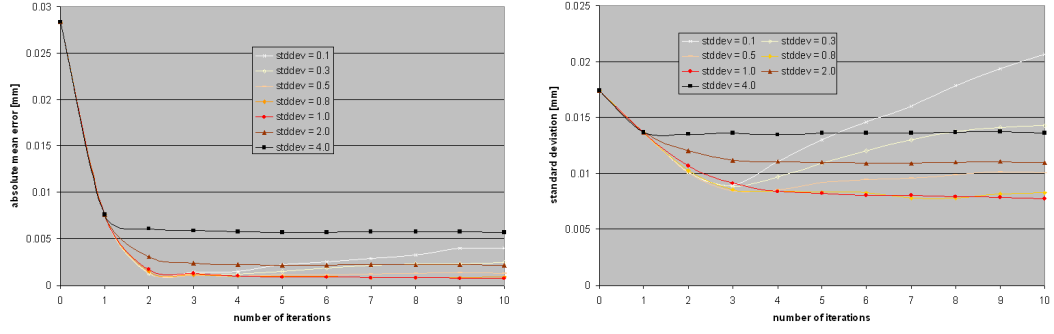


Figure 6.20: Mean value and the appropriate standard deviation plotted against the number of iterations for a 50x50x50 image (images: dog03_gauss_sub.aim and dog05_gauss_sub.aim).

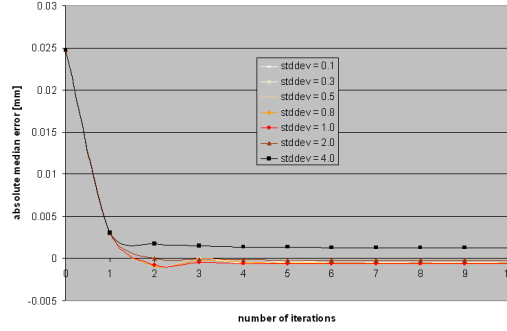


Figure 6.21: Median value plotted against the number of iterations for a 50x50x50 image (images: dog03_gauss_sub.aim and dog05_gauss_sub.aim).

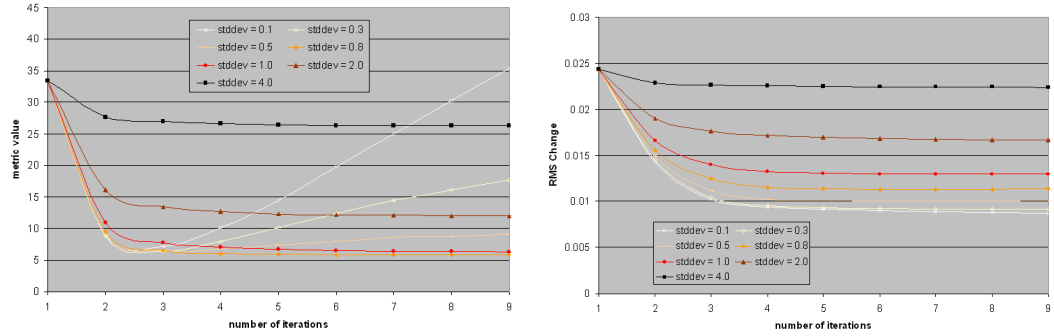


Figure 6.22: Metric and RMSChange plotted against the number of iterations for a 50x50x50 image (images: dog03_gauss_sub.aim and dog05_gauss_sub.aim).

the effective difference of the voxel intensities, a standard deviation of 1.0 is not optimal. The optimal choice, therefore, depends on what the result is needed for:

- If you would like the deformable field to be as accurate as possible over the

whole image, you should look for a **minimization of the metric value**. In this example, a value of about 0.8 would be fine.

- On the other hand, if the surface of the bone has to be registered as accurate as possible, the **mean value and its standard deviation** have to be taken into account. It is important to minimize the standard deviation of the mean value as well since this value is quite sensitive to bad registered points. A good choice in this example would be around 1.0.

In general, one has to be careful not to choose too small a value (e.g. a standard deviation of 0.3 and 3 iterations) because when performing more iterations than the optimal case, the registration can get unstable. The criteria for a good choice of the standard deviation is, therefore, a value as small as possible but not in the unstable region. If uncertain, rather choose a bigger standard deviation and perform more iterations.

The three plots show an average result. The results varied depending on the structure of the bone and the magnitude of bone loss between the two initial images. However, they all had in common that the best registration was achieved around a **standard deviation of 0.8 to 1.0²**.

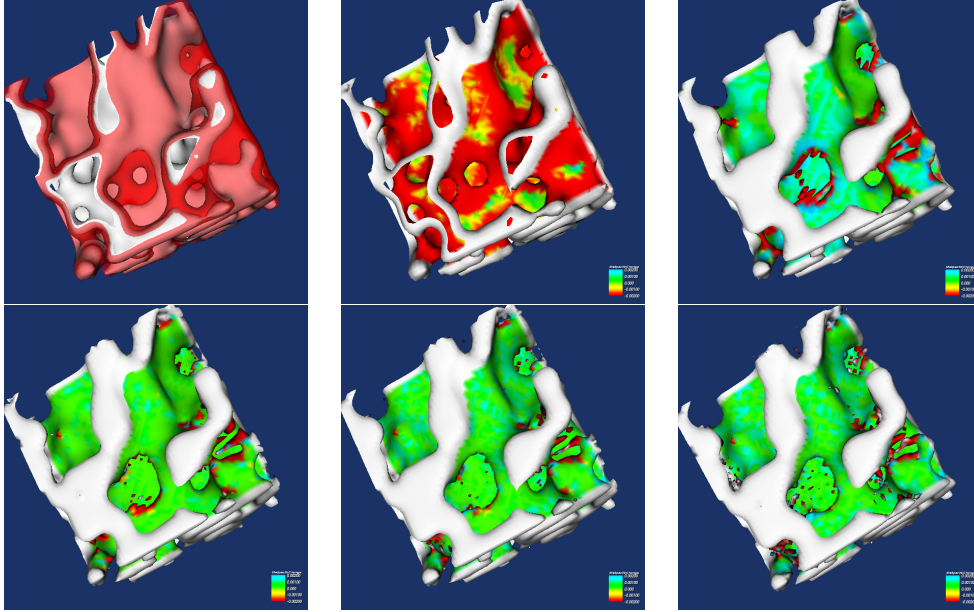


Figure 6.23: Registration after different number of iterations (0, 1, 2, 3, 5, 10) and a fixed standard deviation of 0.1. These images should be compared to the ones in figures 6.6 (standard deviation of 1.0) and 6.24 (standard deviation of 2.0) (images: dog03_gauss_sub.aim and dog05_gauss_sub.aim).

To actually see the difference in the registration with different standard deviations, two registration series with a standard deviation of 0.1 and 2.0 respectively are shown

²Note that these results are based on simulated data. For real data, the optimal value seems to be higher - around 1.0 to 1.5!

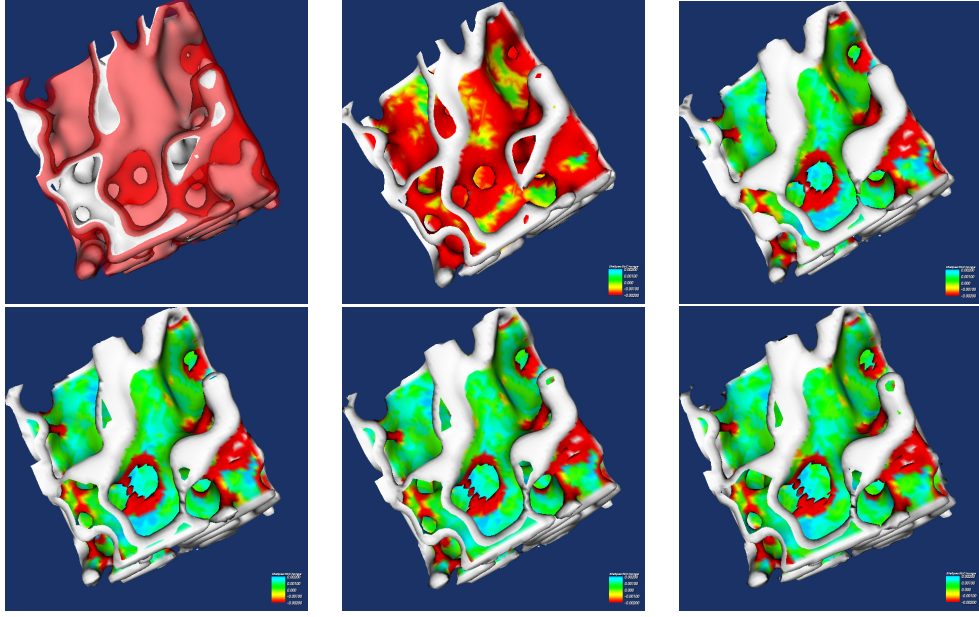


Figure 6.24: Registration after different number of iterations (0, 1, 2, 3, 5, 10) and a fixed standard deviation of 2.0. These images should be compared to the ones in figures 6.6 (standard deviation of 1.0) and 6.23 (standard deviation of 0.1) (images: dog03_gauss_sub.aim and dog05_gauss_sub.aim).

in figures 6.23 and 6.24. These are the same images as used to create the above charts (6.20 to 6.22) and in the series shown in figure 6.6 with a standard deviation of 1.0. It can be seen that in the example with a standard deviation of 0.1, there are pieces of bone reappearing where there should not be bone. This is what was mentioned before as being unstable. The example with a standard deviation of 2.0 is quite bad at locations where a trabecula falls away.

6.4.8 Computation Time

All measurements have been made on a 2.17GHz Computer with 1GByte of RAM.

The computation time of the algorithm is a function of the following parameters:

- **number of iterations**

The first one is the behaviour of the algorithm regarding computation time if the number of iterations is increased, taking always the same two images.

- **image size**

The other issue to look at is the computation time when the images get bigger, performing always the same number of iterations.

In figure 6.25, the computation time has been plotted against the number of iterations for a 50x50x50 image. The x-axis represents the number of iterations whereas the time

is represented by the y-axis. The different series were obtained by choosing different standard deviations for the gaussian filtering which smoothes the deformable field between each iteration (see chapter 6.2.4). It can be seen that the algorithm gets slower if the standard deviation is chosen bigger than one. This derives from the gaussian filtering which has to take into account more voxel due to the kernel width starting to get bigger (see also section 6.2.4). For values lower than one, the speed is independent of the standard deviation.

The other - and very important - thing which can be seen in this chart is the linear connection (order n) between the number of iterations and the computation time. This means that if you double the number of iterations also your computation time is doubled.

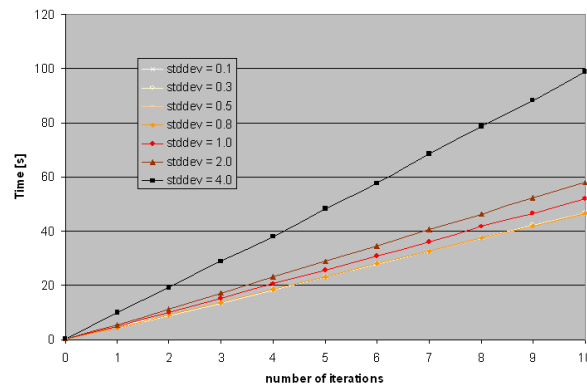


Figure 6.25: Computation time plotted against the number of iterations for a 50x50x50 image (images: dog03_gauss_sub.aim and dog05_gauss_sub.aim).

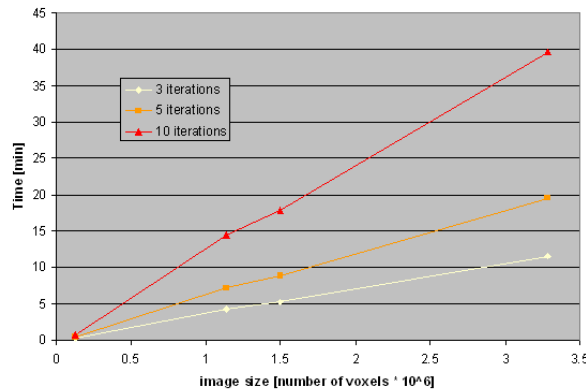


Figure 6.26: Computation time plotted against the image size (standard deviation = 1.0).

Also for larger images the time consumption seems to be of order n . This can be seen in figure 6.26.

6.4.9 Computation Time for multi-resolution techniques)

1. simulated data

It is not necessary to use a multi-resolution technique for small images. But for larger images (i.e. > 5 -10 million voxel), it is very important to use this technique as the required time can be reduced up to 40% with still the same result!

In figure 6.27, plots from different statistical values are shown. Each plot contains three different multi-resolution measurements based on simulated data: To get the white lines, one iteration was performed at the lower resolution whereas there were two to get the yellow lines and ten for the black lines. To compare these results, the appropriate outputs of the single resolution registration have been added (red lines).

It can be seen that the mean and the median error converge to the same value but that there are less iterations needed at the highest resolution if the multi-resolution technique is used. In this example, the standard deviation of the mean error gets bigger with an increased number of iterations at the lower resolution. Yet we also had another example where this standard deviation got

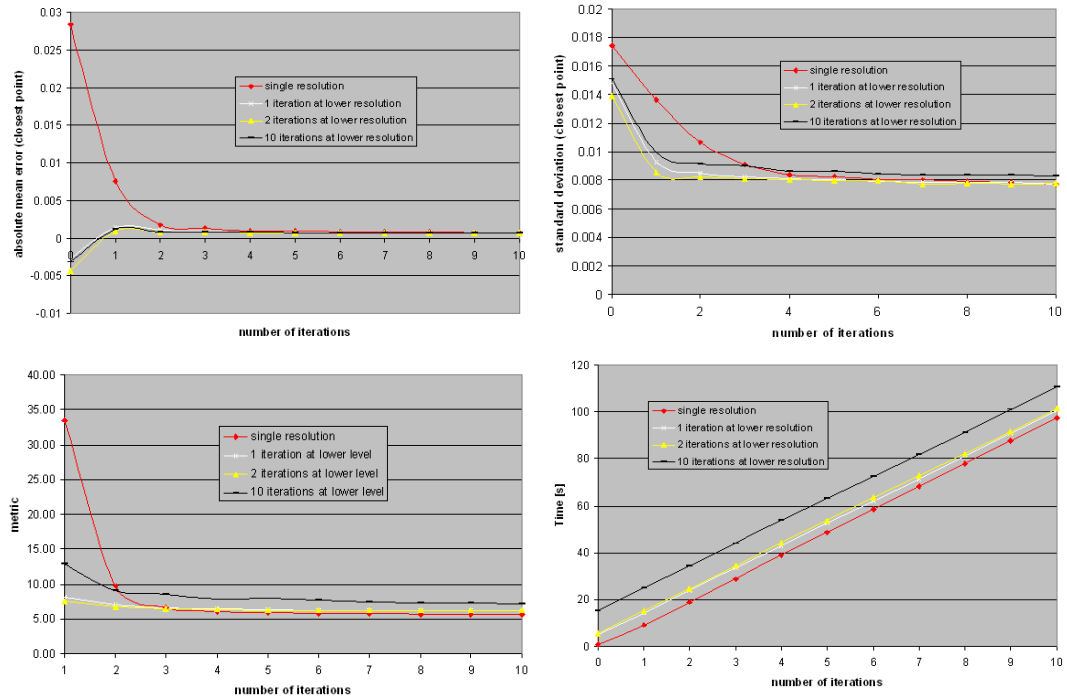


Figure 6.27: Mean error, standard deviation, metric value and the computation time for a multi-resolution registration with two levels (half the resolution in the first level) and a fixed standard deviation of 1.0. The red line respectively is the result for a single resolution registration. The white, yellow and black lines are results for different number of iterations at the lower resolution (1, 2 and 10) (images: dog03_gauss_sub.aim and dog05_gauss_sub.aim, 50x50x50 images)

better. However, if you use just 2 or 3 iterations at the lower resolution, this effect has no influence.

In the following it is shown, by how much the registration could be sped up in the case of the two images used to get the charts in figure 6.27. We assume that we would perform 4 iterations in the case of a single-resolution registration. To get the same result with the multi-resolution technique, we would choose 2 iterations at the lower level (yellow curve) and 2 iterations as well at the highest resolution. With this choice we can now compare the execution times of these two possibilities: The registration with the single-resolution would take around 40 seconds for the 50x50x50 images whereas with a multi-resolution pyramid you would get the same result in 25 seconds. This means that the execution time in simulated data can be almost halved!

2. real data

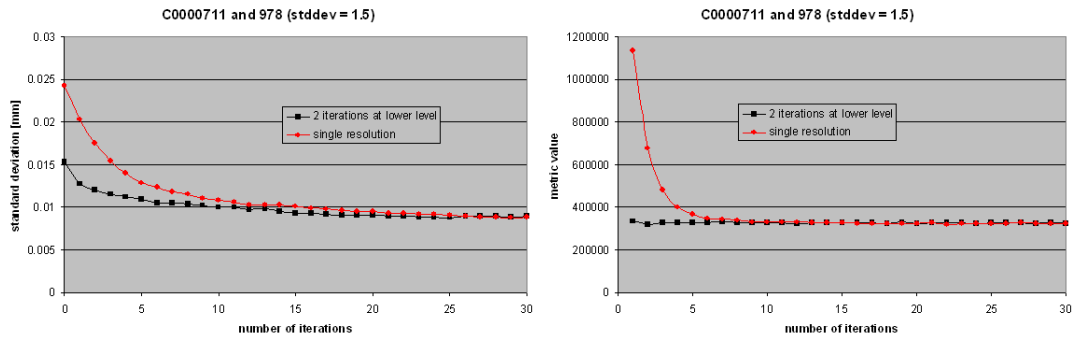


Figure 6.28: standard deviation and the metric value for a multi-resolution registration with two levels (half the resolution in the first level) and a fixed standard deviation of 1.5. The red line is the result for a single resolution registration whereas the black one represents 2 iterations at the lower resolution (images: C0000711_sub_gfilt_regist12_sub_regist12_sub50.aim C0000978_sub_gfilt_regist123_sub_regist123_sub50.aim, 50x50x50 images)

If we now look at real data, the situation gets a little bit harder. It is not as easy as with the simulated data to make a suggestion. Dependant on the parameter (mean, metric, ...) and the data we are looking at, the result is different. Two examples are shown in figure 6.28 and 6.29. By looking at the metric value, we could conclude that we can reduce the number of iterations from almost 10 to about 3 iterations. However, when we consider the standard deviation, there seems to be even an improvement after 30 iterations (figure 6.28, left). If we do not look at these very small improvements, we might conclude that we can reduce the number of iterations from 10 to about 5 to 7.

For the number of iterations at the lower resolution, a value of two seems appropriate for all data we were looking at. The optimal value for the standard deviation (demons deformable registration) seems to be 1.5 because for smaller values the results sometimes not converge properly (they oscillate, see figure 6.27).

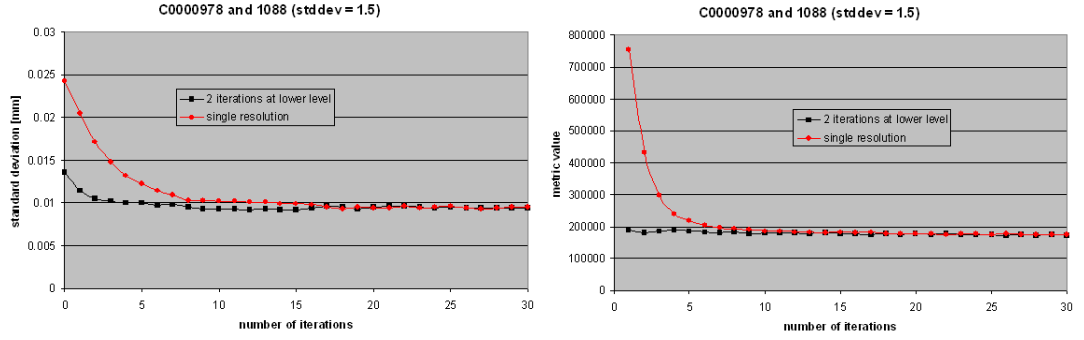


Figure 6.29: standard deviation and the metric value for a multi-resolution registration with two levels (half the resolution in the first level) and a fixed standard deviation of 1.5. The red line is the result for a single resolution registration whereas the black one represents 2 iterations at the lower resolution (images: C0000978_sub_gfilt_regist123_sub_regist123_sub50.aim C0001088_sub_gfilt_regist1234_sub_regist1234_sub50.aim, 50x50x50 images)

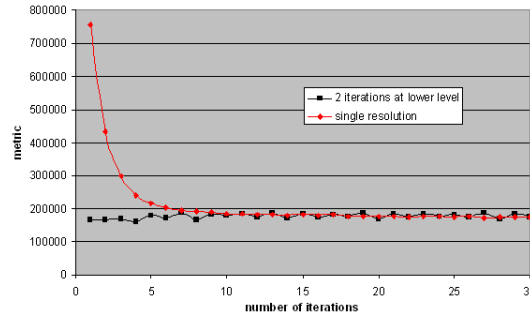


Figure 6.30: The metric value for a multi-resolution registration with two levels (half the resolution in the first level) and a fixed standard deviation of 1.0. In this chart it can be seen that the result seems to oscillate (same images as in figure 6.29).

If we now assume that we can reduce the number of iterations at the highest level from ten to six with 2 iterations at the lower level, we are able to reduce the time from 47sec to 34 seconds for a 50x50x50 image (see figure 6.27). This is an increase of about 30%.

6.4.10 How does the deformable field look like

To get an insight into how the deformable field looks like, two very simple images have been registered: We have chosen two plates, one of them with a thickness in x-direction of 9 voxel and the other one with 13 voxel whereas 1 voxel corresponds to 1mm (see figure 6.31). The amount which has to be registered is therefore 2mm. Like this you get a symmetrical deformation field which can easily be analyzed.

In figure 6.32, the resulting deformable field is shown. It has been colored according

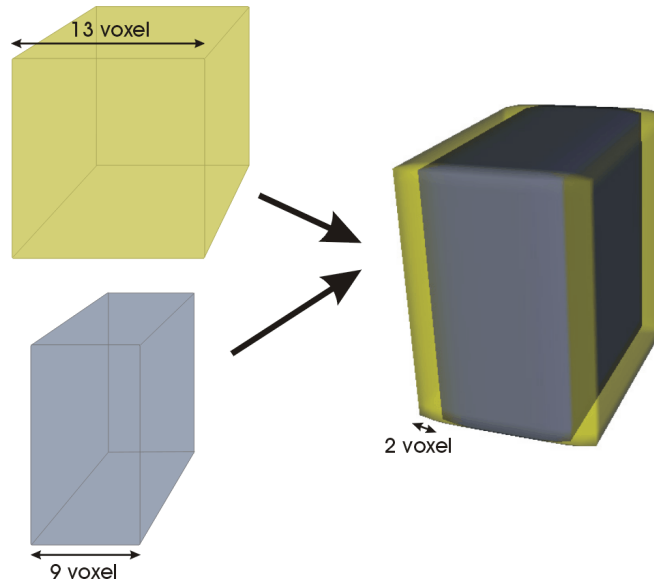


Figure 6.31: Images to be registered to show a very simple deformable field. The yellow image is the moving image (image 1) which has to be registered onto the grey one.

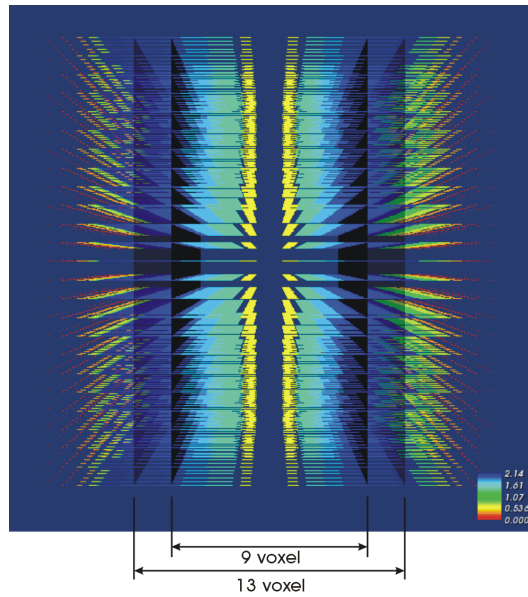


Figure 6.32: Deformable field resulting from registering the two images shown in figure 6.31

to the length of the vectors. The four black planes represent the borders of the plates whereas the two planes in the middle of the image belong to the plate with a thickness of 9 voxel (the grey one in figure 6.31) and the two outer planes to the 13 voxel one (yellow in fig. 6.31). The larger image is the moving image which has to be registered onto the smaller one.

If you look at the chart in figure 6.33, you can see the amplitudes of the resulting deformable field along the x-direction of the original image. The x-axis of the plot corresponds to the x-direction in the image whereas the y-axis contains the intensity value of the deformable field at that position. It can be seen that three voxel contain a value of almost exactly 2.0, as expected, whereas in one voxel there is a small overshoot.

However if you apply this resulting deformable field to the larger image, you get an almost perfect reconstruction of the smaller image (see figure 6.34): The calculated error by the site-specific filter is just $32.26\mu m$ which is about 1.6% of the original distance to be registered.

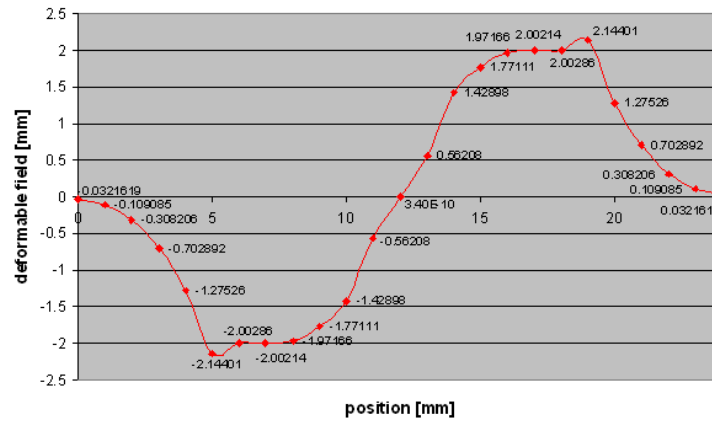


Figure 6.33: This chart shows the amplitudes of the deformable field shown in figure 6.32.

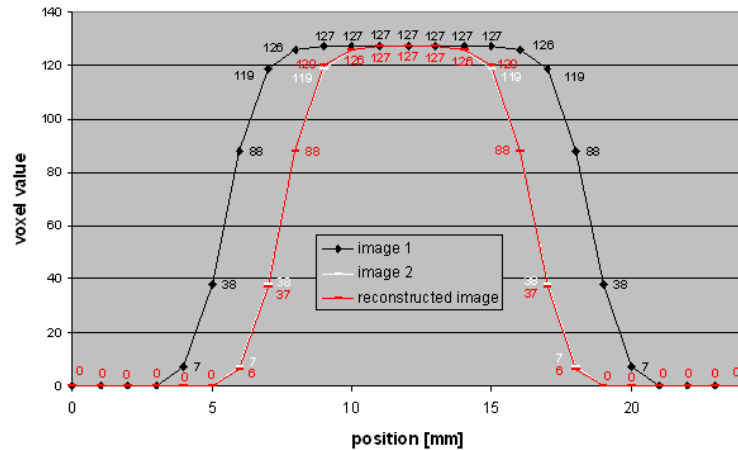


Figure 6.34: In this chart the image values in each voxel along the x-direction can be seen. The red line was calculated by applying the deformable field to the black one. It should have exactly the same values as the white line which is achieved almost perfectly.

6.4.11 FEM Registration

As already mentioned in section 6.3, this algorithm works not as good as the demons registration. In figure 6.35, the registration error calculated with the SiteSpecificChangeFilter is shown for both methods. As the same error is represented by the same color in both images, it can easily be seen that the demons algorithm has by far the smaller error. Also by looking at the histogram of the error in figure 6.36, the difference is enormous. This difference can also be seen in the resulting statistical values:

	FEM registration	demons registration
Mean error:	$-5\mu m$	$-1\mu m$
Standard deviation:	$16.7\mu m$	$9\mu m$
Median error:	$-1\mu m$	$< 1\mu m$

In the following, the used parameters are listed:

number of levels	single-resolution
Number of pixels per element	4
Elasticity (E)	1.e4
Density x capacity (RhoC)	9.e3
Image energy scaling (gamma)	1.0
NumberOfIntegrationPoints	4
WidthOfMetricRegion	1
MaximumIterations	20
metric	mean square
Alpha	1.0
DescentDirection	0
DoLineSearch	0
TimeStep	1.0
Landmark variance	0.5
Employ regriding / enforce diffeomorphism	0
Cross-sectional area (A)	1.0
Thickness (h)	1.0
Moment of inertia (I)	1.0
Poisson's ratio (nu)	0.0
Density (RhoC)	1.0

Of course, it might be possible to find parameters which would lead to a better result. However, we tried quite a bit around with these values and it was the best result we could achieve. If there is a better result, it is quite questionable if it is worth spending that much time to find it.

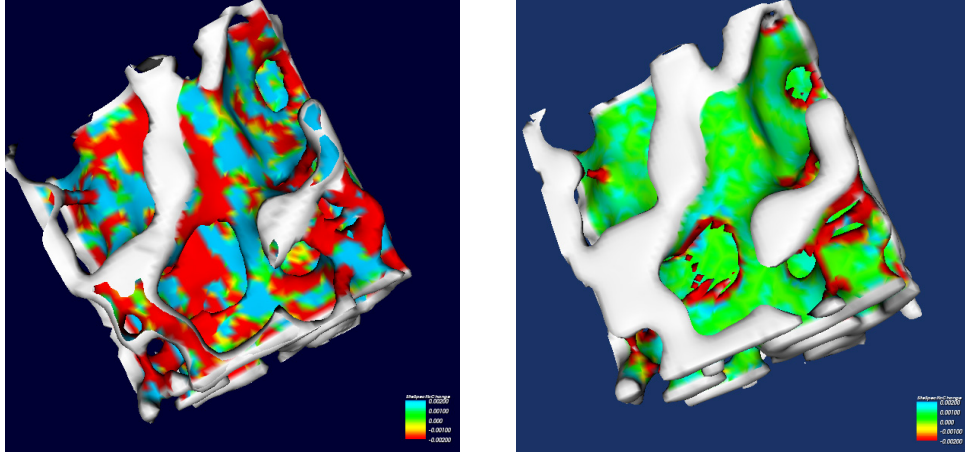


Figure 6.35: Comparison of the FEM (left) and the demons (right) registration. The error has been calculated with the SiteSpecificChangeFilter (images: dog03_gauss_sub.aim and dog05_gauss_sub.aim).

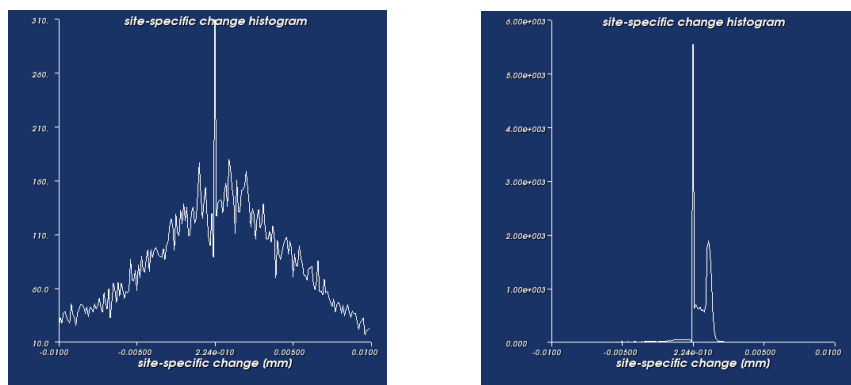


Figure 6.36: Histogram of the mean errors shown in figure 6.35.

6.5 Discussion

In the overall, the demons deformable registration works very well. Wherever there are smooth and uniform changes, the registration error is just a few micrometers. Compared to a image resolution of $15\mu m$ to $30\mu m$, this error is very small. However, this method has a problem where a lot of bone is resorbed, for example where a trabecula disappears.

In general, it can be said that the registration works better with raw CT intensity images than with thresholded "black-white" images! This is due to thresholded images having a huge gradient just exactly on the border bone-background. All other positions have an image gradient of zero which results in no shift at all. However, as it is not only the border which has to be shifted, the result of the registration gets very poor for thresholded data.

For a single-resolution registration, it is recommended to use a standard deviation between 1.0 and 1.5 and to perform at least 10 iterations. It is safer to use a bigger standard deviation (1.5) as then the result never gets wrong. However, it is possible that a better result could be achieved with the smaller standard deviation (1.0).

For a multi-resolution registration with two levels, it is much more difficult to give recommendations as the optimal result varies much more than with just one image resolution. However, for the number of iterations at the lower resolution and the standard deviation, a clear favorite has turned out to be 2 for the number of iterations and 1.5 for the standard deviation. For the number of iterations at the higher resolution, it is very difficult to give recommendations as dependant on the parameter, another number is optimal. Sometimes, it seems to converge already after two iterations. But it can also happen that even after 30 iterations, the convergence is not reached. In general, it might be said that after 5 to 7 iterations, the same result is reached as with 10 iterations and one single resolution. With this recommendation, about 30% to 40% of the computation time can be saved.

However, it is important to realize that it is not useful to have as many pyramid levels as possible. If the resolution is too low at the lowest level, the initial deformation field for the next higher level can get completely wrong. Often, this cannot be corrected at the higher levels of the pyramid and at the end, you get an even worse result than with less levels.

If time is not an issue, it is recommended to just use one resolution as it is easier to get good results. Another reason is that the multi-resolution only brings an advantage if just a few iterations are performed (for many iterations, it converges with the single resolution result). This requires a careful fine tuning which might be time consuming.

To be able to test how many iterations are optimal, an application has been implemented (AppAutomatedTestDeformableRegistration) which calculates for different combinations (number of iterations and standard deviation) some statistical parameters of the registration error and writes them into a text-file. These values can be imported into excel and visualized (a template can be found on the CD to this project in the folder templates\templateToFindBestDemonsParams.xls). It is recommended

to mainly consider the standard deviation of the mean error and the metric value for making a choice.

Another opinion about the choice of the number of iterations is the following:

”In practice, any run of this filter, that uses all the allowed iterations, should be considered as ‘non-converging’, since the limit of number of iterations is there only for preventing the filter from running forever. Users should check at the end of the run if the filter stopped due to a convergence of the changes being under the MaxRMSError value or because it used all the iterations.”

In the filter we implemented, this MaxRMSError can be set as well. However, we think that it is not so easy to make a good choice for that value. If it is chosen too big, the registration stops too early and you do not get a good result. However, if this parameter is chosen too small, the result is considered as non-converging although it might be good. A good choice for this parameter might be 0.01mm, however, it has not been tested thoroughly.

If too few iterations are used, the operator will not automatically notice this. However, a quick check can be done by looking at the statistical error parameters provided by the application. Typical values seem to be the following:

mean error	around $2\mu m$ to $3\mu m$
standard deviation	around $10\mu m$
median error	around 0
metric	varies
RMS change	less than $10\mu m$

In the future, it might be worth to give the FEM method another try. At the time of writing this report, itk was a new toolkit with some errors and potential for improvements. It is possible that at some point, this filter (itkFEMRegistrationFilter) works as good as the demons deformable registration.

There might even other deformable registrations which result in a better registration at large bone changes.

Part II

Generalization

Chapter 7

Introduction

The result of a site-specific change measurement are vectors attached to every point on the surface under consideration (see 5). If the volume of interest is small the easiest way is to look onto the visualization of the site-specific change (see B.1.2). However, if the data is huge a generalization is needed to show what is happening in the bone. Generalizing data has always the goal to simplify a result as much that it is on the first sight clear what is happening. Yet, you have to be careful that no important information gets lost during this process of generalization. The following chapters show different possibilities to simplify the site-specific change.

Chapter 8

Basic measurements

With basic measurement we mean single numbers or vectors that can already give a first impression of the overall site-specific change. These measurements were already mentioned in chapter 5. Table 8.1 summarizes these measurements and gives the references.

Name	Meaning	Reference
NumberOfGrowthPoints NumberOfLossPoints	what percentage of the bone experiences loss/ apposition	5.4.2.1
MeanOfAllSiteSpecificChangeVectors	shows a translation	5.4.2.2
MeanOfLossGrowth	how much bone loss/ apposition is present	5.4.2.2
StandardDeviationOfLossGrowth	is there a big variation of loss and apposition	5.4.2.2
MedianOfLossGrowth	same idea as mean (is not bothered by few wrong numbers)	5.4.2.2

Table 8.1: summary of basic measurements

Chapter 9

Relative site-specific change

9.1 Concept

We define the relative site-specific change as the quotient of the site-specific change and the thickness in every point of the surface. Figure 9.1 shows a sketch which contains at the top a lossy bone that loses on the right side more material than on the left side. The red arrow represents the site-specific change and the green one the thickness. The relative change is defined as division of these two values. On the bottom the histogram of this scene is shown. The thickness is calculated using the thickness measurement described in chapter 12. Be aware that the relative change is not the same using the normals or the closest points method measuring the site-specific change. We recommend to use consistently the closest point method because in the case of resorbed trabecula we got better results.

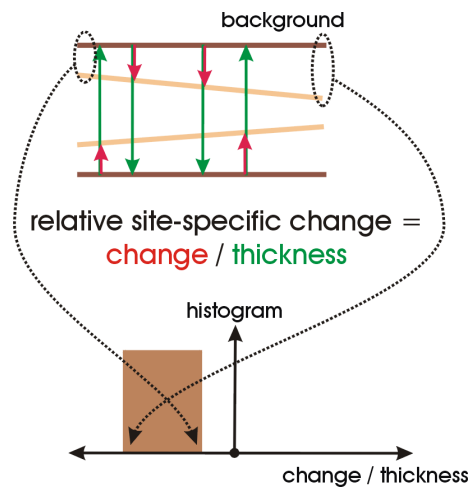


Figure 9.1: This sketch shows at the top a lossy bone that loses on the right side more material than on the left side. The red arrow represents the site-specific change and the green one the thickness. The relative change is defined as division of these two values. On the bottom the histogram of this scene is shown.

There are two possibilities to visualize the relative site-specific change. You can either use a histogram to show the relative changes or use the surface reconstruction and color the surface according to the relative changes. The histogram has the advantage that it is also useful for big data sets.

The implementation of this approach uses the site-specific measurement (see B.1), the thickness measurement (see B.2) and a filter for the division of the two values described in B.5. How these elements are combined to a pipeline is shown in figure 9.2.

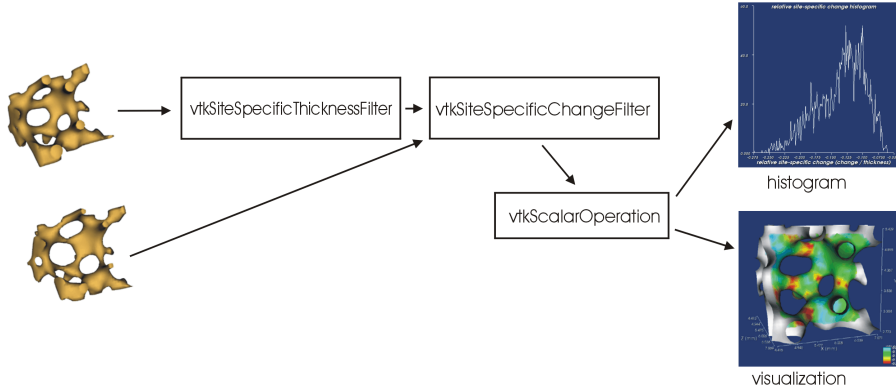


Figure 9.2: this figure shows the pipeline using the thickness and the site-specific change measurements

9.2 Discussion

Figure 9.3 shows a sketch of a relative site-specific change measurement. If a thin structure loses the same amount of bone as a thick one, the relative change will be higher for the thin structure as you can see in the two histograms in the figure. Therefore you get additional information of the microstructural adaptation of the bone. An important point of this measurement could be that it is possible to draw conclusions from this measurement to the mechanical stability of the bone. If two bones lose the same amount of bone looking at the mean of the site-specific change, the one which shows more large relative changes is probably less strong (compare to figure 9.3).

Figure 9.4 shows an example of a relative change measurement. On the left side the visualization of the relative site-specific change is shown. To bone loss was achieved by using SIBA [10]. Therefore the site-specific change is everywhere the same. As a result in the middle of the trabeculas where the bone is thinner the relative change is bigger. On the right hand side of this figure the histogram of the relative site-specific change is shown. In terms of generalization it is very useful because it can also be used for huge datasets.

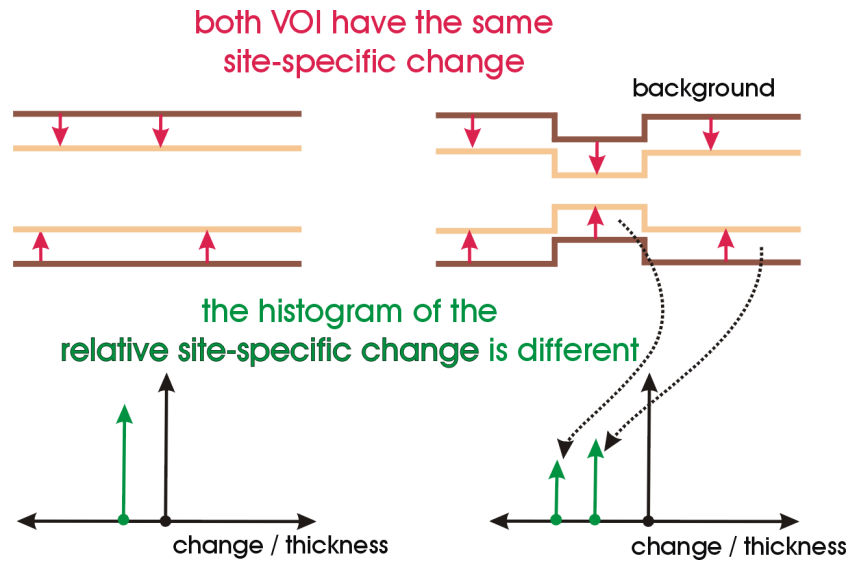


Figure 9.3: The relative site-specific change gives additional information about the microstructural adaptation of the bone. The sketch shows that just the site-specific change gives the same result for both structures. In contrast to that the relative change is different.

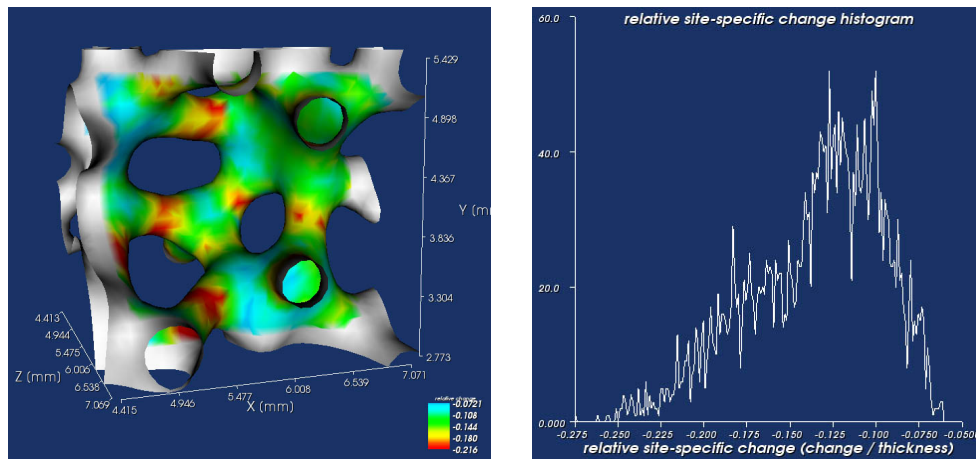


Figure 9.4: on the left hand side the surface is colored according to the relative site-specific change and on the right hand side you can see the histogram of the same VOI (siba1_gfilt_sub_smaller.aim, siba3_gfilt_sub_smaller.aim)

Chapter 10

Principal components of site-specific change

10.1 Concept

After a site-specific change measurement you get scalars and vectors. The vectors represent the direction of the change in every point and the scalars the length of the vector. When all the vectors with the according length are moved to the origin you get an impression of the changes over the whole bone (see discussion in 10.4). An example of such a point distribution is given in figure 10.1 where the points are the end points of the vectors.

To simplify these points to one object first an approach of fitting an ellipsoid to these points was used (see section 10.2) and afterwards a principal component analysis (PCA)(section 10.3).

The implementation of this approach is described in B.3.

10.2 Fitting of an ellipsoid

In [11] is proposed to use a second rank tensor to describe the anisotropy of a material. This leads to an ellipsoid which describes the anisotropy of this material.

We tried to use the same approach with the site-specific vectors. The equation for an ellipsoid 10.1 was brought to the form of a least squares problem $Ax = b$ and solved in order to find the values of the tensor 10.2.

$$Ax_1^2 + Bx_2^2 + Cx_3^2 + 2Dx_1x_2 + 2Ex_1x_3 + 2Fx_2x_3 = 1 \quad (10.1)$$

$$[M] = \begin{bmatrix} A & D & E \\ D & B & F \\ E & F & C \end{bmatrix} \quad (10.2)$$

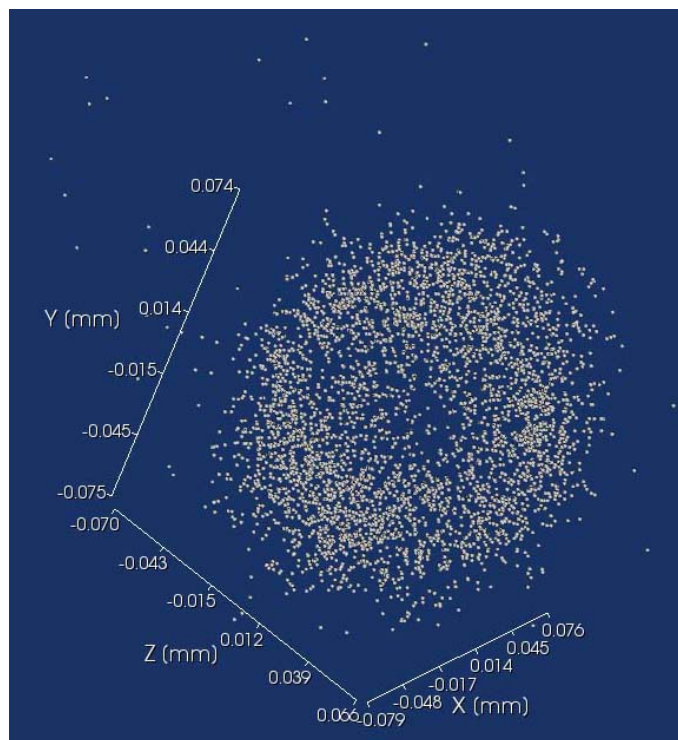


Figure 10.1: visualization of the site-specific vectors moved to the origin as points

The eigenvectors of the matrix M 10.2 are directly the axes of the ellipsoid and the eigenvalues are the lengths of them.

The issue was that the least squares problem didn't have always a solution and therefore a second approach was used in the end.

10.3 PCA

A principal component analysis returns directions for which the variance is maximal (the first axes point into the direction of the maximal variance). The directions are orthogonal to each other. There are a couple of important points that should be considered.

- If the data set you make the PCA has got a mean that has a real meaning (in this case for example the mean vector of all site-specific changes which could show a translation of the samples) it makes sense to compensate this mean before performing the PCA. Otherwise the first principal component will point to the mean of the data and this information is already available without the PCA. Therefore in our case the points were translated by the mean of all vectors.
- The Analysis returns the same number of principal axes as you define the vector space for the analysis itself. In our case this dimension is three.

- When the data for which a PCA is done has different scales it is important to scale the data prior to the analysis otherwise the results are hardly interpretable. In our case that wasn't a problem because all the three axes have the same scales.
- A singular value decomposition (SVD) can be used to calculate the principal components and the variance for each of the PC axes. If the decomposition is given by $A = USV^T$ the principal axes stand in the columns of V (are also the eigenvectors of A). The first column represents the first PC and so on. The matrix that was actually decomposed is given in 10.3.

$$[A] = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ \dots & \dots & \dots \end{bmatrix} \quad (10.3)$$

- The variance for each PC axes was calculated by first applying the inverse of the transformation matrix 10.4 (where M is the mean vector, and V^T the eigenvectors) to the points in order to transform the PC axes onto the x, y and z axes and to compensate the mean. Afterwards the variance for each of the PC directions was calculated by computing the variance in x,y and z direction of the transformed points.

$$[T] = \begin{bmatrix} V^T & M \\ 0 & 1 \end{bmatrix} \quad (10.4)$$

Finally the length of the axes of the ellipsoid are represented by the variance of the points in this direction and the direction by the principal components. Actually we chose two times the variance as length because it represents the points in a better way as can be seen in figure 10.2.

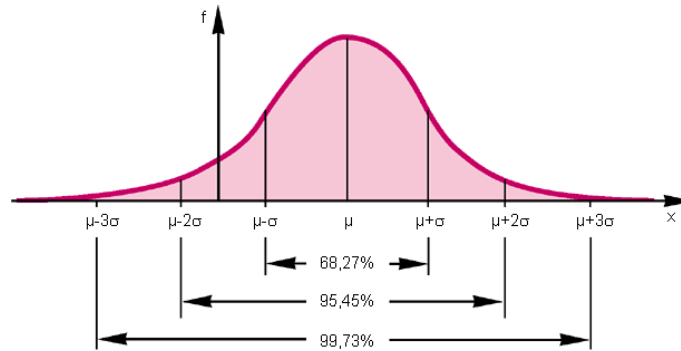


Figure 10.2: normal distribution in order to show why two times the variance was used as length for the axes of the ellipsoid

Figure 10.3 shows an example of an ellipsoid which was calculated by the principal component analysis in the described way.

Figure 10.4 summarizes the steps of how the PCA is applied to the set of points.

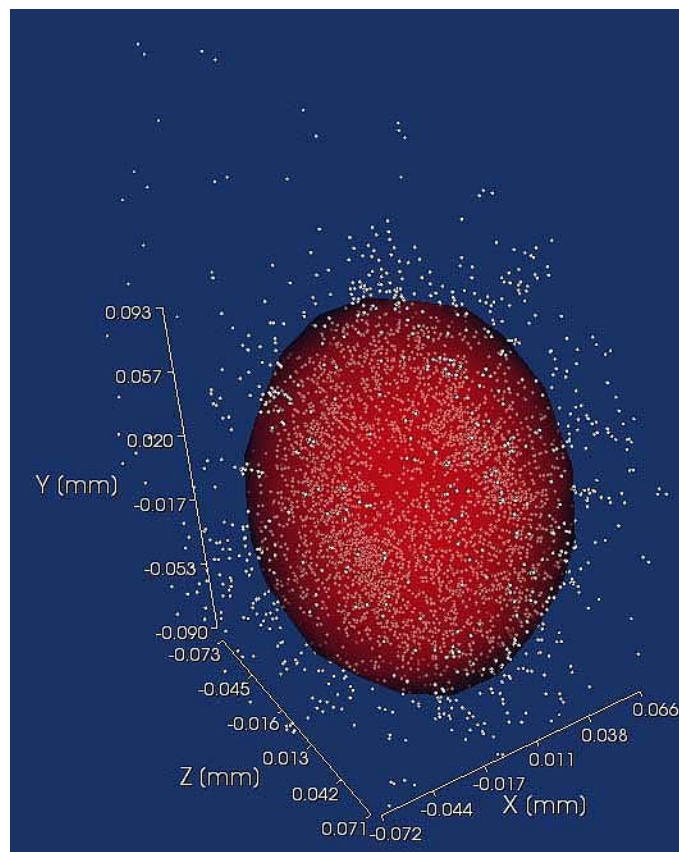


Figure 10.3: this figure shows the resulting ellipsoid of the PCA analysis

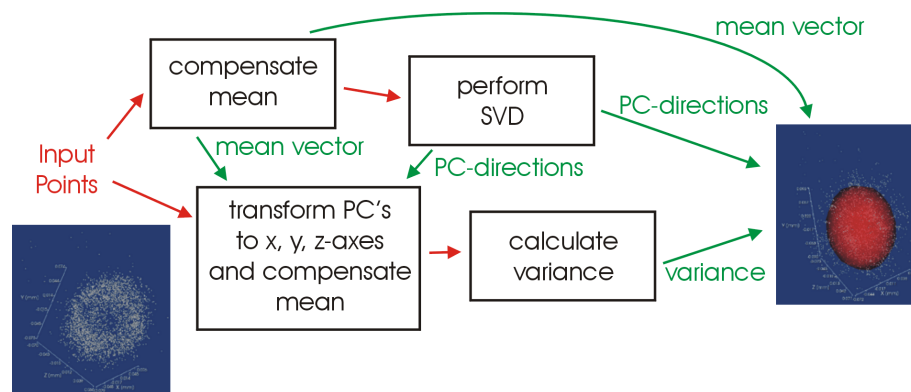


Figure 10.4: this sketch shows how the ellipsoid is calculated out of a set of points

Separate positive and negative changes

It is to anticipate that real data will have bone resorption and apposition in the same volume under consideration. When the vectors are moved to the origin the information if it was apposition or loss gets lost and the ellipsoid would contain both of them. Therefore the positive and negative vectors are separated from each other

in advance. This procedure is shown in figure 10.5.



Figure 10.5: shows the pipeline of the filter and how the negative and positive changes are separated

A possible layout is shown in figure 10.6 where positive changes (apposition) are represented by the green ellipsoid and the negative ones (loss) by the red ellipsoid. It is important to assign the same camera to both renders that you always have the same view of both ellipsoids and also the same zoom scale.

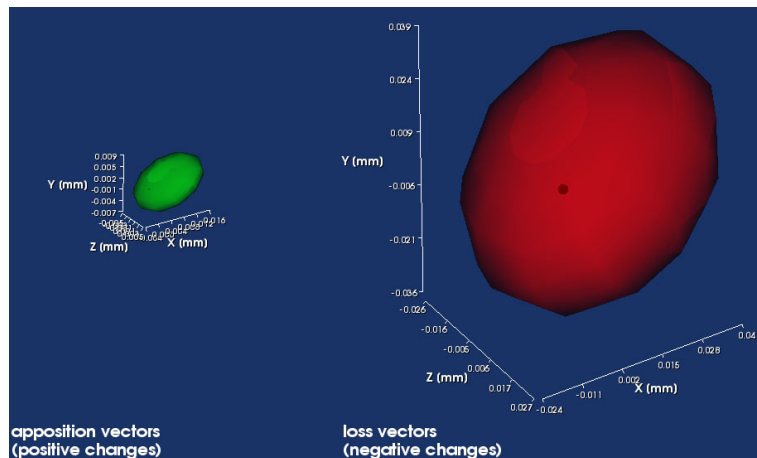


Figure 10.6: Positive (apposition, green) and negative (loss, red) site-specific changes are separated prior to the calculation of the principal components (siba1_gfilt_sub_smaller.aim, siba3_gfilt_sub_smaller.aim)

10.4 Discussion

The subsections in this chapter show the different interpretations that could be discovered using the ellipsoid. The following points can be identified.

- Direction of the site-specific changes (subsection "Different gaussian smooths")
- Amount of resorbtion and apposition (subsection "Size of the two ellipsoids")
- Translation of the hole data set (subsection "Translation")

10.4.1 Different gaussian smooths

This example shows the change in shape of the ellipsoid if the site-specific change is not the same in all directions. Figure 10.7 visualizes on the left side the used data set. From the middle to the right picture the gaussian smooth was changed in a way that the change in x-direction got smaller. This variation of the site-specific change is visible looking at the shape of the ellipsoid. Using this change of shape it could be possible to draw a conclusion to the change of strength of the bone in a certain direction.

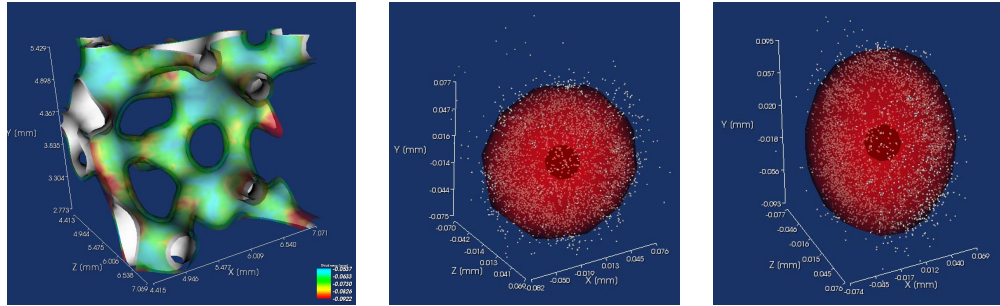


Figure 10.7: The left image shows the volume of interest. From the middle to the right picture the gaussian smooth was changed in a way that the change in x-direction get smaller. This variation of the site-specific change is visible looking at the shape of the ellipsoid (siba1_gfilt_sub_smaller.aim, siba3_gfilt_sub_smaller.aim)

Be aware that the structure also has an impact on the ellipsoid. If there are for example plains in the bone oriented in x-y-direction you will have more vectors in z direction. The fact that there are more vectors in one direction skews the ellipsoid in this direction. Therefore it is important to know that the shape of the ellipsoid is a function of the *size of the site-specific vectors* but also the *number of site-specific vectors* in one direction. A idea of how to separate the two factors is given in the outlook 16.

10.4.2 Size

Considering for example figure 10.6, it can be seen that the loss ellipsoid is bigger than the apposition ellipsoid. The reason for that is that it was a dataset with big bone loss and just very small bone apposition. Therefore the losses are bigger than the appositions in the volume of interest.

10.4.3 Translation

If the data sets where the site-specific change was measured are translated to each other this affects the position of the ellipsoid. Figure 10.8 shows an example of such a translation. The left picture visualizes the actual translation by 0.2 mm in x-direction.

The right picture shows the effect on the ellipsoid of this translation. You can see that the center of the ellipsoid, which always represents the mean vector, is shifted in x-direction. The origin is marked as black sphere.

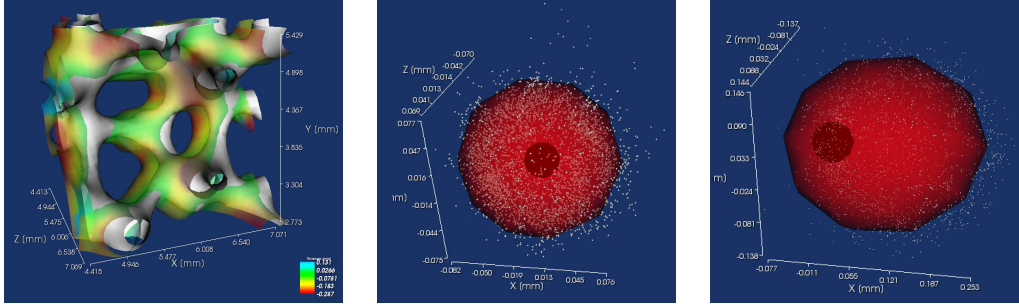


Figure 10.8: The left image shows the actual translation of 0.2 mm. The image in the middle shows the ellipsoid without translation and the picture on the right with translation. You can see that the center of the ellipsoid is shifted compared to the origin (black sphere) (siba1_gfilt_sub_smaller.aim, siba3_gfilt_sub_smaller.aim)

Chapter 11

Identify resorbed trabeculae

It is known that during bone loss, trabeculas get resorbed. Using the in-vivo data we could support this statement (see for example figure 5.28). In terms of generalization it would be very interesting to extract the number of trabeculae which are resorbed in a certain volume and also the position and direction of the resorbed trabeculae. There exists a connectivity measurement [15], but because it is just a statistical number no position or direction can be extracted. We came up with a first solution for this problem. The following section describes the algorithm that has been implemented. The appendix contains under B.6 some information about the implemented filter.

11.1 Procedure

The following items show the steps which were used to identify the resorbed trabeculas and also new holes in the bone. Be aware that we are looking again at two time points and consequently we have two surfaces (again, surface no. 1 is earlier in time and surface no. 2 later in time).

- As preprocessing a deformable registration was made using the demons algorithm which warps surface no. 2 onto surface no. 1. The asymmetric demons was used because it only uses the gradient of surface no. 1 and therefore doesn't rebuild the resorbed trabeculas but it corrects errors of a bad registration or macroscopic movements of the trabeculas.
- In this step points in the middle of surface no. 1 are searched. This is done by blowing up spheres inside the bone and store the center of the sphere.
- Afterwards points that lie inside surface no. 2 are deleted from the set of points (see figure 11.1). In this stage we already have groups of points that represent the resorbed trabeculas and new holes but also some wrong points.
- One place where wrong points are found is when a trabecula gets very thin but isn't resorbed completely. To exclude these points the angle between two vectors is considered. One vector is from the point under consideration to surface no.

1 and the other vector from the point to surface no. 2. In the case of a fully resorbed trabecula or a new hole the angle is around 90 degrees. Yet in the case of a not fully resorbed trabecula the angle is around 180 degrees (see figure 11.1).

- The remaining groups of points represent the resorbed trabeculas and new holes and have only to be identified as group. This is done by laying imagedata over the polydata and set the voxel value to non-zero if one of the points lies inside this voxel. The spacing of the imagedata is chosen proportional to the thickness (e.g. 1/4 of the thickness).
- To make sure that there are no gaps in each of the groups a dilation is made on the imagedata.
- Thereafter the identified connected regions are added up. This number includes resorbed trabeculas and new holes.

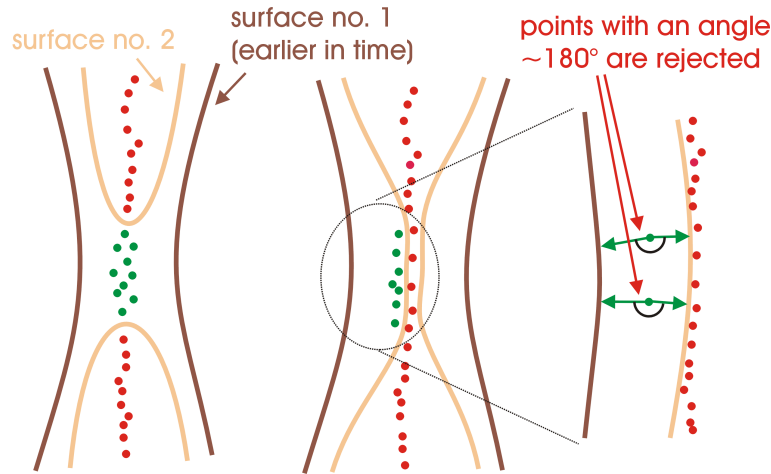


Figure 11.1: left: a resorbed trabecula where red points are rejected because they are inside surface no. 2; middle and right: the green points should be excluded because the trabecula does not fall away but the points do not lie inside surface no.2 therefore the angle is used to exclude these points

11.2 Discussion

Figure 11.2 shows three pictures of two different bones. In all pictures the scan earlier in time is represented by a transparent white surface and the one later in time by a white opaque surface. The top left one is an example of aluminum foam where the implemented filter was tested. The colored blobs show the points where the algorithm has found a resorbed trabecula. In this example the algorithm works perfect. Also in the other two images (canine bone) the algorithm has no problems to identify the absorbed trabeculas. Be aware that for both examples the standard settings of the

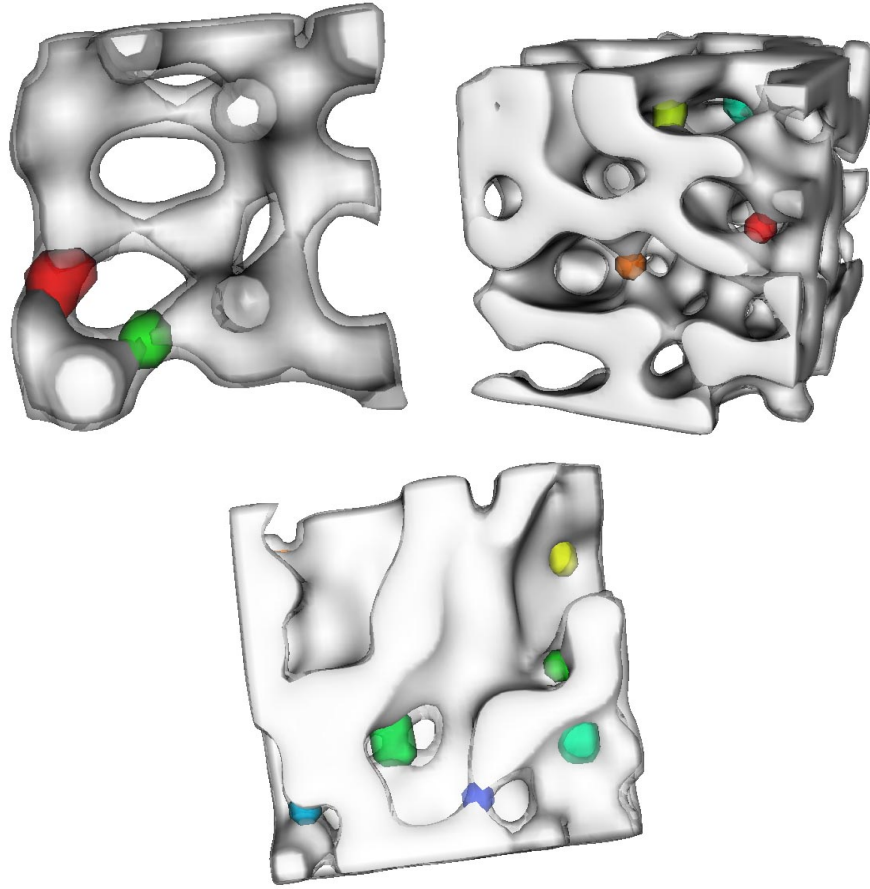


Figure 11.2:

implemented filter were used, although the size of aluminum foam and canine bone is very different.

Yet we found other examples where the algorithm has found too many resorbed trabeculas and new holes. The problematic areas were always around "donut shaped" holes which increase the radius. Yet we can say that if a trabecula is resorbed it is identified by the algorithm, at least in the examples we were looking at.

The algorithm was also applied to the in vivo data that was at our disposal. Because there were only four time points the bone losses were huge sometimes. If a very big structure is resorbed the algorithm will identify the whole region as just one trabecula. Therefore it is important to have more than just four time points when applying this algorithm.

Using the inputs to the filter the other way around, it should be possible to identify forming trabeculas or closing holes as well.

Chapter 12

Thickness measurement

12.1 Introduction

An interesting parameter of trabecular structures is the thickness. We looked into it because we wanted to compare the site-specific change with the thickness as a part of the generalization (see section 9).

At the moment the mostly used thickness measurement was introduced by Tor Hildebrand and Peter Rüggsegger in 1996 [7]. It fits maximal spheres into the bone to calculate a volume weighted mean. The main advantages over the measurements which used the mean intercept length (MIL) (Whitehouse, 1974 [8]) that had been used before is that it is really working on 3D data and that it is independent of the type of the structure (rod-like, plate-like).

For our application (see 9) we needed an implementation which operates on **poly data** (surface data) because a thickness value is needed in every point on the surface whereas the method described in paper [7] just works with **image data**. We call this measurement *surface based thickness* or *site-specific thickness*. With the same method it is also possible to measure the trabecular separation.

In the following sections we start with the basic idea for an implementation which works directly on surface based data. Thereafter we show in the evaluation section that the implementation works with various data sets.

12.2 Concept of the surface based approach

The very basic idea is to traverse every point on the surface and to blow up a sphere until this sphere intersects with the surface in another point (see sketch in figure 12.1). The center of the sphere lies on the negative normal of the surface.

To check if the sphere intersects with a point on the opposite side or with a point just next to the starting point (due to a rough surface) the angle between the two normal vectors is calculated. This is done with the dot product. If the orientation is wrong

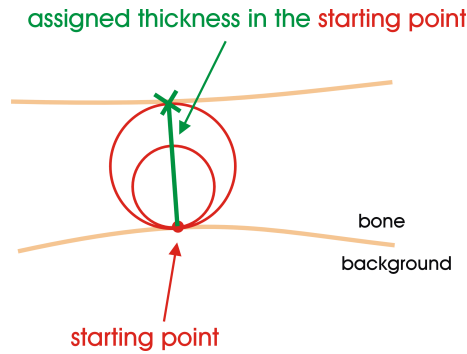


Figure 12.1: basic idea of surface based thickness measurement (blow up a sphere until it intersects with the opposite side)

you have to search for the next point. The angle of rejection was set to 60 degrees (see figure 12.2).

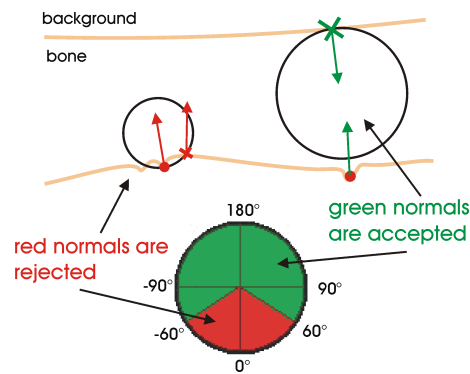


Figure 12.2: orientation checking (rejection if angle between normals smaller than 60°)

Looking on the left side in figure 12.3, this approach would not work so far because the sphere intersects with points on the same side of the bone if the surface is rough. Therefore there is a variable offset used to inhibit this intersections (on the right side in figure 12.3). If a point with wrong surface orientation is found it is possible to increase the offset to avert such intersections. The resulting thickness can be corrected by adding the offset to the sphere radius diameter.

To measure the trabecular separation instead of the thickness, the center of the sphere must lie on the positive surface normal instead of the negative one (i.e. inverting the normals).

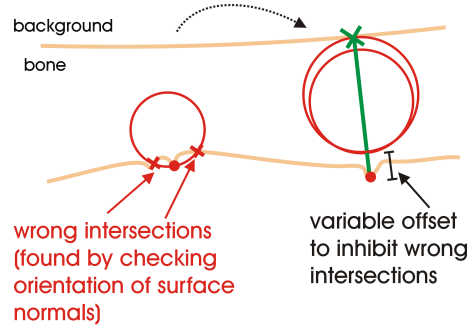


Figure 12.3: variable offset of sphere to inhibit wrong intersections

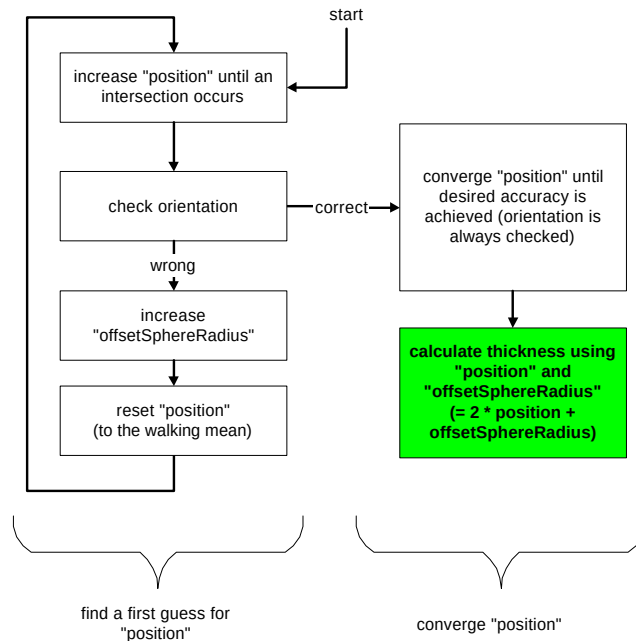


Figure 12.4: basic flow chart of the algorithm

12.3 Algorithm

Figure 12.4 shows the flow chart of the algorithm. First a correct intersection is searched by increasing *position*. The starting radius is chosen as the mean of the already calculated. Figure 12.5 shows what is meant by *position* and *offsetSphereRadius*. The difference between these two values always leads to the sphere radius. After finding this first guess, the accurate thickness is searched by always taking half of the present bin. Figure 12.6 shows an example of how *position* converges to a certain accuracy (the numbers indicate the advance of *position*). The idea is that if the sphere still intersects, the position is decreased and if the sphere does not intersect anymore the position is increased until the desired accuracy is reached (see B.2.1 for the accuracy).

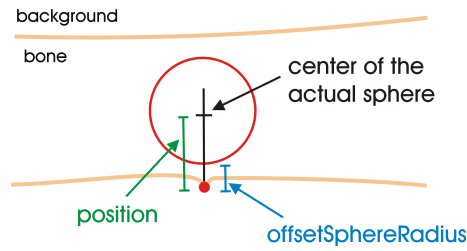


Figure 12.5: explanation of position and offsetSphereRadius

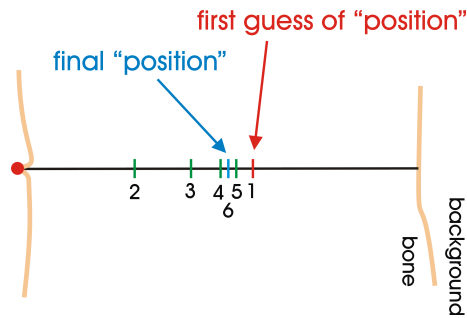


Figure 12.6: find final value of position by always taking the half

The resulting thickness can be calculated by the formula: $thickness = 2 * position - offset$.

12.4 Weighting of the mean

This section is divided into two parts. The first one discusses why the mean of all thickness values is directly surface weighted and the second part shows two possibilities of unweighting the mean.

12.4.1 Surface weighted mean

If a mean thickness (Tb.Th.) value over the hole sample is wanted to be calculated, a weighting of the individual thickness values has to be performed. As mentioned above the thickness is calculated in every point on the surface. By using only a mean of these values you get a surface weighted mean value. This mean has the problem that thicker structures are weighted more than thinner structure but not as strong as a volume weighted approach.

To build the surfaces out of the image data the algorithm marching cubes [9] was used. The points which represent the surface are distributed almost uniformly over the surface. Therefore it should be possible to only calculate the mean of the computed thickness values.

Name of Data	mean of point-thicknesses	surface weighted mean
siba3_gfilt_sub_smaller.aim	0.512809 mm	0.515128 mm
dog01_gauss_sub.aim	0.219333 mm	0.220973 mm
siba3_gfilt_sub.aim	0.396983 mm	0.397379 mm

Table 12.1: comparison of mean using only the weighted sum over the points to the really surface weighted mean

To check this assumption, the same algorithm was used to traverse the cells (triangles) instead of the points. The advantage of using the cells is that a really surface weighted mean can be performed because the area of every cell can be calculated what is not easily possible for the points. To show that the assumption that the points are distributed uniformly over the surface is applicable, table 12.1 shows the results of three tests using different data sets. You can see that the differences between the surface weighted approach and the attempt to only build the mean over all points lie in the micrometer range.

12.4.2 Unweighting the mean

In contrast to the surface weighted mean the unweighted values are independent from the area of the surface. Two different methods were used in order to unweight the mean. They are presented in the following.

The unweighting is achieved by weighting each of the thickness values (th_i) by the following formula.

$$mean = \sum_i \frac{w_i * th_i}{\sum_j w_j} \quad (12.1)$$

Sphere area

Section 12.2 showed that, in order to calculate the thickness, spheres are fitted into the bone. By using the surface area of the sphere the thicknesses can be unweighted. The weight is $w_i = \frac{1}{\pi th_i^2}$ and leads to the following formula.

$$mean = \sum_i \frac{\frac{th_i}{\pi th_i^2}}{\sum_j \frac{1}{\pi th_j^2}} = \sum_i \frac{\frac{1}{th_i}}{\sum_j \frac{1}{th_j^2}} \quad (12.2)$$

Circumference

The idea of using the circumference is that it is assumed that the number of points on the circumference is proportional to the circumference itself. This method is even better in order to unweight the surface weighted values than the one using the sphere area as you see in the next section. The weight is $w_i = \frac{1}{\pi th_i}$ and leads to the following

formula.

$$mean = \sum_i \frac{\frac{th_i}{\pi th_i}}{\sum_j \frac{1}{\pi th_j}} = \sum_i \frac{1}{\sum_j \frac{1}{th_j}} \quad (12.3)$$

12.5 Example

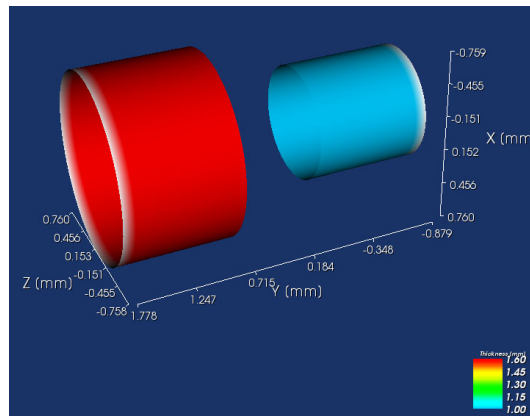


Figure 12.7: these two cylinder were used to show the effect of the unweighting (the diameters are 1 mm and 1.6 mm)

This example uses two cylinder to show the difference between the surface weighted values and the unweighted values. Figure 12.7 shows the scene with the two cylinder. The diameter of the two cylinder are 1 mm and 1.6 mm. Table 12.2 shows the results of the calculation for the different weighting methods and the unweighted values. For the unweighted values a diameter of 1.3 mm is expected. You can see that the surface weighted mean, weights the bigger cylinder stronger because of its larger surface. By unweighting the mean with the circumference the best results were achieved.

surface weighted values	
mean thickness	1.370 mm
standard deviation of thickness	0.292 mm
median thickness	1.601 mm
unweighted mean thicknesses	
using circumference	1.301 mm
using sphere area	1.232 mm

Table 12.2: The results of the thickness measurement using two cylinders. You can see that the unweighting by the circumference leads to the best results.

12.6 Boundary points

We propose that points near the boundary should not be measured and assigned a thickness because the calculation does not make sense in most cases. Particularly in small images, the wrong calculated boundary points have a big impact on the mean measurement. Using big images this errors tend to get smaller.

The width of the boundary can be controlled (see [B.2.3](#)).

The second issue with the boundary is that it is possible that the *position* (center of the sphere) moves out of the boundary on the edges. We do not try to assign a thickness to points where the center of the sphere moves out of the boundary of the picture because in most cases the calculated thickness does not make sense. Therefore such points are omitted.

12.7 Closest Point instead of sphere

Instead of intersecting the sphere with the surface it is also possible to search for the closest point from the center of the sphere and use a bounded radius for the closest point search. We used this approach because it is easier to search for a closest point than to intersect a sphere with a surface. The result is conceptually equivalent to the intersection of a sphere.

12.8 Evaluation

This section covers the evaluation of the thickness measurement. It starts by testing the algorithm with a geometric object. Thereafter the comparison to already existing thickness measurements is outlined. At the end the setting of the accuracy of the measurement will be tested and also the computation time is going to be discussed.

12.8.1 Cylinder with a known diameter

This test shows that the algorithm and the filter work fine for a very simple structure. A cylinder with a known diameter was used (1 mm). [Figure 12.8](#) demonstrates the results. On the left side each point on the cylinder is colored according to the thickness. Comparing to the scalar bar you see that green means one. The same result shows the histogram of the thicknesses on the right side in [figure 12.8](#).

12.8.2 Compare to Tb.Th and Tb.Sp

In this subsection we compare the introduced surface based thickness and separation measurements to Tb.Th and Tb.Sp which are used today. The Tb.Th and the Tb.Sp were calculated with the method introduced by Tor Hildebrand (1996) [\[7\]](#).

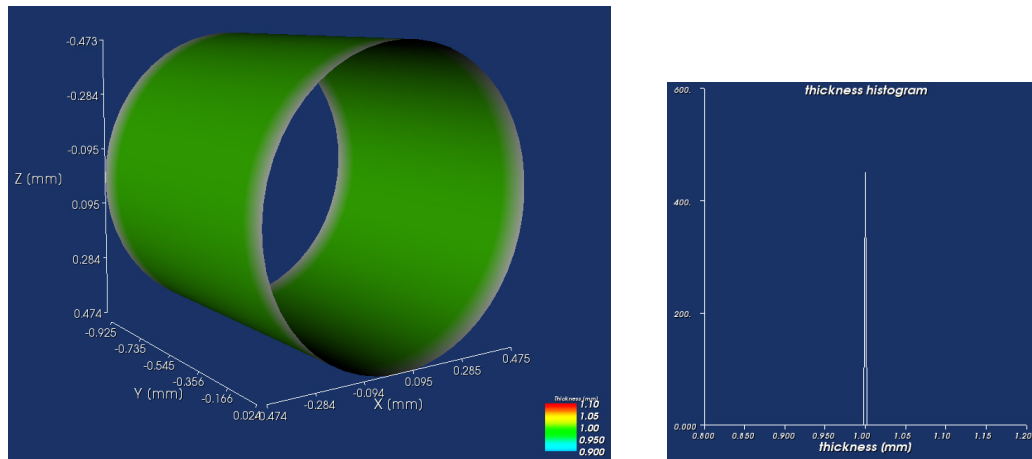


Figure 12.8: measurement of a cylinder with a known diameter (1 mm)

It is always possible to close the bone at the boundaries. It is debatable whether they should be closed or left open for the calculations. Both ways have their advantages. We recommend to leave them open with the introduced method because otherwise the mean of the thickness is underestimated. Certainly it does not matter if the boundary is set big enough and the spheres do not touch the closing surface. Therefore the best way always is to calculate the structural measurement for a boundary that is set big enough. Consequently the boundary does not influence the measurement any more. Be also aware that working with bigger data sets this boundary effect gets smaller.

The threshold of the marching cubes algorithm was set to a value that the volumes of the image data and the volume of the closed surface match each other. The volume of the image data was calculated by counting the voxels of the bone, and multiplying with the voxel volume. The volume of the surface data was calculated using the filter `vtkMassProperties`. The last important thing is to not gaussian filter the data because otherwise the surface will change its position and the thicknesses cannot be compared to each other.

Table 12.3 shows the results of the measurements. The surface based thicknesses have been measured by closing the boundaries because the image based approach has also boundaries (normally we recommend to leave the boundaries open). You can see that the values are always in the same range. The reason for the differences is that the methods are different. The choice of the threshold makes huge differences. The big difference in the `Tb.Sp.` can be explained by looking at the pictures: Mainly for the images shown in figures 12.9 and 12.10 it is not useful to define a separation.

Figures 12.9 to 12.12 show the four pictures used for this test, colored according to the calculated thickness with the spheres method. White color means that no value was calculated at this position (see also B.2.2).

	SAMPLE_D025-01_TOP_THRES.AIM	SAMPLE_D025-02_TOP_THRES.AIM	SAMPLE_D050-01_TOP_THRES.AIM	SAMPLE_D050-02_TOP_THRES.AIM
[mm]				
used threshold (0-127) to match volume	45	48	52	52
Tb.Th	0.119	0.066	0.161	0.109
mean thickness	0.124	0.071	0.141	0.094
unweighted (circumference)	0.115	0.063	0.123	0.077
unweighted (sphere area)	0.105	0.051	0.100	0.056
Tb.Sp	0.166	0.090	0.379	0.564
mean separation	0.372	0.315	0.385	0.415
unweighted (circumference)	0.289	0.188	0.280	0.287
unweighted (sphere area)	0.157	0.118	0.179	0.175

Table 12.3: comparison of mean thickness and mean separation (boundary set to 0.05 mm) (datasets/MorphSamples/)

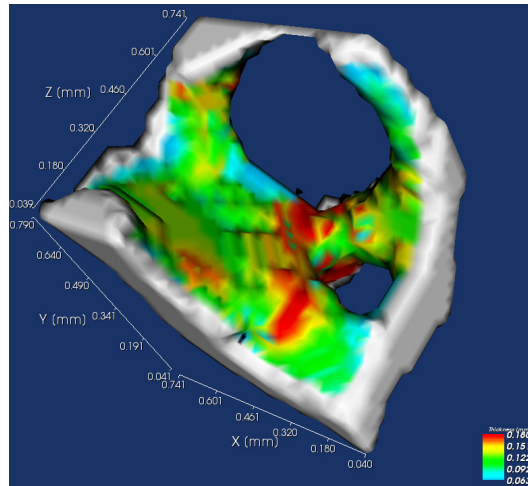


Figure 12.9: SAMPLE_D025-01_TOP_THRES

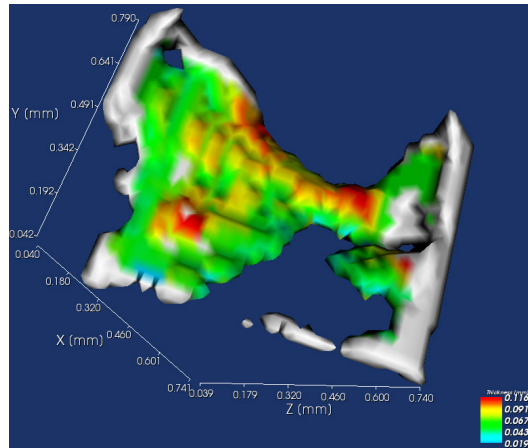


Figure 12.10: SAMPLE_D025-02_TOP_THRES

12.8.3 Change accuracy to speed up the calculation

It is possible to do the calculation of the thickness/ separation incrementally. That means that the filter is started with a big *endLevel* (for example `SetEndLevel(6)`) (compare to [B.2.1](#)) for the first run. Thereafter the *endLevel* is set to five and the vtk-pipeline re-executed. An example is shown in figure [12.13](#) where it was started with a level of six and iterated until level zero was reached. Figure [12.14](#) shows the histograms for each iteration. You can see how the algorithm searches the thicknesses iteratively. It can be seen that the next histogram has more peaks and that they are always in the middle of the last ones. The reason for this is that the iterative search always halves the remaining bins. This search of the final thickness is also shown in figure [12.6](#).

Figure [12.15](#) shows the changing of the mean, the standard deviation and the median

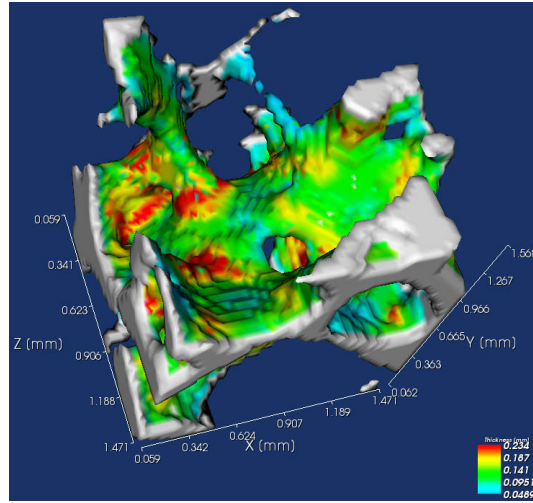


Figure 12.11: SAMPLE_D050-01_TOP_THRES

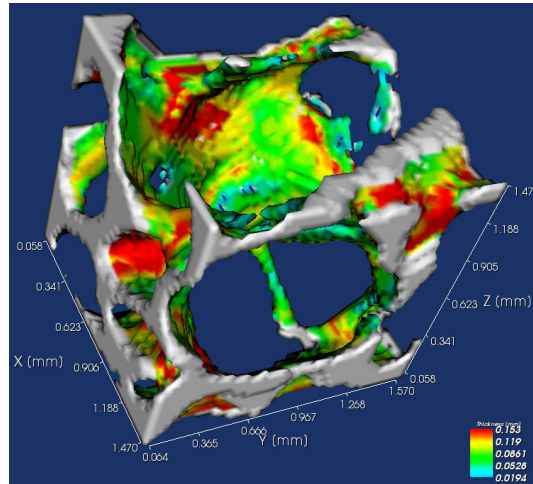


Figure 12.12: SAMPLE_D050-02_TOP_THRES

for different *maxLevels*. You can see that it is not necessary to iterate until level 0 is reached to get a good approximation for these values. If a smaller accuracy is also adequate it is possible to save time by setting the *endLevel* to a bigger value. The time dependency is given in figure 12.16. But be aware that even if the mean converges quite fast (2 to 3 iterations in figure 12.15), this does not mean that the thickness in every point (site-specific thickness) converges with the same speed. The accuracy in a certain point is still given by the formula $(2^{endLevel}[\mu m])$. It is just the statistical values (mean, median) that converge faster.

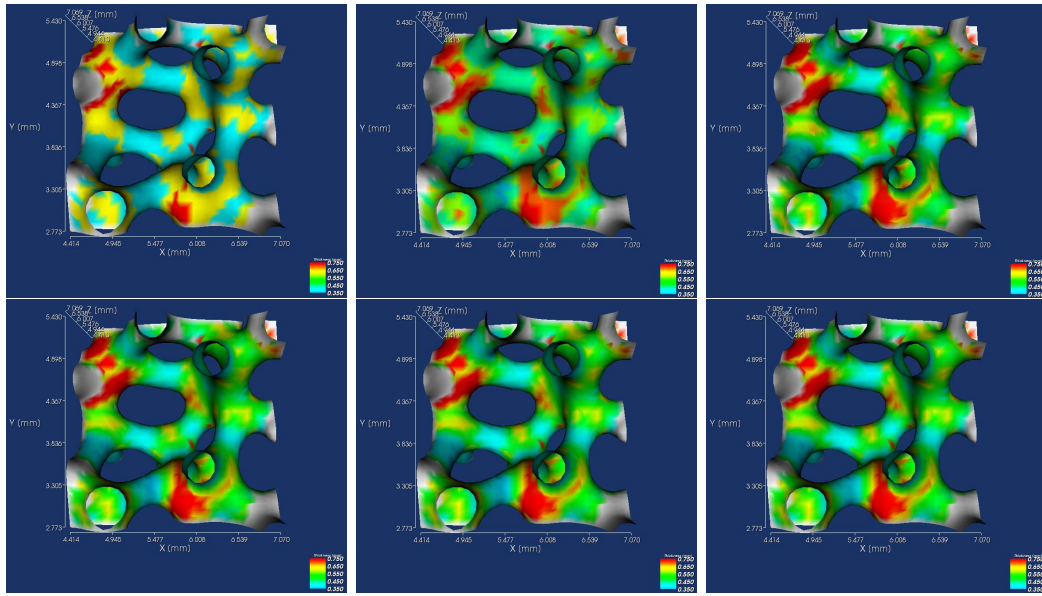


Figure 12.13: measuring thickness using different maxLevels (6,5,4,3,2,0) (siba3_gfilt_sub_smaller.aim)

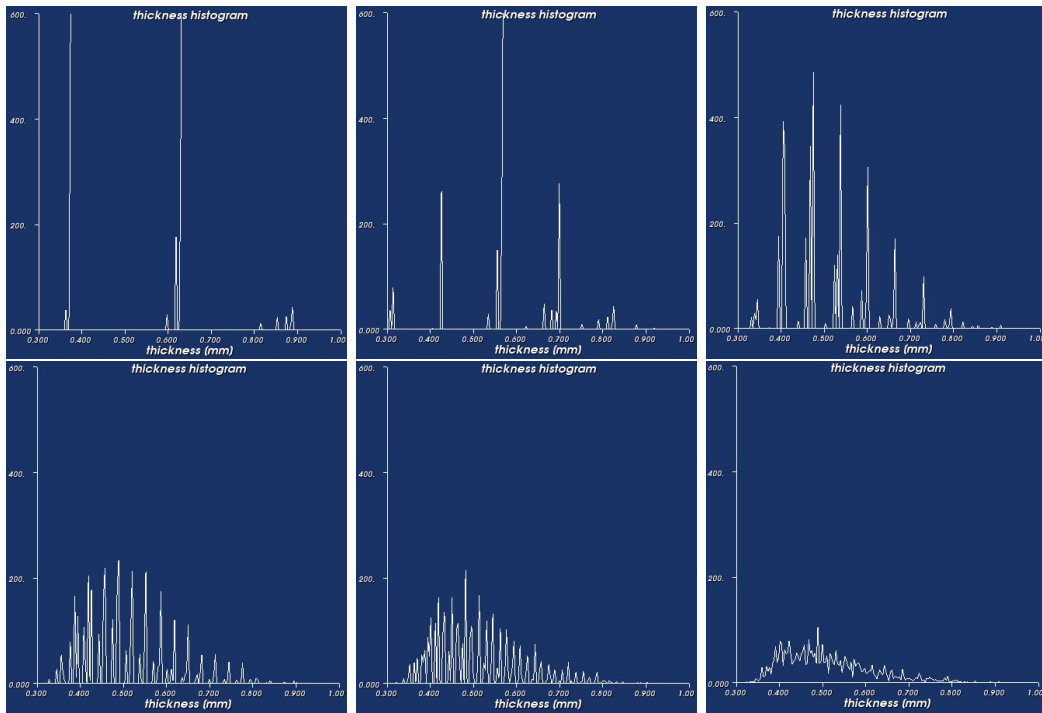


Figure 12.14: measuring thickness using different maxLevels (6,5,4,3,2,0) (siba3_gfilt_sub_smaller.aim)

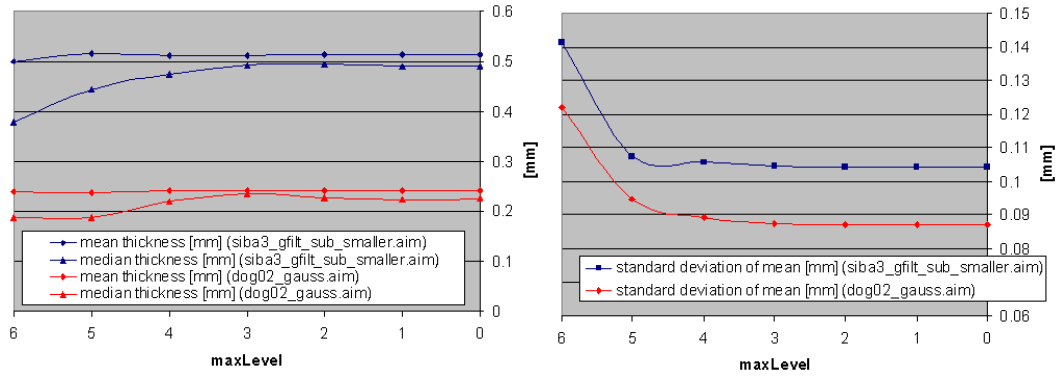


Figure 12.15: statistical values for different maxLevels

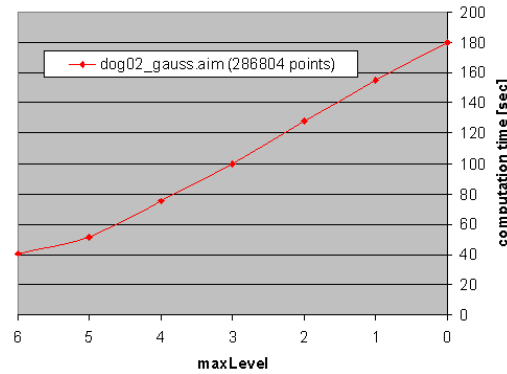


Figure 12.16: computation time for different maxLevels

12.8.4 Computation time

The disadvantage of this method is the slowness. It is computationally very expensive because a sphere has to be fitted into the bone in every point of the surface. The problem of calculation time could be solved by distributed computing because every point is independent of the other ones.

Figure 12.17 shows a diagram of the computation time for the spheres and the surface weighted spheres method of the filter for different number of points. Both methods have the same computation time because the algorithm is exactly the same. The normals method is much faster and isn't plotted in the chart because the thickness is measured very inaccurate with this method (see B.2). The test was running on a Windows XP PC with 2.17GHz and 1GB of RAM.

Be aware that the computation time is not only a function of the number of points but also of the roughness of the surface and the geometry of the sample. This fact can be seen looking at the scatter in figure 12.17.

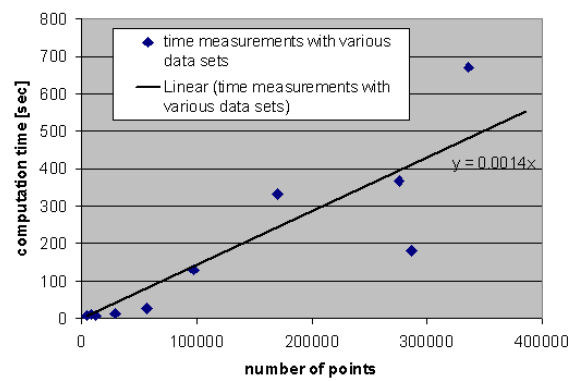


Figure 12.17: computation time for different number of point and different data sets

Part III

Extrapolation

Chapter 13

Extrapolation with Deformable Registration (Prediction of the site-specific bone change)

13.1 Introduction

In order to determine by how much a person is affected by osteoporosis, a density measurement of the bone is done. However, by just looking at the density of the bone, it is assumed that less bone mass directly correlates to a weaker bone. Yet it might be possible that a person with much less bone is not affected by fractures whereas another person with more bone but a very unstable structure is.

In order to find out which persons to treat we need to know not only the density and the structure of the bone but also how it changes in the future. This would permit a new and much more reliable prevention.

The method we implemented is a first approach in order to see if it is possible at all. It is based on the assumption that real bone behaves like simulated bone. This is due to the fact that at the time of implementing this method, there was just simulated data available. However, the principle can easily be extended in order to include future knowledge about how bone adapts.

13.2 Important Considerations

13.2.1 Choice of the Deformable Fields Used to Extrapolate

In principle, there are two main possibilities, which deformable fields to consider when calculating an extrapolation:

- One way of doing it, is to have just one reference image - the first one in time. You would then calculate the deformable field from each image later in time to

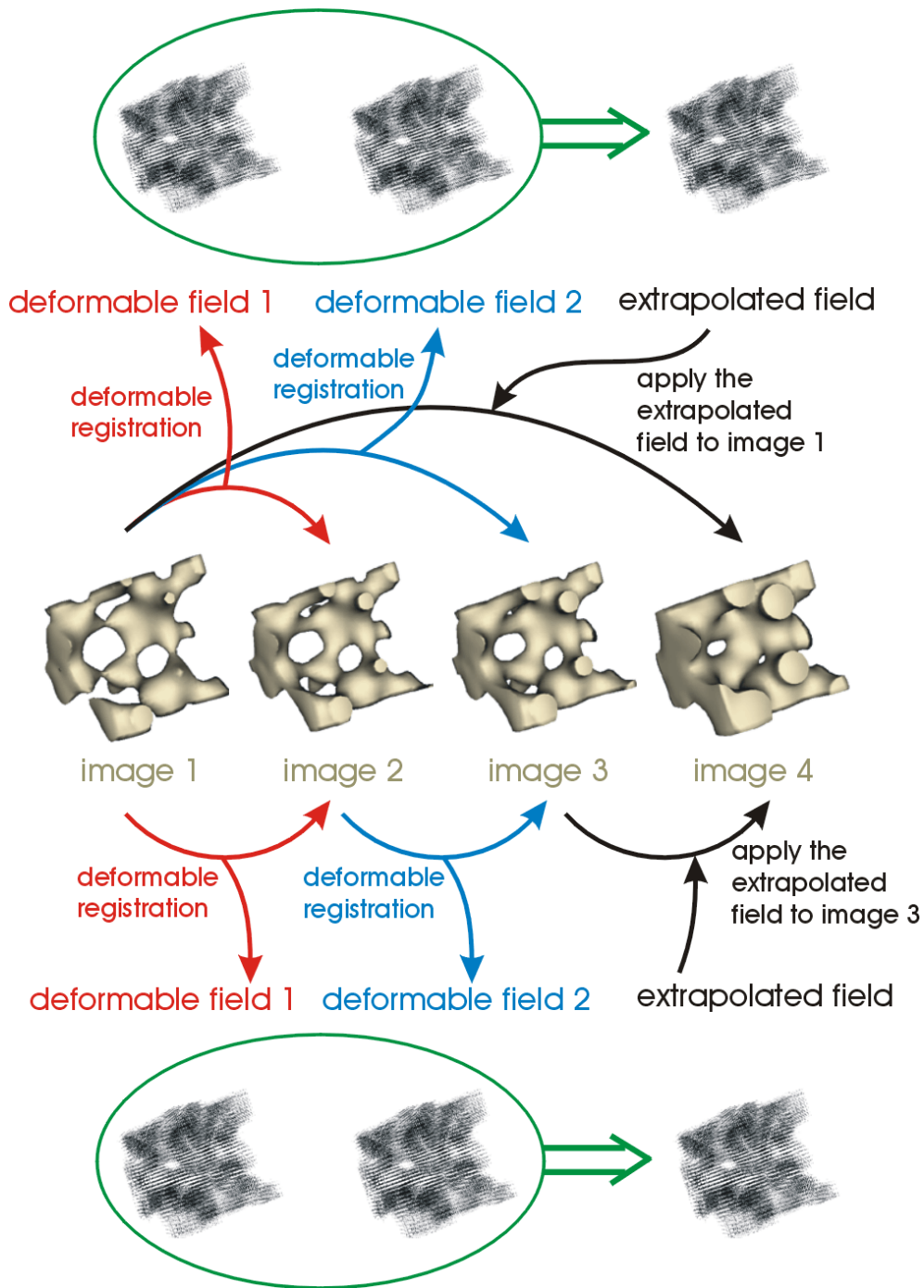


Figure 13.1: Two possibilities to choose the deformable fields for the extrapolation: In the upper part, the deformable fields used for registration refer always to the first image. The second possibility, the one we chose, is shown in the lower part of this figure and considers the deformable field between two subsequent images.

this reference image (1-2; 1-3; 1-4; ...). See upper part in figure 13.1. By looking at the same voxel in each inverted deformable field (see chapter 13.2.2), you

would get the shift of this voxel intensity in reference to the first image in each time point and it would be quite easy to interpolate it into the future. The problem with this method is that, if you look at pictures over a large amount of time, the change in bone might be huge and therefore the registration would get poor.

- The other possibility is to always calculate the deformable field between two consecutive images (1-2; 2-3; 3-4; ...). See lower part in figure 13.1. With this method, it is more difficult to extract the belonging shifts, however, the big advantage is the better registration due to smaller changes.

As a good registration is an important requirement to be able to extrapolate, we decided to implement the second method; to take the deformation fields between to subsequent images.

13.2.2 Forward versus Backward Field

There are two principal ways how to represent a deformable field:

- **Forward field**

The first method is to have a vector in each voxel pointing to the location where this voxel is shifted to in the next step. In other words it tells you, *where this particular point goes to*. We call this field a forward field.

- **Backward field**

The backward field is more difficult to understand but it is how everything is implemented in ITK. In a backward field, the vector tells you, which point is going to be shifted to the voxel the vector is stored in. It shows *where the point comes from*.

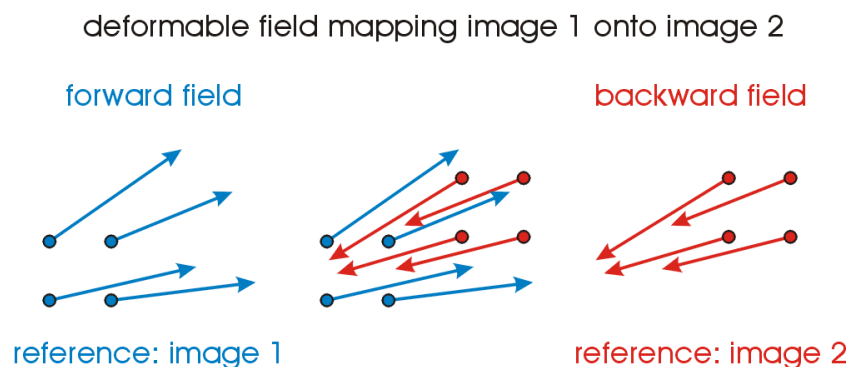


Figure 13.2: Comparison of the forward and the backward field.

In figure 13.2, an example of a forward and a backward field is shown for the same deformation. The big difference between this two types of field is that the forward field takes image 1 as a reference whereas the backward field has image 2 as reference.

The reason why in ITK the deformable field is implemented as a backward field is that it is much simpler to be applied to an image: You just iterate through the image, go along the vector stored in the voxel and you know the position of the image value you are looking for. As this position is hardly ever an exact voxel position, the last step to be done is to interpolate between the surrounding voxel intensities.

In a forward field, however, you know where the points move to. But this will not be an exact voxel position, either. The big difference to the backward field is that you know the values at random positions. To get a certain voxel value, you would have to interpolate between the known intensities at these random positions. Yet how can you find the closest ones you would need for interpolating? You might even have some "holes", some areas no vector is pointing to: How accurate would an interpolation be at those positions? For all these reasons it is more exact and mainly much easier (programming complexity) to use a backward field.

13.2.3 Finding the Belonging Vectors to Extrapolate

We had two ideas how to choose the vectors in the different deformable fields used for the extrapolation:

- One possibility is to look at the same voxel in every deformable field. You would then see by how much this point is shifted in each step and you could extrapolate based on those vectors and predict the future change.

We did not implement this idea because this method did not seem to lead to an accurate prediction. Figure 13.3 shows on the left side the voxel values along the x-axis of three different images and on the right side the resulting deformable fields mapping image 1 onto image 2 and image 2 onto image 3. It can be seen that the fields are biggest around the borders of the images (transition from 127 to 0 and vice versa) and they get smaller further away from a border. If you imagine the field over time in one voxel (e.g. position 7 in the right chart), it gets bigger. In a next step, however, it would get smaller. This means for the prediction that you somehow have to know if in the next step, the deformable field gets bigger again or if it gets smaller. This could be quite difficult to realize.

- The second method is the one we implemented. The idea is that you look always at the same piece of bone and consider by how much this piece is shifted in each step. This means - suppose we have a backwards field - that you begin with a voxel (see figure 13.4, point 4) in the last deformable field (field 3) and add the vector stored in this voxel to the starting voxel position. Like this you get the position of the same part of bone in image 3 (point 3). As this position is very unlikely to lie exactly in a voxel, the next step is to interpolate between the vectors of the surrounding voxel. Like this you can calculate the position of the same part of bone in image 2 and so on.

When you have reached the first deformable field, you have the vectors which tell you exactly how a specific point has been shifted over time. These vectors (we

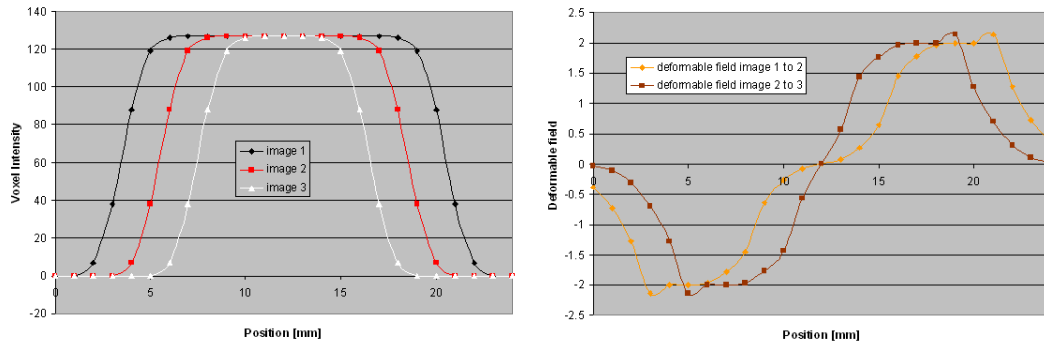


Figure 13.3: The left picture shows the voxel values of three images along the x-axis. The right picture visualizes the deformable fields between the first two and the second two images.

will call them a series of vectors) can now be extrapolated. The implementation of the methods is described in section 13.3.

After having extrapolated the vectors, you get a predicted field containing a vector in each voxel with the information, where this point is going to be shifted in the future. But you know it in reference to the voxel you looked at in field 3 (where you started). This means that you get a forward field which has to be inverted before being usable in ITK. How we invert the field is described in the next section 13.2.4.

13.2.4 Inversion of the Field

As mentioned, it is necessary to invert the extrapolated deformable field before it can be used. Again, there are different methods doing this; four of them are described in the following section:

Negative field

The very simplest method is to just take the negative vector in each voxel (see left part in figure 13.5). Although this approximation might not be too bad, we did not look into this any further because it does not seem to be exact enough for our use.

Apply field to itself

Another idea is to apply the negative forward field to itself. This means that you take the negative field described in the last section, follow the vector stored in a voxel and interpolate between the surrounding vectors (in the same negative vector field) at that point. This interpolated vector is then stored in the voxel position you started.

The perfect solution to get the inverted field would be to move the forward field by

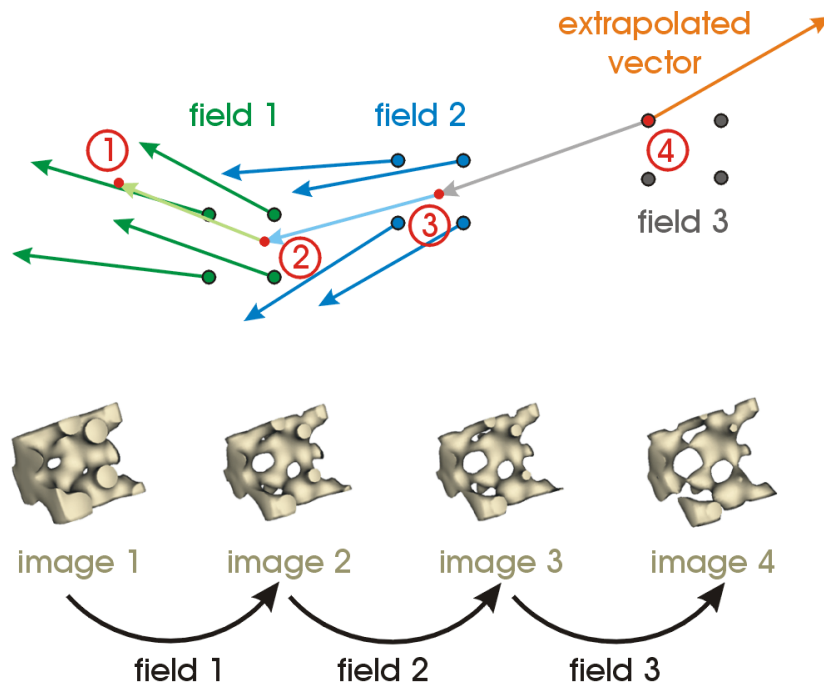


Figure 13.4: In this figure, it can be seen how we find the appropriate vectors used for the extrapolation. In principal this is done by following the deformable fields, in other words, by looking where each point came from. Like this we know exactly how each voxel position moved through space.

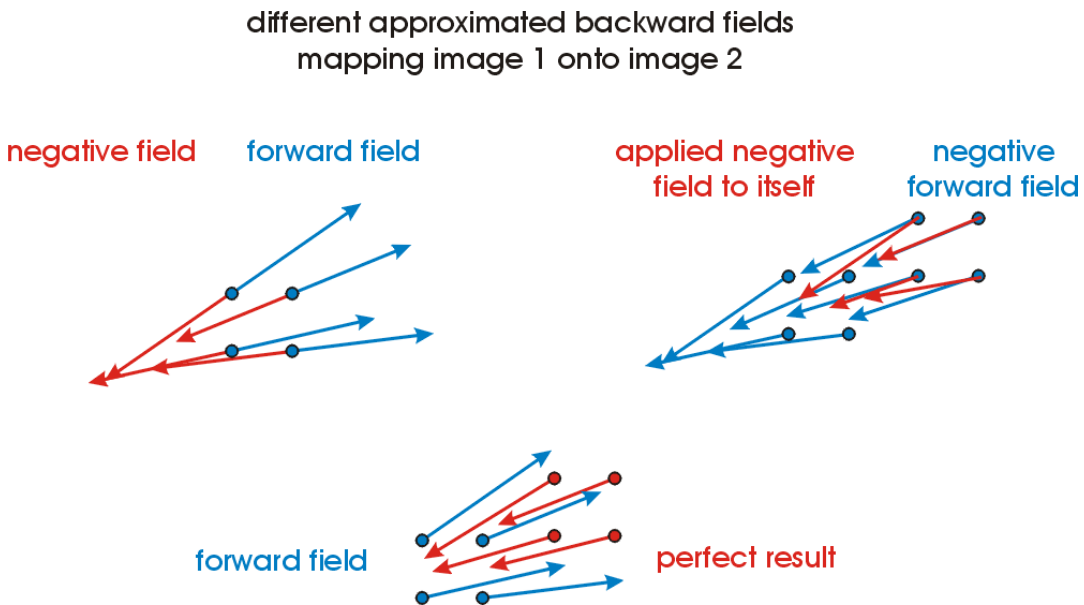


Figure 13.5: Principle of 2 (out of four) possibilities how to invert the deformable field. On the left side: Negative Field; On the right side: Apply field to itself

itself and then to take the negative vector in each voxel. But as this is not possible (the moved vector would not exactly be "in a voxel"), we need to approximate it. The idea behind applying the negative field to itself is, that with this field you can move the field in the right direction, however, not by the exact amount. Yet as the field is smooth, the shift is quite close to the "perfect" shift.

ITK filter: `itkInverseDeformationFieldImageFilter`

At the time we were thinking about how to invert a field, a new filter was added to ITK. However, the result of this filter was quite disappointing! It is implemented using landmarks and to get an acceptable result, the number of landmarks has to be chosen big which makes the filter very slow. If less landmarks are selected, it gets faster but the result is so bad that it is not usable. Therefore, at the time of writing this report, the existing filter in ITK is not usable. Yet it might be worth looking into it in the future again!

Our Implementation: `itkIterativeInverseDeformationFieldImageFilter`

As the filter in ITK was not working properly, Luis Ibàñez [3] suggested another approach: The idea is to first calculate an approximation of the backward field (e.g. with the existing filter in ITK) and then to perform several iterations until the error $e = p - F(B(p))$ is very small, whereas F is the forward field, B the backward field and p a voxel position. This means that you start at a position p in the backward field (see figure 13.6, first sketch), go along this vector (to position q) and interpolate the forward field at that position. If you now go along the forward field, you should get to exactly the same position p you started. As this is hardly ever the case, the distance to the starting position is defined as the error e . The goal is now to minimize this error by looking where the vectors around position q point to (second sketch in figure 13.6). If any of these forward vectors points closer to p , a vector to that position is stored in p .

This would be one iteration of the filter. In the next iteration, the same calculation is done for the new point q . If no vector with a smaller error e can be found, the distance s is set to a smaller value so that also other points can be taken into account.

In our implementation, the initial guess of the backward field is calculated with the method "apply field to itself" described on page 111 because this method is much faster than the existing filter in ITK.

13.3 Different Approaches for the Extrapolation

At the time of programming this filter, only simulated data was available. That's why we supposed real bone loss to look like the simulated bone atrophy. We are aware that real bone behaves different, however, until today it is not fully understood how exactly it adapts and it therefore is impossible to correctly predict the structure.

As a result, the implemented filter can predict a future bone shape in the simplest case which is a very smooth and uniform site-specific change.

Five different approaches to find the extrapolated vector are implemented. However, it has turned out, that most of these methods lead to quite similar results.

The aim is to find an extrapolated vector at each voxel. For each vector, there are two parameters which can be influenced: the direction and the length. To find this vector, the described series of vectors (see section 13.2.3) is used.

- **linear extrapolation of the length in the direction of the last vector** (method 1)

In a first approach, the absolute value of each vector of the series is calculated. These absolute values correspond to the site-specific change between the time points. The changes are then linear extrapolated into the

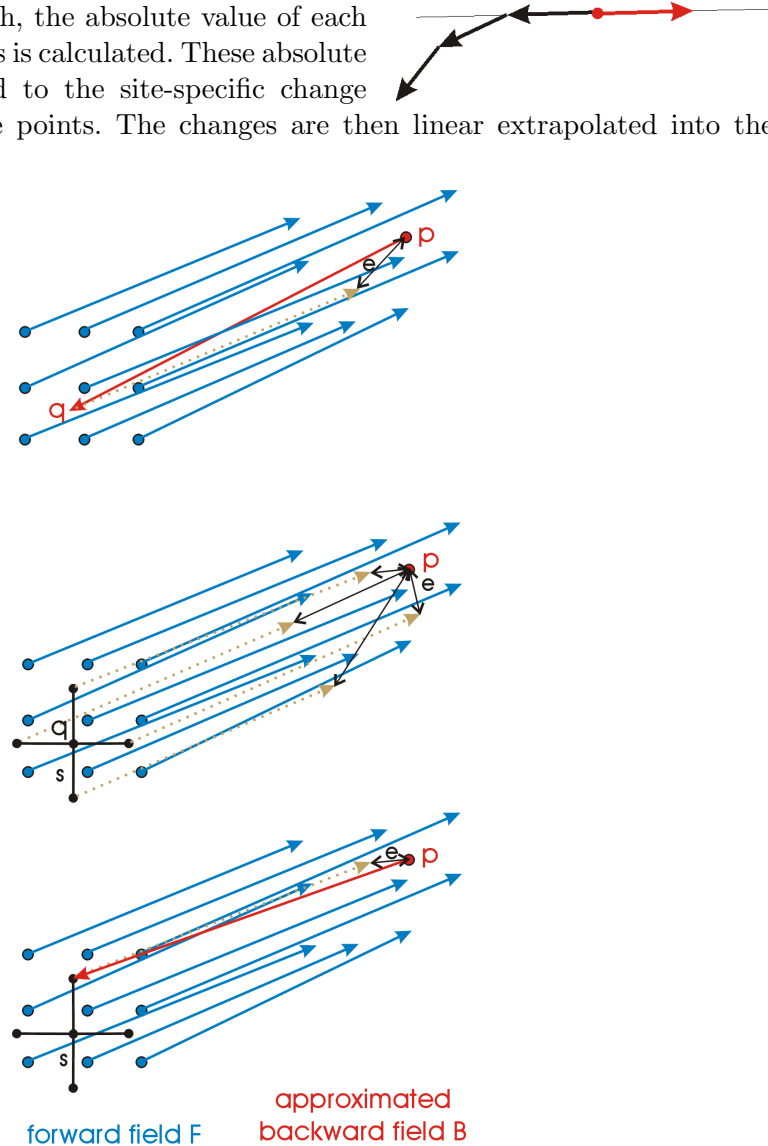


Figure 13.6: In this draft, the method of the itkInverseDeformableFieldFilter is visualized. The aim is to find a point q in the forward field whose vector minimizes the error e .

future.

For the direction of the extrapolated vector, just the same one as in the last deformation field is taken.

- **extrapolate each component separately** (method 2)

Another method we implemented is to linear extrapolate each component of the vector separately. Like this, the direction depends on all vectors earlier in time (in contrast to the approach described first). However, the length of the extrapolated vector gets rather overproportioned long or short. That is why we also implemented the next method.



- **extrapolate each component separately, adapt the length to a linear extrapolation** (method 3)

This is a combination of the first two ideas. The *direction* is calculated by the second method - to extrapolate each component separately. As the *length* in method 2 is not directly correlated to the ones in the earlier deformable fields, it is calculated as the linear extrapolation of the absolute values (method 1).



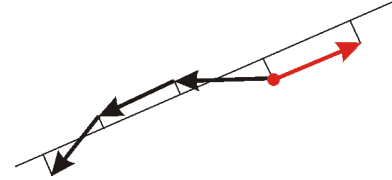
- **3D linear extrapolation** (method 4)

Another idea is to find a linear approximation of the series of vectors in space. This linear approximation is represented by the following formula:

$$\text{extrapolated change per time} = t_n * \vec{k} + \vec{c}$$

This leads to a least square problem of the following form:

$$\begin{bmatrix} t_0 & 1 \\ t_1 & 1 \\ t_2 & 1 \\ t_3 & 1 \\ \dots & \dots \end{bmatrix} * \begin{bmatrix} \vec{k}^T \\ \vec{c}^T \end{bmatrix} = \begin{bmatrix} cpt_0^T \\ cpt_1^T \\ cpt_2^T \\ cpt_3^T \\ \dots \end{bmatrix}$$



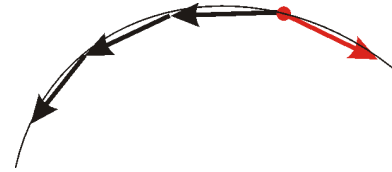
cpt... change per time

- **3D quadratic extrapolation** (method 5) As the linear extrapolation in space is not accurate enough, we also implemented the quadratic version:

$$\text{extrapolated change per time} = t_n^2 * \vec{k} + t_n * \vec{l} + \vec{c}$$

The according least square problem is:

$$\begin{bmatrix} t_0^2 & t_0 & 1 \\ t_1^2 & t_1 & 1 \\ t_2^2 & t_2 & 1 \\ t_3^2 & t_3 & 1 \\ \dots & \dots & \dots \end{bmatrix} * \begin{bmatrix} \vec{k}^T \\ \vec{l}^T \\ \vec{c}^T \end{bmatrix} = \begin{bmatrix} cpt_0^T \\ cpt_1^T \\ cpt_2^T \\ cpt_3^T \\ \dots \end{bmatrix}$$



13.4 Evaluation

13.4.1 with simulated data

In a first test, we applied the implemented extrapolation methods to simulated bone changes. This has the advantages that there is just bone loss appearing and that the images are registered perfectly. Therefore, you do not get any errors due to bad registered locations.

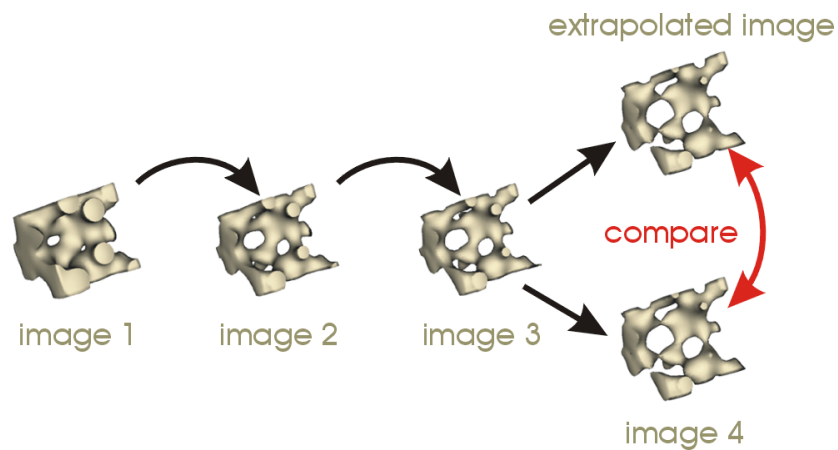


Figure 13.7: This sketch shows how we evaluated the extrapolation method. Images 1 to 4 are simulated images. An extrapolation is calculated based on images 1 to 3 which should match exactly image 4.

For each method, we reported the volume and the mean trabecular thickness of the calculated extrapolated image. These values can be compared to a simulated future image (image 4 in figure 13.7). Additionally, we calculated the error between the calculated and the simulated predicted structure with the geometric approach. In the following table, the results of these measurements are shown (images: dog01_gauss_sub.aim; dog02_gauss_sub.aim; dog03_gauss_sub.aim; reference: dog04_gauss_sub.aim):

extrapolation method	reference image	method 1	method 2	...
volume				
PolyData	1.75237	1.75279	1.72901	
ImageData	1.75095	1.75217	1.72698	
thickness				
Mean (surface weighted)	0.18095 mm	0.178777 mm	0.177038 mm	
Standard deviation	0.0432178 mm	0.0456776 mm	0.0458049 mm	
Median	0.177 mm	0.177 mm	0.176 mm	
Unweighted mean (circumf.)	0.170058 mm	0.165548 mm	0.163296 mm	
error				
Mean		0.000347016 mm	0.000708805 mm	
Standard deviation		0.00617167 mm	0.00669209 mm	
Median		0 mm	0.000238223 mm	

...	method 3	method 4	method 5
	1.75039	1.78694	1.72888
	1.74978	1.78814	1.7267
	0.17877 mm	0.178072 mm	0.176927 mm
	0.0455548 mm	0.044273 mm	0.04563 mm
	0.177 mm	0.177 mm	0.173 mm
	0.165334 mm	0.16592 mm	0.163225 mm
	0.000368148 mm	-0.0012301 mm	0.000707328 mm
	0.00649743 mm	0.00596569 mm	0.00668953 mm
	0 mm	-0.000318225 mm	0.000238223 mm

It can be seen that for simulated data, most of the methods behave similar. This has to do with the very nice site-specific change SIBA produces.

In this example, method 1 (linear extrapolation in the direction of the last vector) and method 3 are the most accurate ones (direction: linear extrapolate each component; length: linear extrapolation of the absolute value). However, this does not mean that they are also the best choice for real data.

To see by how much the algorithm had to extrapolate, the volume and thickness of the input images (dog01 to dog03) are shown:

	image 1	image 2	image 3
volume			
based on PolyData	2.50195	2.27757	2.02441
based on ImageData	2.46696	2.25393	2.01229
thickness			
Mean (surface weighted)	0.220128	0.206158	0.193627 mm
Standard deviation	0.0480978	0.0463138	0.044163 mm
Median	0.217	0.205	0.193 mm
Unweighted mean (circumference)	0.208221 mm	0.194554 mm	0.182813 mm

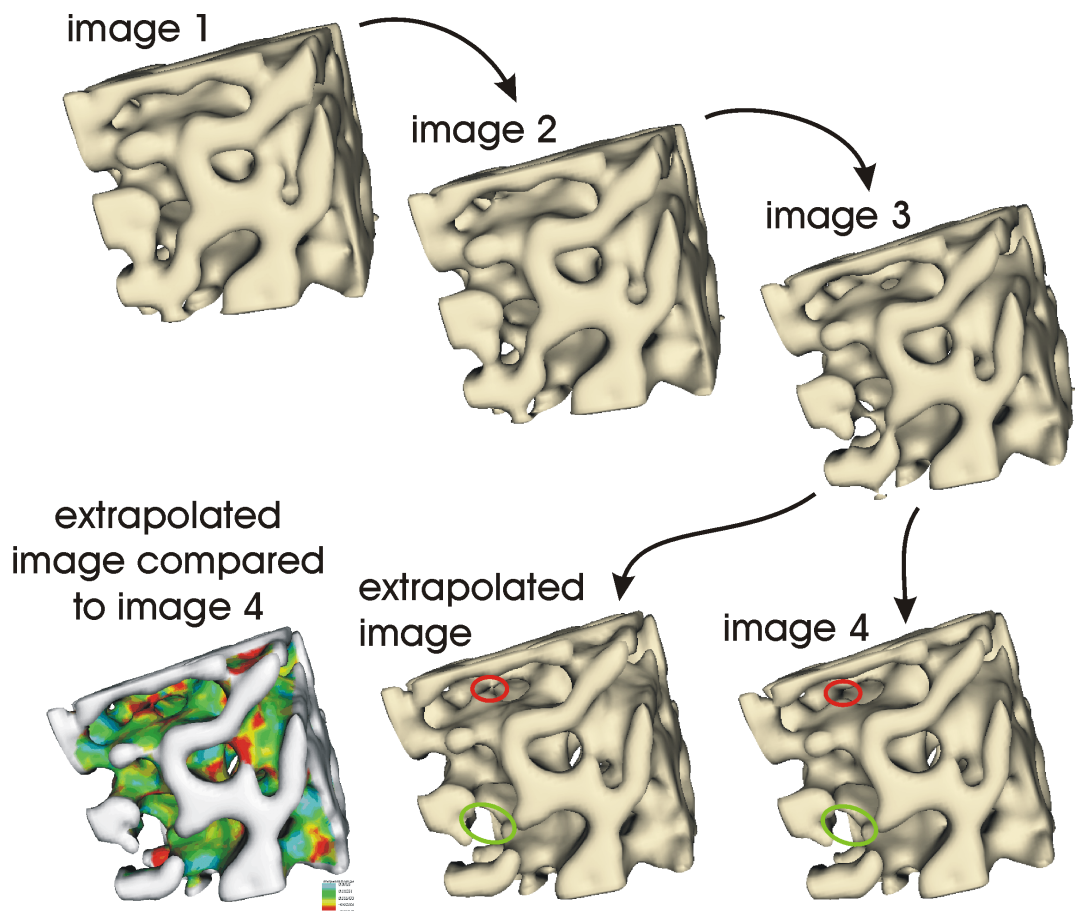


Figure 13.8: Images 1 to 4 are calculated with SIBA. Based on the first three images, an extrapolation has been calculated which should match exactly image 4. red circle: example where extrapolation did not work completely ; green circle: example where it predicted correctly. The scalar bar ranges from $-10\mu\text{m}$ to $10\mu\text{m}$.

In figure 13.8, images 1 to 4 and the extrapolated image are shown. The predicted image should match exactly image 4. As the applied simulated changes are quite small, the two following regions are pointed out: The algorithm was able to correctly

predict the green encircled position. The red encircled disappearing trabecula, on the other hand, could not be predicted correctly.

To get an overall impression how the extrapolation worked in this case, the colored picture shows the site-specific changes between the extrapolated image and image 4. All parts colored green and yellow have an error less than $\pm 5\mu m$ which is quite accurate compared to the image resolution of $34\mu m$.

13.4.2 with real data

Even though it looks quite good with simulated data, the situation gets much different when it comes to real data.

- In a first test we tried to extrapolate sham-data. This is data where there should be no or almost no change. But if you look at the data, you can see that there are changes. Even though the overall shape stays the same, some holes and even trabecula disappear and appear again. This makes a correct prediction at that positions - at least based on the data we had at the time of writing this report - quite impossible.

In figure 13.9¹, an example of the extrapolation is shown. The first three images (1 to 3) show the bone at time points 0, week 4 and week 8 whereas image 4 represents the bone at week 12. On the bottom left, the extrapolated image based on images 1 to 3 is shown which should match image 4.

Two positions are pointed out where it can clearly be seen that the algorithm calculated a linear extrapolation based on the first three images. Yet as the real bone change behaved non-linear at these positions, the results are different.

- Then we also tried to extrapolate OVX-data. The result can be seen in figures 13.10 and 13.11. As in the SHAM-example, we calculated an extrapolation based on the first three images in order to use the fourth image as a reference. It can be seen that the filter works in the right direction, however, there are positions where it does not match image four. If we look at the reason for this, we can see again that the bone adaptation does not follow a linear function. The very opposite is the case: It is highly non-linear and there even seems to be a local bone apposition from image 3 to 4 in figure 13.10. Possible reasons for this behaviour will be discussed in section 16.

¹images: B2.4OVX\C0000523_sub_gfilt_sub_sub50.aim; C0000751_sub_gfilt_regist12_sub_regist12_sub50.aim; C0001018_sub_gfilt_regist123_sub_regist123_sub50.aim; C0001128_sub_gfilt_regist1234_sub_regist1234_sub50.aim

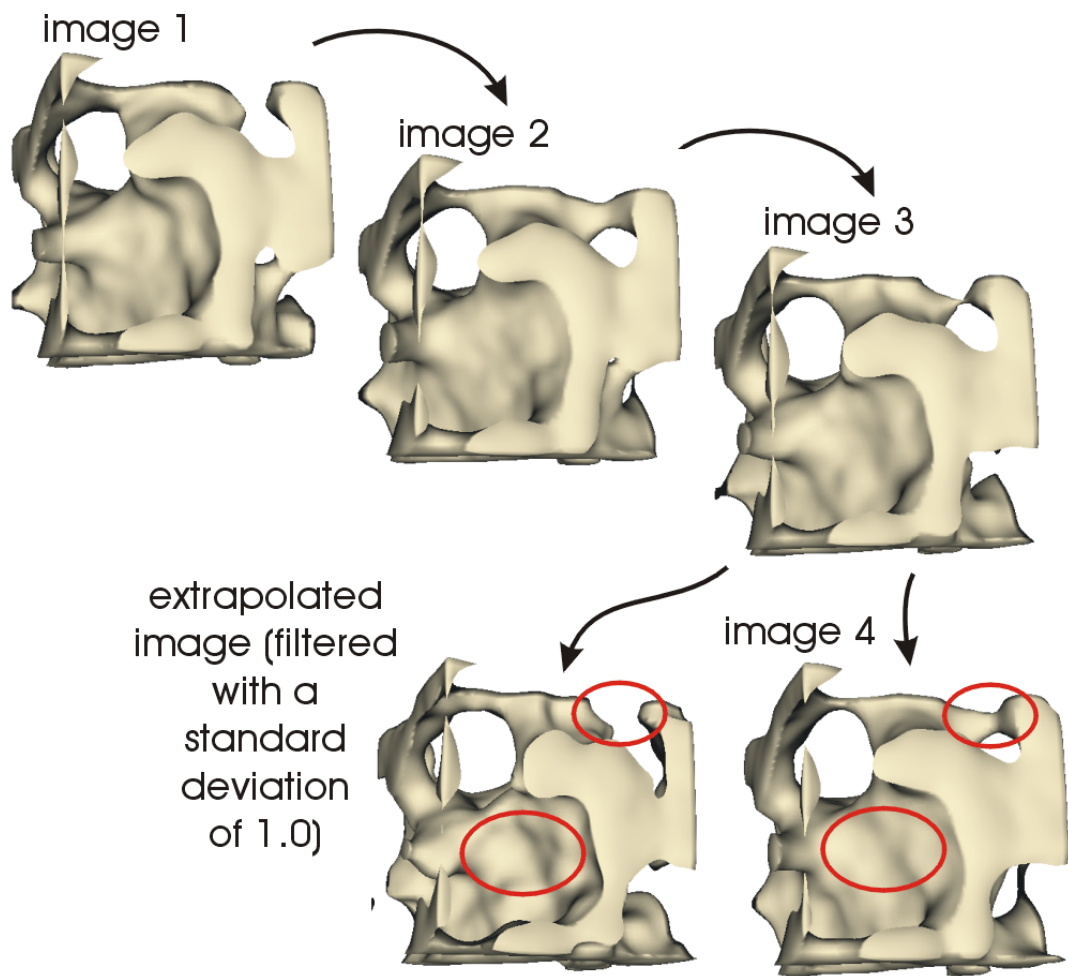


Figure 13.9: This is an extrapolation based on SHAM-data (A1.4). The extrapolation has been calculated based on images 1 to 3 so that it can be compared to image 4.

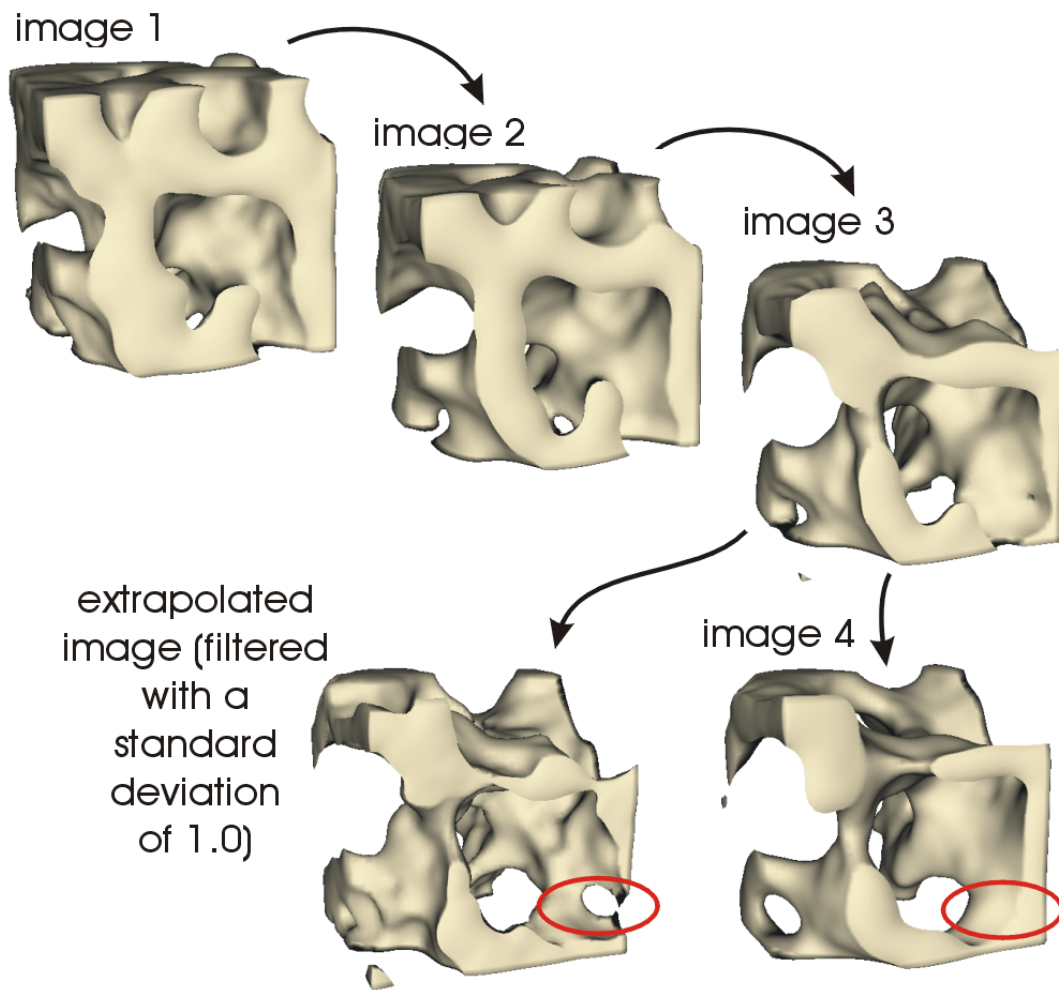


Figure 13.10: This is an extrapolation based on OVX-data (B2.4). The extrapolation has been calculated based on images 1 to 3 so that it can be compared to image 4. The red circle shows a position where the prediction is completely different to the real change.

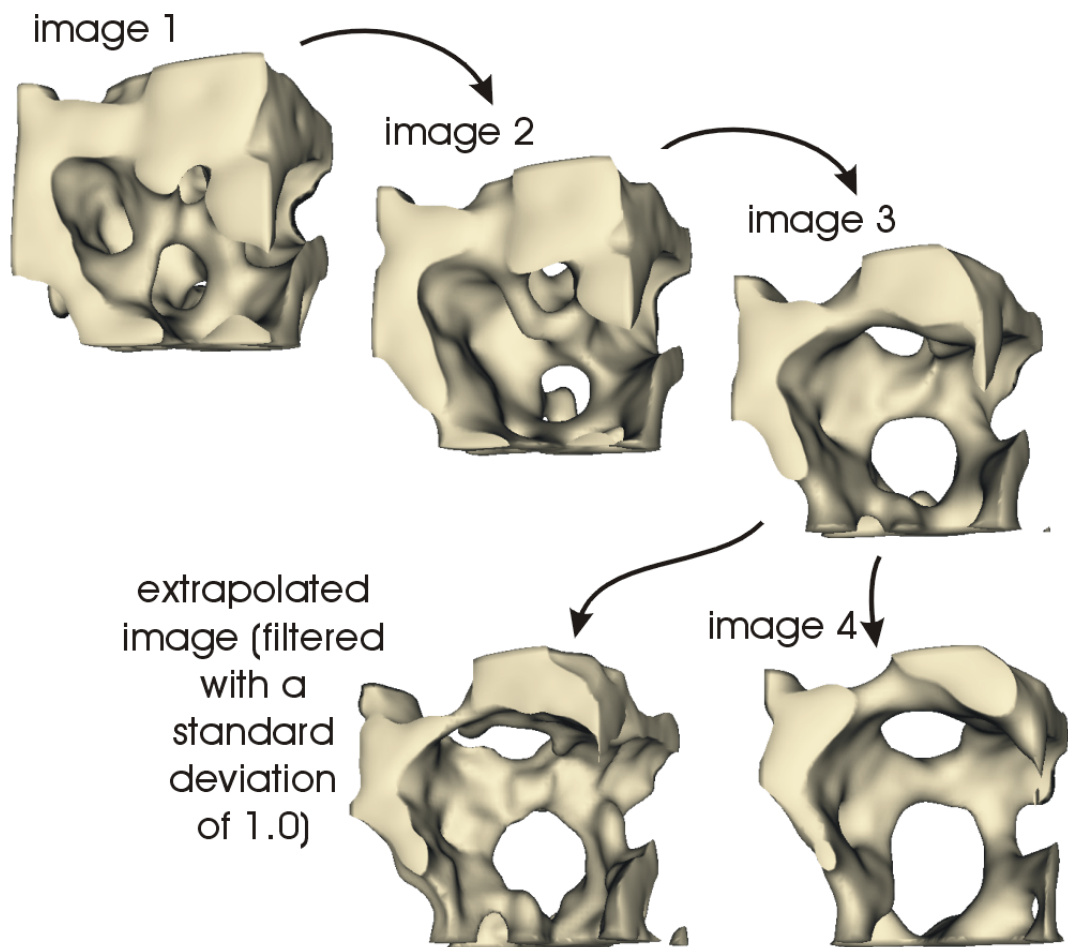


Figure 13.11: This is an extrapolation based on OVX-data (B2.4). The extrapolation has been calculated based on images 1 to 3 so that it can be compared to image 4.

13.5 Discussion

In conclusion we could say that the extrapolation method is a promising first approach which will have to be improved based on better knowledge of how bone adapts over time. It is a basic attempt which calculates an extrapolated field and which can actually apply this field and create a linear extrapolated image.

The extrapolation algorithm is a method where we exactly know what we are doing (looking where a voxel position has been shifted through space). This makes it to a principle which can easily be extended. Even though the deformable registration might be messy in some places (e.g. where a trabecula is resorbed), this should not have a great impact on the result of the extrapolation. In our opinion, it is important to achieve a good registration where there is a smooth change - which is fully fulfilled by the demons deformable registration - as it is very difficult to predict what happens at a position just after a trabecula has resorbed even if the deformable registration was correct at that position.

What can be said for sure is that the extrapolation works better with small changes. On one hand because the deformable registration is more accurate. Another reason is that the bigger the extrapolation step, the more errors are in the extrapolated image. As we have to invert the extrapolated field before it can be applied to the last image, there are also errors happening during the inversion as there is a bigger possibility that two vectors point to the same position which results in a "random" inverse field.

Very important - may be even the most important thing - are good registered initial images. If the registration is not done very perfectly, there might be in one image a shift in one direction and in the next one in the other direction. The extrapolated change would then just be some random shift which has nothing to do with the real bone change.

Future work should include a collection of data with more time points. This would allow a better understanding of what is going on and it would give the possibility to calculate a better prediction of the structure of the bone.

Part IV

In vivo measurements

Chapter 14

Volume and thickness

Figure 14.1 and 14.2 show the BV/TV and the thickness measurement of the in vivo data. One very interesting point we would like to mention about the OVX specimens is that each individual rat loses the bone in a different speed over time. That would mean that it is not best having as much bone as possible when start losing bone but that there are other influences.

However, we have to be careful with too many interpretations on this dataset because we did not have control over the measurement. The data was provided by Juerg Gasser, Novartis, Switzerland.

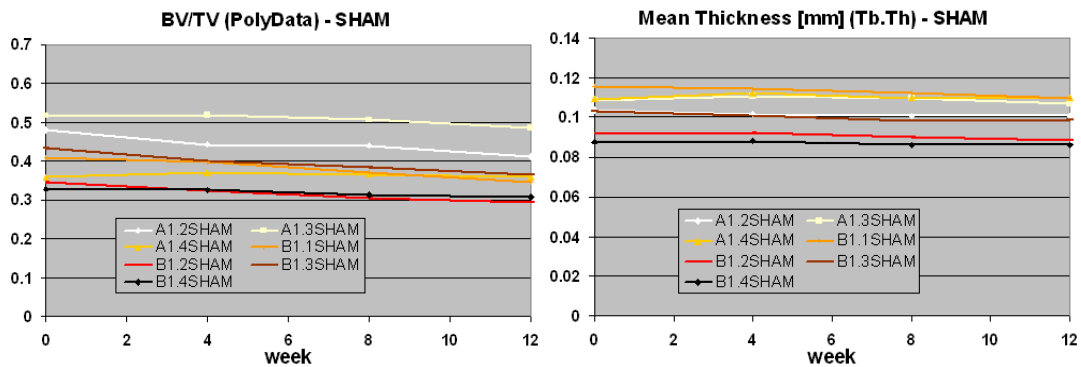


Figure 14.1: This figure shows the BV/TV and the thickness measurement of the SHAM operated rats. It can be seen that the changes are very small.

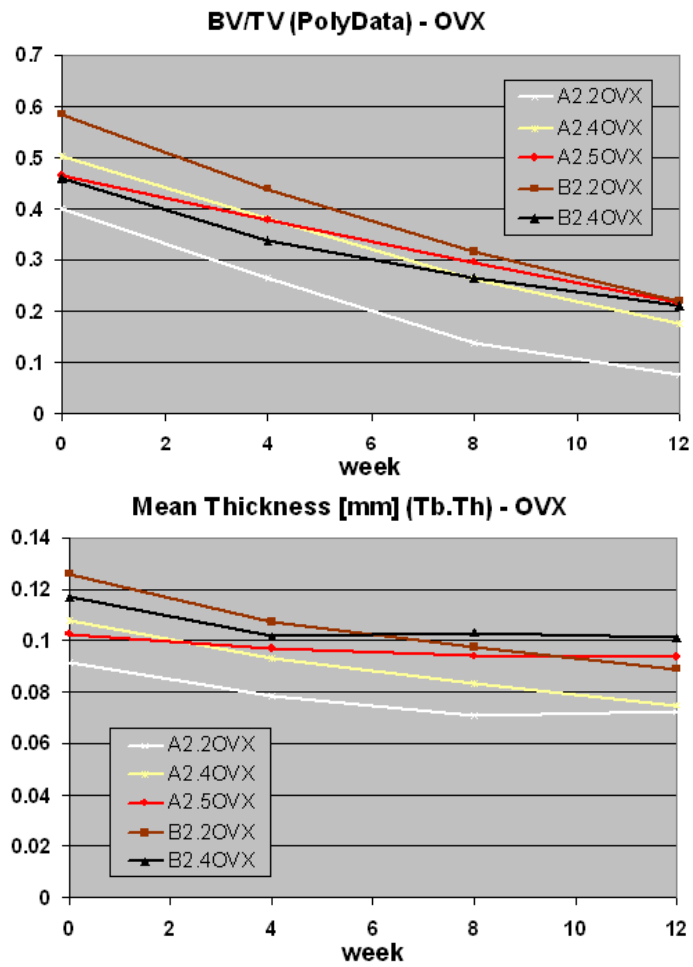


Figure 14.2: This figure shows the BV/TV and the thickness measurement of the sham OVX rats. It can be seen that there are significant changes.

Chapter 15

PCA

By applying the principle component analysis to the in vivo data we saw that the smallest principle component was always well aligned with the z component. Figure 15.2 shows a good representative of a loss ellipsoid, viewed from two different directions. In figure 15.1 it can be seen that the z component is always smaller than the other two. If it is known that the z-direction is also the loading direction (see 15.1) we can conclude that the loading direction has the smallest bone loss. This point has also been mentioned in publications.

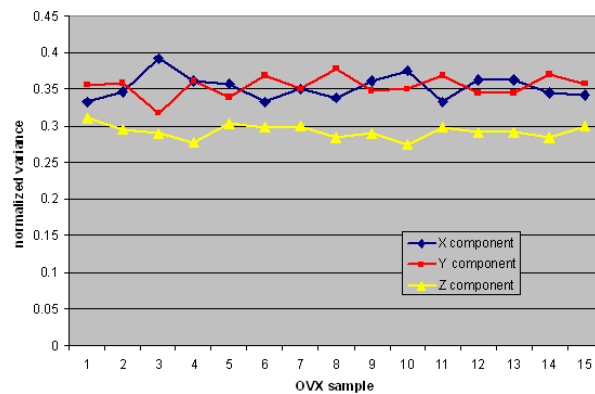


Figure 15.1:

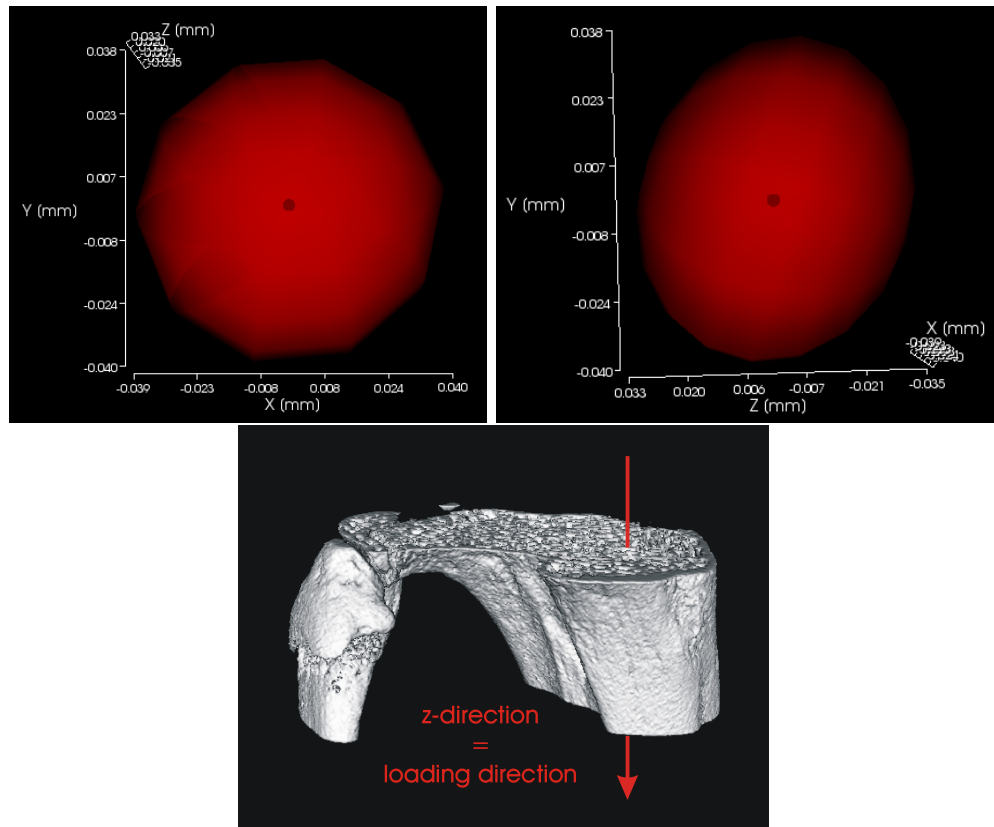


Figure 15.2: The top two images show the loss ellipsoid from two different directions. It can be seen that the z-direction is aligned with the smallest principle component. The lower image shows that the z-direction is aligned with the loading direction.

Chapter 16

Visual inspection

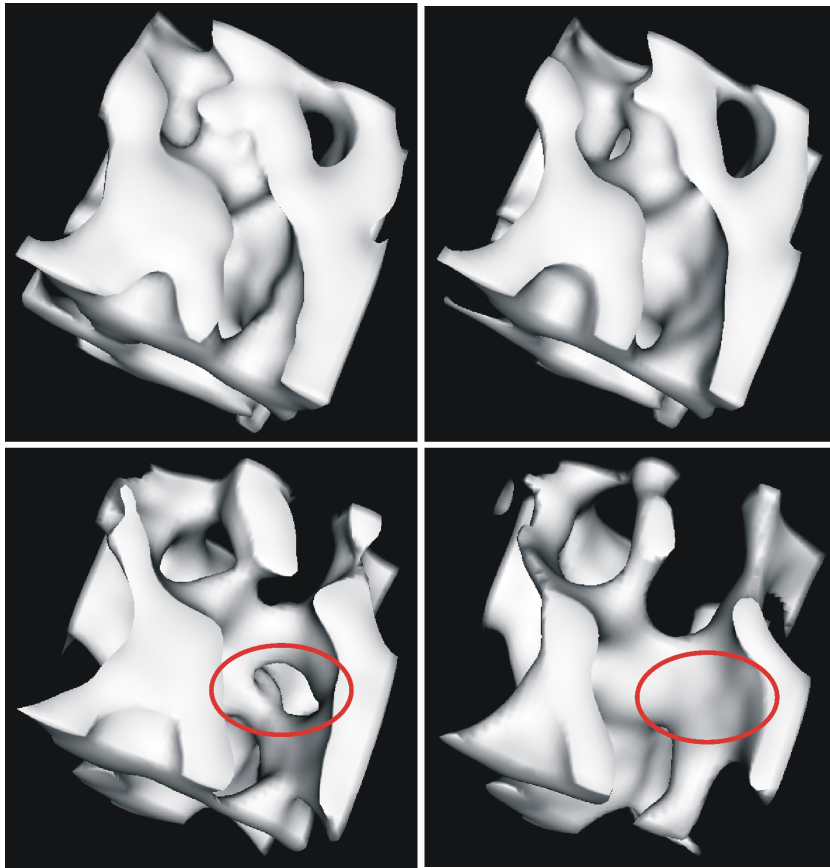


Figure 16.1: bone structure at week 0 (top left), week 4 (top right), week 8 (bottom left) and week 12 (bottom right). It can be seen that there seems to be a local bone apposition between week 8 and 12 (sample: A2.4 OVX).

We also just looked at the in vivo data and what we saw was quite unexpected. In some positions seems to be a local bone apposition (examples are shown in figure [13.11](#) and [16.1](#)). The examples are taken from OVX-data, however, the same can be

observed in SHAM-data.

Possible explanations we could think of are the following:

- A first thought was that this might be due to a wrong reconstruction (e.g. a wrong threshold). Yet this seems to be quite impossible as there is just next to the apposition a bone loss.
- Another explanation is that the bone might have shifted.
- A last explanation is that these strange bone changes are due to the radiation these rats were exposed. However, it seems more reasonable that radiation would accelerate bone loss instead of cause bone apposition.

This makes us aware once more that there is a lot of work to be done in order to understand what exactly is going on in the structure of the bone. It also shows that it is necessary to look at the site-specific changes to be able to understand how bone adapts.

Outlook

- Make in vivo scans with more time points than only four. Probably it would be easier to find extrapolation functions that fit the evolution of the bone and it would also help to understand the changes in the trabecular bone. Whatever the specimens are, always scan a piece of aluminum foam with known diameter to check the reproducibility of the scanner.
- The search speed of the normals and the closest point method could be improved by implementing a faster search structure than the `vtkCellLocator`. (see [5.5.1.2](#))
- The iterative closest point method (ICP) is an algorithm that minimizes the mean error between two surfaces by applying a rigid transform to one of them. The method is based on landmarks that are placed on the two surfaces using the closest point method. The distances between these landmarks are then minimized. This algorithm could be used prior to the geometric methods in order to improve the registration because it registers the two surfaces that will be used to measure the site-specific changes and not the image data. It cannot be used prior to the deformable registration because the ICP works only on poly data and not on image data.
- At the moment the principle component analysis of the site-specific vectors returns an ellipsoid whose shape depends on the number of points and the length of the vectors in this direction (see also [10.4.1](#)). The number of points is highly dependent on the structure of the bone and is not a function of the actual site-specific change. To look only at the structure the vectors could be normalized prior to the PCA. Consequently, the shape of the ellipsoid would only depend on the number of points in each direction. To look only at the length of the vectors, is a little bit more difficult. An idea is to compare the ellipsoid without normalized vectors and the one with normalized vectors to each other. The difference should then be due to the length of the vectors.
- Look at the ellipsoids of the principle component analysis over multiple time points and try to predict future ellipsoids if certain trends are visible.
- Extend the three dimensions of the principle component analysis with the time as fourth dimension in order to find new ways of generalizing the results.
- Use neural networks in order to extrapolate the deformable fields of the deformable registration by using local extrapolation function and not a global one because it seems that that the bone changes not in all places with the

same function. A book about neural networks is: Neural Networks for Pattern recognition, Christopher M. Bishop, Oxford, University press, reprinted 1999 ¹

- In this project the principle component analysis was used to generalize the site specific change vectors. This analysis finds the direction of the biggest variance. This variance could be due to noise or random translations of the bone in different directions and contains therefore no information about the real site-specific changes. Therefore an independent component analysis could be used to generalize the site-specific vectors instead. This analysis finds the directions of maximal non gaussian variance. A book that talks about this analysis is: Independent component analysis, Aapo Hyvarinen, Juha Karhunen, Erkki Oja, John Wiley & sons, Inc , New York 2001 ²
- Improve the algorithm which detects resorbed trabecula and new holes by finding constraints to exclude more wrong points.
- The algorithm that identifies resorbed trabeculas does not return the position or direction of the resorbed trabeculas at the moment. The position could be easily returned by just finding the center of the connected regions. The connectivity filter assigns scalars to the points and each connected region has the same number. The center could then be found by just making a mean over all points with the same number assigned. The direction is harder to find because new holes and resorbed trabeculas have to be separated first.

¹recommendation by Vincent von Tscharner, HPL, UofC

²recommendation by Vincent von Tscharner, HPL, UofC

Bibliography

- [1] Will Schroeder, Ken Martin, Bill Lorensen; *The Visualisation Toolkit*; ISBN 1-930934-07-6
- [2] Jean-Philippe Thirion; *Fast Non-Rigid Matching of 3D Medical Images*; May 1995; Publication no. 2547 at INRIA
- [3] Luis Ibáñez, William Schroeder; *The ITK Software Guide*; August 21, 2003; ISBN 1-930934-10-6
- [4] *www.itk.org*
- [5] *The VTK User's Guide*; Kitware, Inc.; ISBN 1-930934-08-4
- [6] *VTK HTML Documentation*; <http://www.vtk.org/doc/nightly/html/>
- [7] Hildebrand T., Rüegsegger P.; *A new method for the model-independent assessment of thickness in three-dimensional images.*; 1996; Journal of Microscopy, 185:67-75
- [8] Whitehouse W.J.; *The quantitative morphology of anisotropic trabecular bone.*; 1974; J. Microsc. 101:153-168
- [9] Lorensen W.E., Cline H.E.; *Marching cubes: A high resolution 3D surface construction algorithm*; 1987; Computer Graphics, 21:163-169
- [10] Müller R., Rüegsegger P.; *Analysis of mechanical properties of cancellous bone under conditions of simulated bone atrophy*; 1996; J. Biomechanics, Vol. 29, No. 8, pp 1053-1060
- [11] Harrigan T.P., Mann R.W.; *Characterization of microstructural anisotropy in orthotropic materials using a second rank tensor*; 1984; J. Materials Science, Vol. 19, pp 761-767
- [12] Weinans H. et al.; *Detecting and tracking local changes in the tibiae of individual rats: a novel method to analyse longitudinal in vivo micro-CT data*; 2004; Bone
- [13] Laib A. et al.; *The temporal changes of trabecular architecture in ovariectomized rats assessed by microCT*; 2001; Osteoporosis International 12:936-941
- [14] Boyd S. et al.; *Mechanical and architectural bone adaptation in early stage experimental osteoarthritis*; 2002; J. Bone and Mineral Research, Vol. 17, Number 4, pp 687-694

-
- [15] Odgaard A.; *Three dimensional methods for quantification of cancellous bone architecture*; 1997; Bone, 20:315-328
- [16] Kuhn Michael, Moser Stephan, Dr. Steven Boyd, Prof. Dr. Ralph Mueller; *Semester Thesis: Development of Intra-Modality Image Registration Methods for Assessment of Micro-Structural Bone Adaption*; 2003
- [17] <http://udn.epicgames.com/pub/Technical/UnrealScriptHighlighting/>, Option 2
- [18] <http://www.itk.org/pipermail/insight-users/2003-February/002429.html>

Appendix

Appendix A

Implemented Applications

This chapter contains some descriptions about the implemented applications. Mostly it is just the help which is also provided by the applications itself by typing the application name without any arguments in the command prompt.

A.1 AppSiteSpecificChange

An example can be found in the `vtkMy\dat-directory: SiteSpecificChange.bat`

- This application performs a site specific measurement between two surfaces. The surface earlier in time is `file1.aim` and the one later in time is `file2.aim`. The example shows a common setting for the application.
- The setting `-y` is a box widget which lets you taking subsamples of the whole dataset.
 - Figure [A.1](#) contains a screen shot of the application. The render window in the top left corner shows the overview of the whole dataset. The bigger render window on the right, always shows the selected subpart of the image. To select a subpart of the image use `"i"` in either of the two render window to bring up the box (if the box is inside the bone it is easier to use the box in the bigger render window because only the subpart is visible). The box can be rotated, translated and the size can be changed by using the handles. If the box is at the position the subpart is wanted, press again `"i"`. After the calculation is done the bigger render window will only contain the selected subpart. The report which is outputted to the command prompt will also be updated and also the two ellipsoids of the principal component analysis and the histogram of the change.
 - We suggest to use `"t"` after startup of the windows. It is easier to rotate the dataset when the box is visible because you can always rotate the box or the whole scene.
 - Use `"r"` for a camera reset in the bigger render window after selecting a subpart.

- Both render windows have the same camera therefore always the same part of the bone can be seen.
- The big advantage of this tool is that the rendering for a smaller piece is much faster than for a big one. Nevertheless, the whole dataset is in memory and you can by just moving the box around look at different positions in the bone.
- The bigger render window contains, in the bottom right corner, the mean vector which shows if the selected subsample is shifted. The arrow has the actual size of the shift therefore it is only visible if there is a substantial shift.
- The scalar bar in the bigger render window can be scaled and translated using first "b".

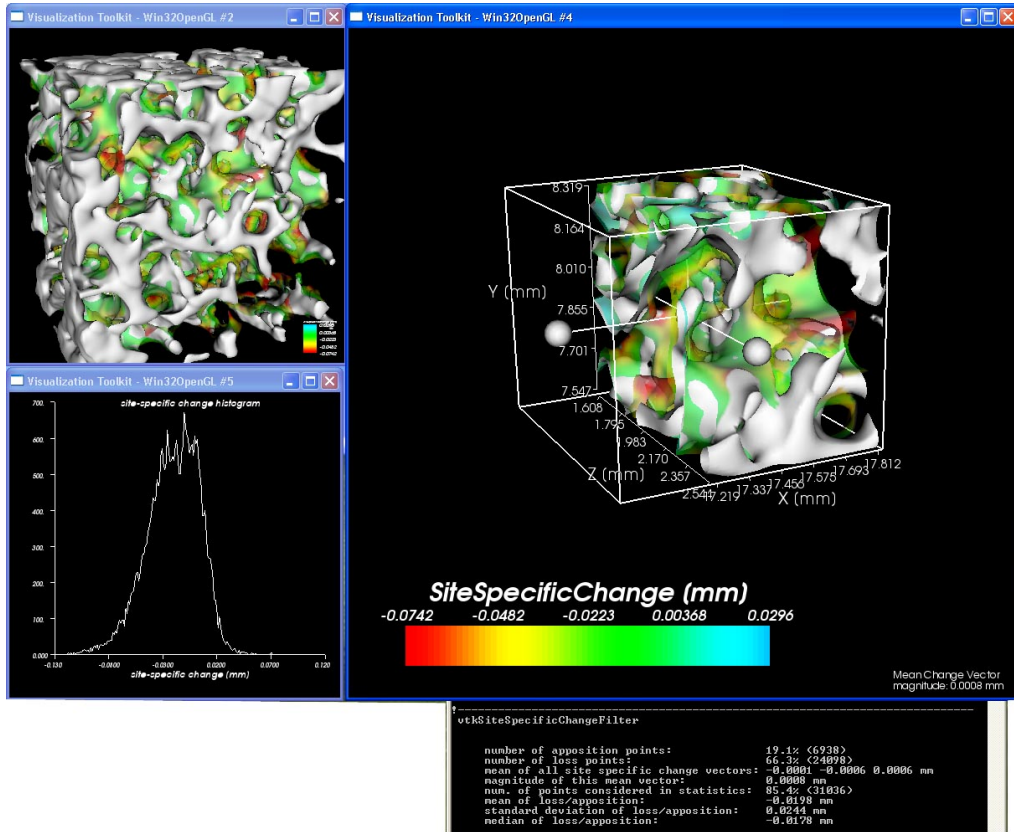


Figure A.1: site-specific change application with fancy box widget

Usage: AppSiteSpecificChangeFilter [-args] file1.aim file2.aim

Example: AppSiteSpecificChangeFilter -t 6400 -h -u -e -j -y image1.aim image2.aim
 This is an example of this application showing the histogram, closing the boundaries of the bone, forcing to exclude all points on the boundary, calculating the PCA and using the fancy box widget.

Arguments:

- Gaussian Image Smooth Settings
 - **-d arg1** : standard deviation (df. 1)
 - **-f arg1** : radius factor (df. 0)
- Marching Cubes Settings
 - **-t arg1** : threshold (df. 30)
- Padding Options (close the boundary of the bone)
 - **-u** : close the boundaries (df. off)
- Site-Specific Filter Settings
 - **-n** : change method to normals (df. closest point method)
 - **-s arg1** : search depth [mm] (df. 1)
 - **-o** : calculate statistics off (df. on)
 - **-h** : show histogram (df. off) (needs statistics -> don't use -o)
 - **-h2 arg1,2** : show histogram and set the range (df. auto)
 - **-b arg1** : set boundary tolerance [mm] (df. 0.1)
 - **-e** : force to exclude boundary points (df. off)
 - **-v** : visualize not considered points (df. off)
 - **-r arg1** : sphere radius not considered points [mm] (df. 0.01)
 - **-p** : pass normals off (df. on)
 - Settings Only for the Closest Point Method
 - * **-c** : don't attach closest point vectors (df. on)
 - Settings Only for the Normals Method
 - * **-g** : don't attach vectors (df. on)
 - * **-i** : intersect with own surface off (df. on)
- Visualize the Normals (used for finding the adequate search depth)
 - **-a** : visualizes the current search depth (change with -s)
 - **-m arg1** : mask normals (only show every arg1-th) (df. 1)
- Output
 - **-w arg1** : write output of site-specific filter to .vtk file
- Lookup Table
 - **-l arg1,2** : set lookup table range [mm] (df. auto)
- Render Window
 - **-x** : don't display axes (df. on)
 - **-q** : just visualize one surface (colored according change (df. off))

- **-z** : don't visualize the results in a render window (df. on)
- Generalization Settings
 - **-j** : show ellipsoids using PCA (df. off)
 - **-k** : calculate relative site-specific change (df. off)
 - **-k2** : show histogram of relative change (df. off)
 - **-k3 arg1,2** : change the range of the histogram (df. auto)
 - **-k4** : show surface, colored according to relative change (df. off)
 - **-k5 arg1,2** : set lookup table range [mm] for these colors (df. auto)
- Fancy Box Widget
 - **-y** : activates the box widget in order to clip subparts of the data (press 'i' in either of the two windows to bring up the clipping box; use handles to resize box; press 'i' to clip the data and calculate the statistics for this region). Use 'b' to move or resize the scalar bar.

A.2 AppSiteSpecificThickness

An example can be found in the vtkMy\dat-directory: SiteSpecificThickness.bat

This application performs a site-specific thickness measurement on the data set. Mean values with different weighting are also provided.

Usage: AppSiteSpecificThickness [-args] file1.aim

Example: AppSiteSpecificThickness -t 6400 image.aim

This is an example of this application showing the histogram of the site-specific thicknesses.

Arguments:

- gaussian image smooth settings
 - **-d arg1** : standard deviation (df. 1)
 - **-f arg1** : radius factor (df. 0)
- marching cubes settings
 - **-t arg1** : threshold (df. 30)
- padding options (close the boundary of the bone)
 - **-c** : close the boundaries (df. off)
- site-specific thickness filter settings
 - **-s** : change method to spheres surface weighted (df. spheres)

- **-n** : change method to normals (df. spheres)
- **-m** : measure separation with set method (df. off)
- **-b arg1** : set boundary tolerance [mm] (df. 0.1)
- **-p** : pass normals off (df. on)

- settings only for the spheres and surface weighted spheres method
 - * **-o arg1** : set initial offset sphere radius [mm] (df. 0.005)
 - * **-a arg1** : change accuracy (= $2^a r_{g1}$ [um]) (df. 0)
- settings only for normals method
 - * **-l arg1** : set length of normals [mm] (df. 2)

- histogram
 - **-h** : do not show the histogram (df. on)
 - **-h2 arg1,2** : set the range of the histogram (df. auto)

- output
 - **-w arg1** : write output of site-specific filter to .vtk file

- lookup table
 - **-r arg1,2** : set lookup table range [mm] (df. auto)

- render window
 - **-x** : don't display axes (df. on)
 - **-z** : don't visualize the results in a render window (df. on)

A.3 AppDetectResorbedTrabeculae

An example can be found in the `vtkMy\dat-directory: DetectResorbedTrabeculae.bat`

This application finds resorbed trabeculas and new holes from `file1.aim` to `file2.aim`. If you want to find holes that close or new trabeculas just insert the files the other way around. Yet the applications finds in some cases too many resorbed trabeculae. The report contains the number of trabeculas and holes as absolute number and per cubic millimeter. The visualization shows the identified places as colored blobs.

Because we could not template the vtk filter you have to change the typedef on line 24 of the filter `"/Unsorted/vtkDetectResorbedTrabecula.h"` either to *char* or *short* depending on the input images.

Usage: `AppDetectResorbedTrabeculae [-args] file1.aim file2.aim`

Example: `AppDetectResorbedTrabeculae -t 6400 -u image1.aim image2.aim`

Arguments:

- Gaussian Image Smooth Settings
 - **-d arg1** : standard deviation (df. 1)
 - **-f arg1** : radius factor (df. 0)
- Marching Cubes Settings
 - **-t arg1** : threshold (df. 30)
- Padding Options (close the boundary of the bone)
 - **-u** : close the boundaries (df. off)
- Resorbed Trabecula Filter Setting
 - **-b arg1** : set boundary tolerance [mm] (df. 0.2)
- Render Window
 - **-x** : don't display axes (df. on)
 - **-z** : don't visualize the results in a render window (df. on)

A.4 AppDeformableRegistration

An example can be found in the `vtkMy\dat-directory: DeformableRegistration.bat`

This application mainly uses the `itkDeformableRegistrationFiler` (see section [B.7](#)). Based on the two input images it calculates a deformable field which maps image 1 onto image 2. The deformable field can be stored in three aim-files (`-w filename`). To see how well the registration worked, the following images are visualized:

1. The deformable field is applied to the image 1. The resulting image is compared to image 2 with the `vtkSiteSpecificChangeFilter` and colored according to the registration error.
2. The original input images (image 1 and image 2). Image 1 is colored red and is transparent whereas image 2 is white.
3. The last visualization is the same one as the first (deformable field applied to image 1), however, not colored.

Usage: `AppDeformableRegistration [-args] image1.aim image2.aim`

Example: `AppDeformableRegistration -l 2 -i 3 5 -d 1 1 -s 2 2 2 image1.aim image2.aim`
This is an example for a deformable registration using a multi-resolution pyramid with 2 levels. It is also possible to just use the parameter `-l 2`. The other settings would than be chosen according to the default values.

Arguments:

- gaussian image smooth settings before registration
 - **-g arg1** : sigma [in voxel] (default = spacing)
- gaussian image smooth settings after registration
 - **-o** : gaussian smooth the images after registration (default = OFF)
 - **-m arg1** : sigma [in voxel] (default = spacing/2)
- deformable registration settings
 - **-l arg1** : number of levels in a multi-resolution pyramid (see 6.2.3) (default = 1)
 - **-i arg1...** : set number of iterations at each level (default = 4)
⇒ number of arguments = 'number of levels'.
 - **-d arg1...** : set the standard deviation for each level (default = 1.0)
⇒ number of arguments = 'number of levels'.
 - **-s arg1,2,3** : set the starting shrink factors (default = 1 1 1)
⇒ number of arguments = 'image dimension'.
- output
 - **-w filename** : write the deformable field to 3 aim-files (one for each component).
Notice that you have to specify just one basic file name.
The filter adds `_X`, `_Y`, `_Z` to this name.
 - **-v 0,1** : visualize the reconstructed image (default = 1)
0 = do not visualize; 1 = visualize.
- SiteSpecificChangeFilter settings
 - **-t arg1** : set the threshold to create vtkPolyData (default = 30)
 - **-r arg1** : set the search depth [mm] (default = 0.15)
 - **-h arg1,2** : set histogram range [mm] (default = auto)
- lookup table
 - **-k arg1,2** : set lookup table range [mm] (default = auto)

A.5 AppAutomatedTestDeformableRegistration

An example can be found in the `vtkMy\dat-directory: AutomatedTestDeformableRegistration.bat`

This application performs several deformable registrations between image 1 and image 2 using different combinations of the 'number of iterations' and the 'standard deviation'. The output is a text-file containing the registration errors of these combinations (e.g. mean, median, metric value, ...). These values can easily be imported into

Excel and visualized in order to find the best registration parameters. A template can be found on the CD in the folder `templates\templateToFindBestDemosParams.xls`.

It is recommended to mainly look at the standard deviation and the metric value because they represent best the quality of the registration. The mean value is not very representative because if there are huge errors in both direction (loss and growth), the mean would be around zero. However, with the standard deviation it is possible to see these huge errors.

Usage: `AppAutomatedTestDeformableRegistration [-args] image1.aim image2.aim output.txt`

Example: `AppAutomatedTestDeformableRegistration -l 2 image1.aim image2.aim output.txt`

This is an example of this application using a multi-resolution pyramid with 2 levels. In this application, the standard deviations and the number of iterations at each level cannot be chosen as these parameters are variable in the application.

Arguments:

- gaussian image smooth settings before registration
 - **-s arg1** : sigma [in voxel] (default = spacing)
- gaussian image smooth settings after registration
 - **-g** : gaussian smooth the images after registration (default = OFF)
 - **-i arg1** : sigma [in voxel] (default = spacing/2)
- deformable registration settings
 - **-l arg1** : number of levels (default = 1)
- SiteSpecificChangeFilter settings
 - **-t arg1** : set the threshold to create vtkPolyData (default = 30)
 - **-d arg1** : set the search depth [mm] (default = 0.25)

A.6 AppExtrapolation

An example can be found in the `vtkMy\dat-directory: Extrapolation.bat`

This application calculates an extrapolated image based on several input images or deformable fields. The extrapolated field can be written into three aim-files.

As a default, the following images are visualized in the case of input images (in the case of input deformable field, nothing can be visualized):

1. the the last image (red, transparent) with extrapolated image (white)
2. the second last image (red, transparent) with the last image (white)

Usage: AppExtrapolation -n .. [-args] image1.aim image2.aim ...
or
AppExtrapolation -f -n .. [-args] defField1.aim defField2.aim ...

Example: AppExtrapolation -n 4 -l 2 -i 3 5 -d 1 1 -s 2 2 2 -p 1 2 3 4 5 -t 6100
image1.aim image2.aim image3.aim image4.aim

This is an example of this application using a multi-resolution pyramid with 2 levels for the registrations and which takes 4 input images measured at equally distributed time points. The extrapolation time is at 'time point' 5.

Arguments: Settings if two or more **IMAGES** are inputted

- gaussian image smooth settings before registration
 - **-g arg1** : sigma [in voxel] (default = spacing)
- deformable registration settings
 - **-l arg1** : number of levels (default = 1)
 - **-i arg1...** : set number of iterations at each level (default = 4)
→ number of arguments = 'number of levels'.
 - **-d arg1...** : set the standard deviation for each level (default = 1.0)
→ number of arguments = 'number of levels'.
 - **-s arg1,2,3** : set the starting shrink factors (default = 1 1 1)
→ number of arguments = 'image dimension'.
- Extrapolation filter settings:
 - **-n arg1** : number of input images (default = 0)
 - **-p arg1...** : set the time points of these images and the extrapolation time (default = equal time steps).
→ number of arguments = 'number of input images'+1.
 - **-c filename** : compare extrapolated image with this one.
 - **-m arg1** : set the extrapolation method (default = 1) 1 linear extrapolation of the length/ direction of the last vector
2 linear extrapolate each component separately
3 lengths as 1 but in the direction of 2
4 3 dimensional linear extrapolation: $P = t * A + B$
5 3 dimensional quadratic extrapolation: $P = t^2 * A + t * B + C$
- gaussian image smooth settings after extrapolation
 - **-o 1,0** : gaussian smooth the images after registration (default = 0)
0 = do not filter; 1 = filter.
 - **-a arg1** : sigma [in voxel] (default = spacing/2)
- Output:
 - **-w filename** : write the extrapolated field to 3 aim-files (one for each component). Notice that you have to specify just one basic file name. The filter adds _X, _Y, _Z to this name.

- **-v 0,1** : visualize the reconstructed image (default = 1)
0 = do not visualize; 1 = visualize.
- **-h 0,1** : calculate the thickness (default = 1)
0 = do not calculate; 1 = calculate.
- SiteSpecificChangeFilter settings:
 - **-t arg1** : set the threshold to create vtkPolyData (default = 30)
 - **-r arg1** : set the search depth [mm] (default = 0.25)

Settings if one or more **DEFORMABLE FIELDS** are inputted

- Extrapolation filter settings:
 - **-f** : input images are deformable fields
 - **-n arg1** : number of input fields (default = 0)
 - **-p arg1...** : set the start and end time points of the def. fields and the extrapolation time (default = equal time steps).
→ number of arguments = 'number of input images'+2.
 - **-m arg1** : set the extrapolation method → see above (default = 1)
- Output:
 - **-w filename** : set the filename for the extrapolated field. The field is stored to 3 aim-files (one for each component) → see above. (default = extrapolation.aim)

Appendix B

Implemented Filters

All of the implementations are written in C++. As the classes and applications are implemented using VTK and/ or ITK, the following subsections contain VTK and ITK specific parts. For more information about VTK consider the references [1], [5] or [6], about ITK the references [3] or [4].

In this chapter we mention the functionality of the classes but not every single method. Consult the .h file of the classes for all methods and members.

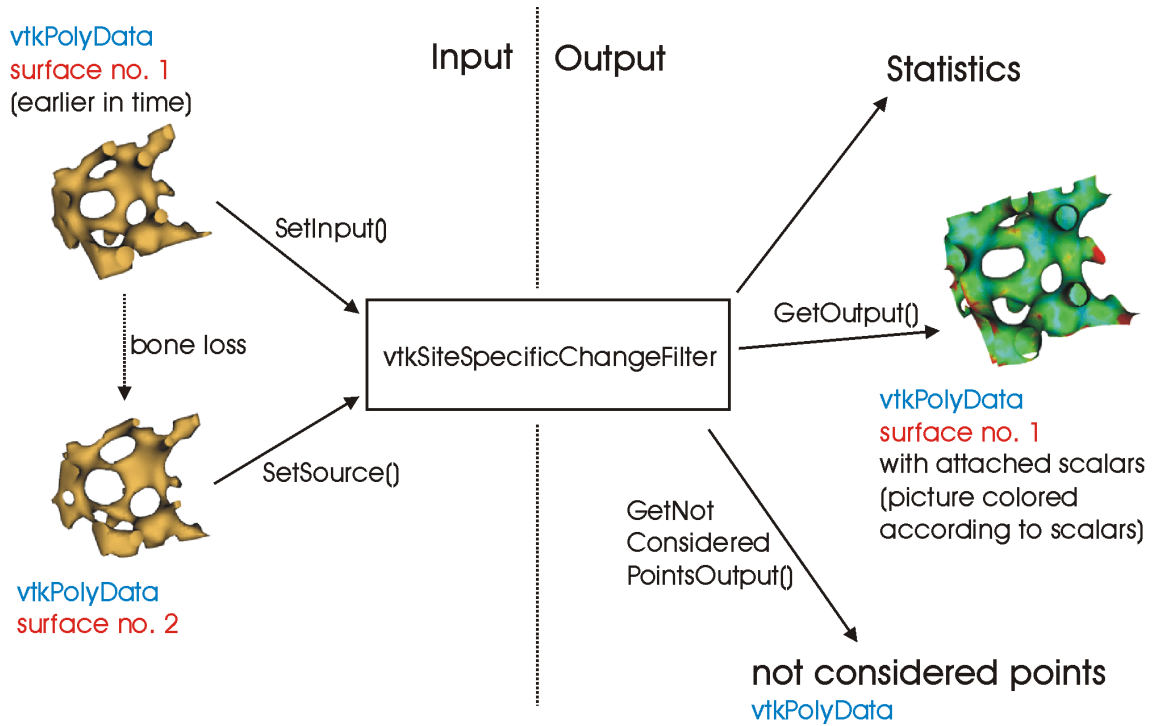
B.1 vtkSiteSpecificChangeFilter

We implemented the two methods discussed in chapter 5 (normals method section 5.2, closest point method section 5.3) into one VTK-filter with the possibility to change the method. This filter is derived from the class `vtkPolyDataToPolyDataFilter`. The name of the implemented filter-class is *vtkSiteSpecificChangeFilter*.

B.1.1 Input/ Output

As mentioned before the two methods are implemented into one VTK-filter. According to [1] the filter is an "attribute transformation". That means that we do not change the geometry or topology of the input data but only add new attribute data, namely the site-specific change. Point and cell data which are already attached to the input `vtkPolyData` are all passed to the output. The algorithm itself can be classified as "vector algorithm" again according to [1] because we use vectors between the two surfaces to calculate the site specific change. The input to the filter must be provided as two *vtkPolyData* surfaces. Surface no. 1 (earlier in time) is set by `myobject->SetInput(vtkPolyData *surface1)` and surface no. 2 by `myobject->SetSource(vtkPolyData *surface2)`.

The filter adds the attribute data (site-specific change) to the `vtkPolyData` passed by `SetInput` (surface no. 1) and gives it to the output of the filter. Surface no. 2 is not passed to the output. This behaviour is shown in figure B.1. Both methods attach scalars and vectors. The names and types of the attached data is given in table B.1.


 Figure B.1: inputs and outputs of the `vtkSiteSpecificChangeFilter`

Method	Attached Data	
	Type	Name
Normals	scalar	SiteSpecificChangeUsedNormals
	vector	NormalVectors
Closest Point	scalar	SiteSpecificChangeUsedClosestPoint
	vector	ClosestPointVectors

Table B.1: names and types of the data attached to surface no. 1

The name can later on be used to identify what method was used to calculate the site-specific change.

- Be aware when using the site-specific change vectors that you check if the according scalar value does not have the value `VTK_FLOAT_MIN`. The vectors themselves cannot be flagged and therefore you do not know if this vector had for example a wrong surface orientation (compare to 5.4.1).

Second output

The filter has the possibility to configure a second output. This output could be used to show the not considered points. With not considered points we mean points that did not find an intersection (normals method), points with no closest point (too short search depth) or points with the wrong surface orientation. This cases are colored red in the figures 5.7 and 5.10.

The function *GetNotConsideredPointsOutput()* returns the not considered points as *vtkPolyData*. You will get the *vtkPolyData* with a sphere (with radius *SphereRadiusNotConsideredPoints*) at every point which is not considered (see figure 5.24 for an example).

To activate the second output use the function *GenerateNotConsideredPointsOn()*.

B.1.2 How to visualize the result of the filter (site-specific change)

In subsection B.1.1 it was mentioned that *vtkSiteSpecificChangeFilter* returns *vtkPolyData* (surface data) with attached scalars in every point. The scalars are the calculated site-specific changes measured with the filter. Using small datasets we suggest to color the scalars on the surface according to their value with a lookup table.

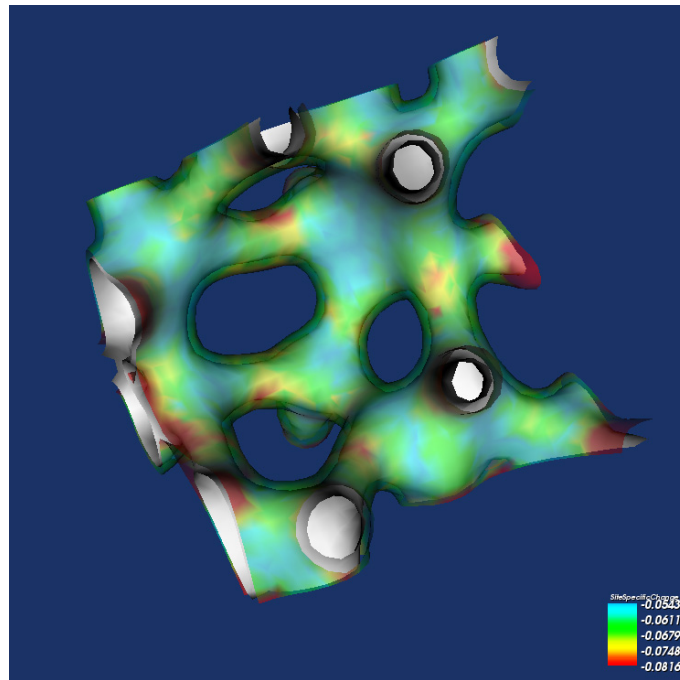


Figure B.2: example of the visualization of the site-specific change

Figure B.2 shows an example where red colors represent big changes and blue colors small changes according to the scalar bar in the figure. The site-specific change was measured between the outer surface (transparent) and the inner surface using the closest point method. The bigger surface in this picture is transparent and colored as described before. The smaller surface is colored in white to not change the colors of the transparent surface due to mixing effects.

It is also recommended to use the function *GetAdaptedColorRange()* which is included in *vtkUtilty* and described in section B.1.4 to define the range of the lookup table.

B.1.3 Settings of the filter

Because of two different methods being implemented in one filter also the settings differ in a couple of points. That is why we mention the settings in three parts. The first section is about settings which are the same for both methods. Afterwards the settings that differ for each method are noted (these functions or variables always contain *_Normals* or *_ClosestPoint* in the name).

If we talk about *myobject* in the following we mean an object of the class *vtkSiteSpecificChangeFilter*.

Shared settings

myobject->SetMethodTypeToUseNormals()

***myobject->SetMethodTypeToUseClosestPoint()* (default)**

These two functions set the method to either use normals or the closest points to calculate the site specific change. The two methods are described in section 5.2 and 5.3.

***myobject->SetSearchDepth(x)* (default=1)**

With this function you can change the length of the normals respectively the maximal distance of the search in the case of closest points (see also section 5.2.4 "search depth normals" and 5.3.4 "search depth closest point").

***myobject->PassNormalsOn()* (default)**

myobject->PassNormalsOff()

In VTK the datatype Polydata has normally a normal attached in every surface point. In the case of large surfaces these normals cause a big additional memory expense. Therefore there exists the possibility to not pass the normals from the input to the output of the filter.

Be aware that the surfaces will have a patchy outlook because the renderer uses the normals to smooth them.

myobject->GenerateNotConsideredPointsOn()

***myobject->GenerateNotConsideredPointsOff()* (default)**

***myobject->SphereRadiusNotConsideredPoints(x)* (default=0.01)**

There exists the possibility to return the not considered points as second output of the filter (see also B.1.1).

GenerateNotConsideredPointsOn generates the not considered points and activates the second output of the filter.

With the function *SphereRadiusNotConsideredPoints(x)* the radius of the spheres representing the not considered points can be specified.

***myobject->BoundaryTolerance(x)* (default=0.1)**

It was mentioned in 5.2.3 and 5.3.3 that points on the boundary are omitted in some cases. With this function the width of the boundary can be set adequately to the spacing and dimensions of the image.

myobject->AlwaysExcludeBoundaryOn

***myobject->AlwaysExcludeBoundaryOff*(default)**

If this setting is on, the boundary (*BoundaryTolerance(x)*) is always excluded. Normally only boundary points without closest point or without intersection are not considered.

myobject->DoCalculateStatisticsOn() (default)

myobject->DoCalculateStatisticsOff()

myobject->ShowHistogramOn()

myobject->ShowHistogramOn(lowerLimit, upperLimit)

myobject->ShowHistogramOff() (default)

If the statistics are activated there additionally is calculated the histogram of the site specific change. The histogram provides the mean, the standard deviation and the median of the site specific change. If ShowHistogram is also activated the histogram of the site specific change is displayed as xy-plot (use the overloaded function to show the histogram and additionally specify the range of the histogram). Otherwise only the calculated values (mean, ...) are displayed in the report.

Settings for the normals method

myobject->IntersectWithOwn_NormalsOn() (default)

myobject->IntersectWithOwn_NormalsOff()

It was described in 5.2.2 that the normals method intersects with the own surface (surface no. 1) if no intersection in positive or negative direction with the other surface (surface no. 2) was found. With this function you have the possibility to turn off intersections with the own surface.

myobject->GenerateVectors_NormalsOn() (default)

myobject->GenerateVectors_NormalsOff()

It is possible to not pass the normal vectors (site-specific change vectors) to the output to make memory consumption smaller. Be aware that we do not mean the normalized normals but the vectors from the starting point to the intersection point.

Settings for the closest point method

myobject->GenerateVectors_ClosestPointOn() (default)

myobject->GenerateVectors_ClosestPointOff()

As mentioned in subsection 5.3.5 it is possible to not pass the closest point vectors to the output to make memory consumption smaller.

B.1.4 Useful functions included in vtkUtility

There are two functions which are written to use in combination with *vtkSiteSpecificChangeFilter*. They were not included into the class itself because they can also be useful for other applications.

void vtkUtility::GetAdaptedColorRange(vtkPolyData *getColorRangeOfThisSurface, float adaptedRange[2], float rangeLimit) (default rangeLimit=0.1)

A common task while working with visualization is to color a surface according to scalars (the scalars are defined in every point of the surface). In our case the scalars are the site-specific change that can vary strongly over different data sets and therefore you have always to adapt the range of the lookup table. To make this easier this function has been implemented. The first argument has to be `vtkPolyData` with attached scalars. In our case you have to pass surface no. 1 with the attached scalars. The function will internally calculate the histogram of the scalar values. Figure B.3 contains a sketch of such a histogram. The function goes through the values in the histogram starting from the left and from the right side (see blue arrows in figure B.3) until it finds a value that is bigger than a certain limit. This limit (*rangeLimit*) is argument 3 of the function that can be specified (1 corresponds to the maximum in the histogram). The return values (*adaptedRange[2]*) of the function are two numbers which match the histogram in the described way. These calculated two limits are returned by the function in argument 2 as an array of size 2.

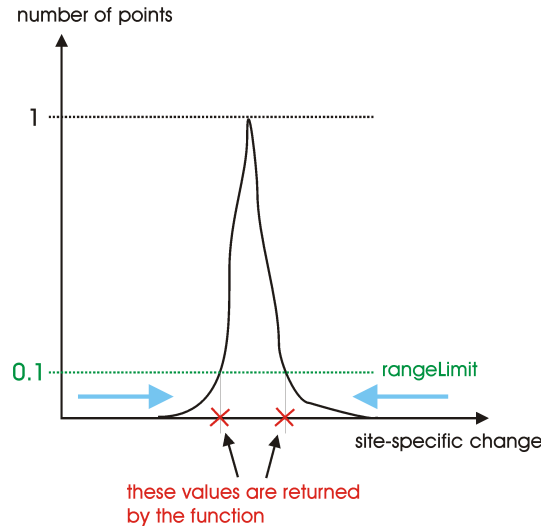


Figure B.3: histogram matching function

void vtkUtility::GenerateNormals(vtkPolyData *generateNormalsOfThisSurface, int maskRatioNormals, float searchDepth, vtkPolyData *generatedNormalsOutput)

This function gives help when adapting the search depth to specific data (see also 5.2.4 and 5.3.4 for search depth). It generates `vtkPolyData` containing all normals of the original `vtkPolyData` as lines (see figure B.4). The returned normals have the length *searchDepth* specified in the function. Using big datasets you can change *maskRatioNormals* to mask the normals on the surface (to show every normal use *maskRatioNormals*=1, if you just want to see every tenth normal set *maskRatioNormals*=10).

void MaskScalars(vtkFloatArray *inputArray, vtkFloatArray *outputArray)

The *outputArray* contains all scalars of *inputArray* without the numbers with the values `VTK_FLOAT_MIN` and `VTK_FLOAT_MAX`. The function is used to calculate the statistics without the excluded points.

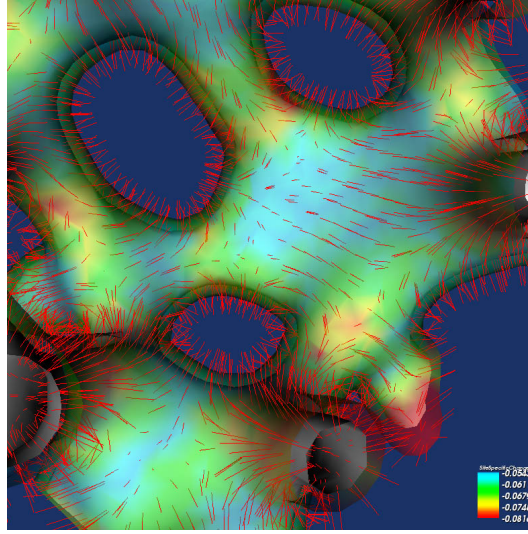


Figure B.4: picture with visualized normals to estimate the search depth

Method	Name of Attached Data
Thickness	
normals	ThicknessUsedNormals
spheres	ThicknessUsedSpheres
surface weighted spheres	ThicknessUsedSpheresSurfaceWeighted
Separation	
normals	SeparationUsedNormals
spheres	SeparationUsedSpheres
surface weighted spheres	SeparationUsedSpheresSurfaceWeighted

Table B.2: names and types of the data attached to the input surface

B.2 vtkSiteSpecificThicknessFilter

The algorithm mentioned in section 12.3 is implemented into a VTK-filter called *vtkSiteSpecificThicknessFilter*. It includes three methods (normals method, sphere method, surface weighted sphere method) to calculate the thickness or separation which are discussed in the following. The input to the filter is always *vtkPolyData* (surface data). The thicknesses/ separations are attached as scalars to the points in the cases of the *normals* and the *sphere method*. In the case of the *surface weighted sphere method*, the thicknesses are attached to the cells of the surface.

The Filter provides a second output (`GetMidPoints()`) which returns the center points of the spheres which were used to calculate the site-specific thicknesses. These points lie on the skeleton of the bone.

Normals method

In the first method, the normals of the surface are inverted and intersected with the other side of the surface. Many points are not defined using this method. Figure B.5 contains an example of the normals method. The colors represent the thickness at a certain position of the surface. White areas do not have a value assigned because no intersection was found. The histogram in figure B.5 also shows that the *normals method* is not a good choice to measure the thickness because the values vary in a too big range.

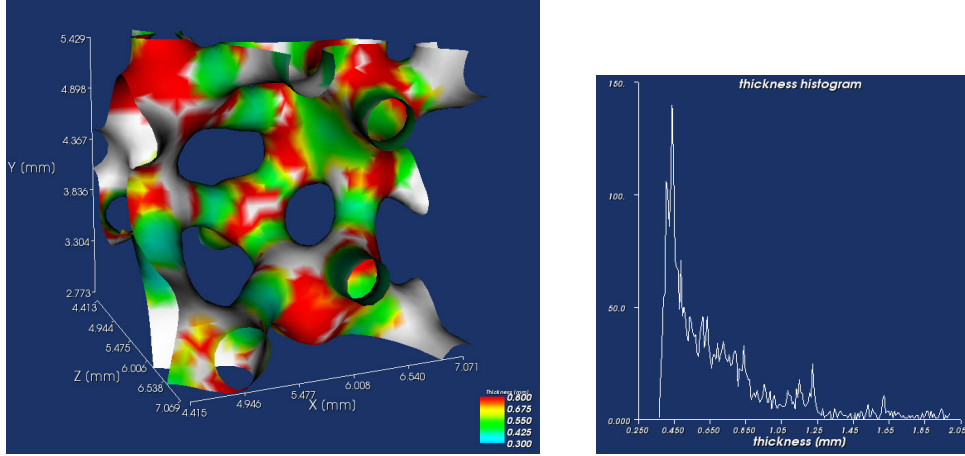


Figure B.5: Thickness measurement used normals method (Mean: 0.7056 mm) (the histogram is too broad with the normals method)

Sphere method

The second method incorporates the algorithm introduced in this chapter and uses the algorithm described in subsection 12.3. Figure B.6 shows an example where this algorithm was used. The surface is colored according to the thickness in every point.

Surface weighted sphere method

A third method was implemented to show that the *sphere method* is nearly surface weighted. This issue was discussed in section 12.4. Figure B.7 shows an example. The reason for the patchy outlook is due to the renderer which only interpolates the colors when point scalars are used. In this method however the scalars are attached to the cells.

B.2.1 Accuracy

The accuracy of the result can be affected by not letting the *position* converging far enough (see also 12.3). The *position* converges until a certain level (*endLevel*) is

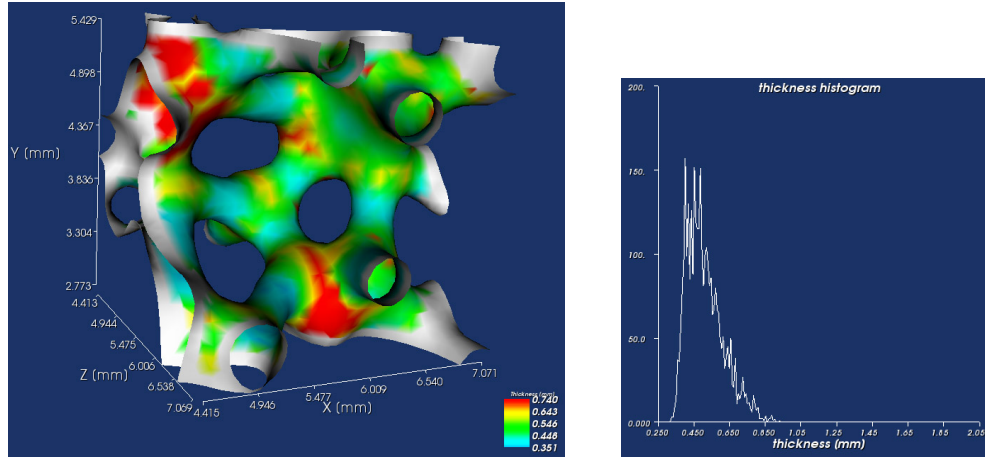


Figure B.6: Thickness measurement used spheres method (Mean: 0.51533 mm) (compare to surface weighted sphere method in figure B.7)

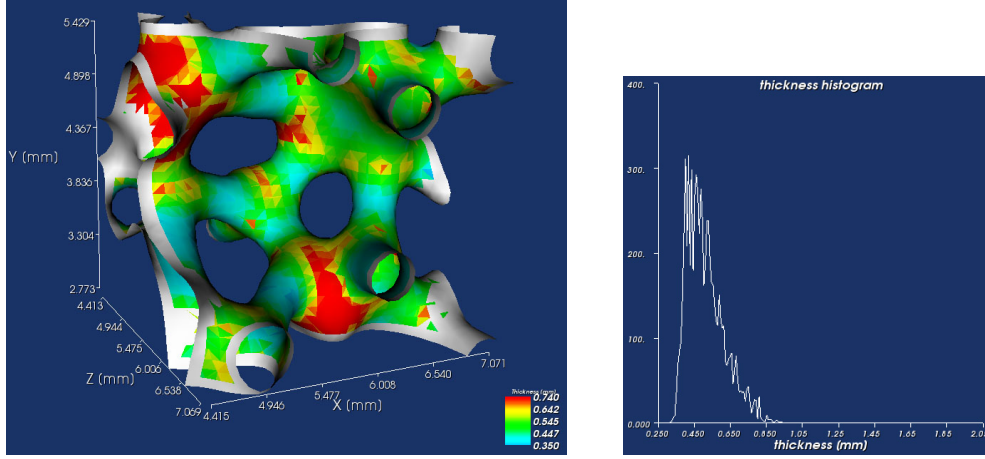


Figure B.7: Thickness measurement used surface weighted spheres method (Mean: 0.516426 mm) (compare to surface weighted sphere method in figure B.6)

reached.

The position is always changed by the amount: $change = 2^{level}[\mu m]$ (compare to figure 12.6). For example if the *endLevel* is set to 0, the accuracy is 1 micrometer. The level starts in each point with a big number (for example 7) and is decremented with each step.

B.2.2 Points with no thickness

Points on the surface where no thickness could be assigned or which lie on the boundary the value `VTK_FLOAT_MIN` is assigned to be able to identify them later on.

B.2.3 Settings of the filter

myobject->PassNormalsOn() (default)

myobject->PassNormalsOff()

See under [B.1.3](#).

myobject->SetMethodType(x) (default=1)

With this function the method of the filter can be changed. It uses either the *normals method* (0), the *sphere method* (1) or the *surface weighted sphere method* (2).

myobject->SetCalculateSeparationOn()

myobject->SetCalculateSeparationOff() (default)

If this option is turned on, the separation instead of the thickness is calculated using the method set with *SetMethodType*.

myobject->SetSearchDepth(x) (default=2 [mm])

This setting adjusts the length of the normals in the case of the *normals method*.

myobject->SetBoundaryTolerance(x) (default=0.1 [mm])

Boundary points are treated specially in the way described in subsection [12.6](#). This setting adapts the thickness of the boundary.

myobject->SetInitialOffsetSphereRadius(x) (default=0.005 [mm])

This length defines the initial *offsetSphereRadius* shown in figure [12.5](#). It is used as initial value for every point on the surface. If you want to calculate the thickness of a very rough surface you should choose a bigger number for this initial offset. Otherwise the computation slows down but the resulting thickness stays the same.

myobject->SetEndLevel(x) (default=0)

How the setting of the level affects the accuracy is described in subsection [B.2.1](#).

B.2.4 Returned values (report)

The following values are written as report into the "command prompt".

MeanThickness/ MeanSeparation

The issue about the mean is the weighting of the single values. The filter returns surface weighted mean values. The weighting is discussed in more detail in section [12.4](#).

StandardDeviation

The standard deviation tells you if the measured structure has everywhere the same thickness or if there are big differences in the thickness or separation.

MedianThickness/ MedianSeparation

The median is less sensitive to outliers than the mean.

NumberOfPointsWithAssignedThickness

This variable holds the number of points where the algorithm was able to assign a thickness value. It is also returned as percentage value to the total number of surface points.

NumberOfOmittedPointsOnBoundary

Points on the boundary are left out. This variable returns the number of points (and the percentage) omitted on the boundary.

NumberOfPointsWithNoDistanceAssigned

If it was not possible to assign a thickness to a point and this point is not located on the boundary it is added to *NumberOfPointsWithNoDistanceAssigned*. If this number is big compared to the total number of points either the boundary was set to small or there is another problem with the dataset. In such a case you should not trust the result.

ComputationTimeOfFilter

The computation time contains the time for the execution of the hole filter. Therefore passing the already attached point and cell data from the input to the output affects the time as well.

B.3 vtkReportPCA

The proceeding to calculate the ellipsoid using PCA mentioned in section 10.3 is implemented into the VTK-class called *vtkReportPCA*. The input to the filter is *vtkPointSet*. It must only include points and no cells. From these points the filter calculates the principal components and the variances in these PC-directions.

Make sure that an *Update()* has been called before fetching anything from the filter.

The site-specific change filter does not directly provide the point you have to input to *vtkReportPCA*. The points have to be extracted using the filter *vtkConvertAttachedVectorsToPolyData* which is described in the following section.

B.4 vtkConvertAttachedVectorsToPolyData

This filter is implemented to be the connector between the *vtkSiteSpecificFilter* and the *vtkReportPCA* classes. It is designed in a way that it is easy to be extended later on. The concept of this filter is to fetch the attached vectors from the input *vtkPolyData* and store them into a new *vtkPolyData* that is passed to the output.

In our case it was used in the way that only vectors were passed where the corresponding scalar was not `VTK_FLOAT_MIN` and either positive or negative. The reason for excluding `VTK_FLOAT_MIN` vectors is that these are points that were not calculated correctly (see for example 5.4.1). Positive and negative values are separated because we want to show one ellipsoid for the positive and one for the negative values.

The filter uses the active scalars and vectors of the input polydata.

B.4.1 Settings of the filter

```
myobject->SetOperationToPassPositive()  
myobject->SetOperationToPassNegative()
```

With these options you can choose if vectors with either positive or negative scalars are passed.

B.5 vtkScalarOperation

This filter is used to calculate the relative site-specific change by a division of the site-specific change and the thickness (see chapter 9). The name of the filter is *vtkScalarOperation*. The filter is designed in a way that it is easy to extend with other operations on scalars. The concept is to add a scalar to the input data set which is the result of an operation on the scalars of this data set. The scalars can be specified by their name.

B.5.1 Settings of the filter

```
myobject->SetOperationToDivisionFilter()  
myobject->SetOperationToDivision()  
myobject->SubtractionFilter()
```

With these options you can choose one of the operations on the scalars. The new scalars have either the name *DivisionFilter*, *Division* or *SubtractFilter* (*Filter* means that values VTK_FLOAT_MIN and VTK_FLOAT_MAX are excluded from the division).

```
myobject->SetName1(char *)  
myobject->SetName2(char *)
```

Use these functions to specify the name of the input scalars on which the operation shall be performed.

B.6 vtkDetectResorbedTrabecula

The procedure described in section 11.1 is implemented in the filter *vtkDetectResorbedTrabecula*. The inputs to the filter are two imagedata. As usual *SetInput()* must be used to pass the image earlier in time and *SetSource()* for the one later in time.

The number of resorbed trabeculas and new holes can be fetched with the function *GetNumberOfResorbedTrabeculas()* or the function *GetNumberOfResorbedTrabeculasPerCubicMillimeter()* which returns the density of the number of resorbed trabeculas and new holes. Make sure to call *Update()* prior to get these numbers. The function *GetNumberOfPointsPerConnectedRegion()* can be used to get the number of points per connected region as array with the length equal to the number of connected regions.

By using *GetOutput()* the filter returns the connected regions as polydata.

B.6.1 Settings of the filter

myobject->SetThreshold(x) (default=30)

This threshold is used to build the isosurfaces after the deformable registration.

myobject->SetAcceptanceAngle(x[2]) (default=[55, 125])

It was mentioned in section 11.1 that the angle between the vector from the point to surface no. 1 and the vector from the point to surface no. 2 is considered to distinguish between a trabecula that is very thin and one that actually disappeared.

B.7 itkDeformableRegistrationFilter

B.7.1 Input/ Output

The DeformableRegistrationFilter needs two inputs of the type itkImage, one of them taken earlier in time than the other one. It is very important that those two images are already registered non-rigidly, which means that they need to be positioned in exactly the same way. Poorly registered images will result wrong solutions!

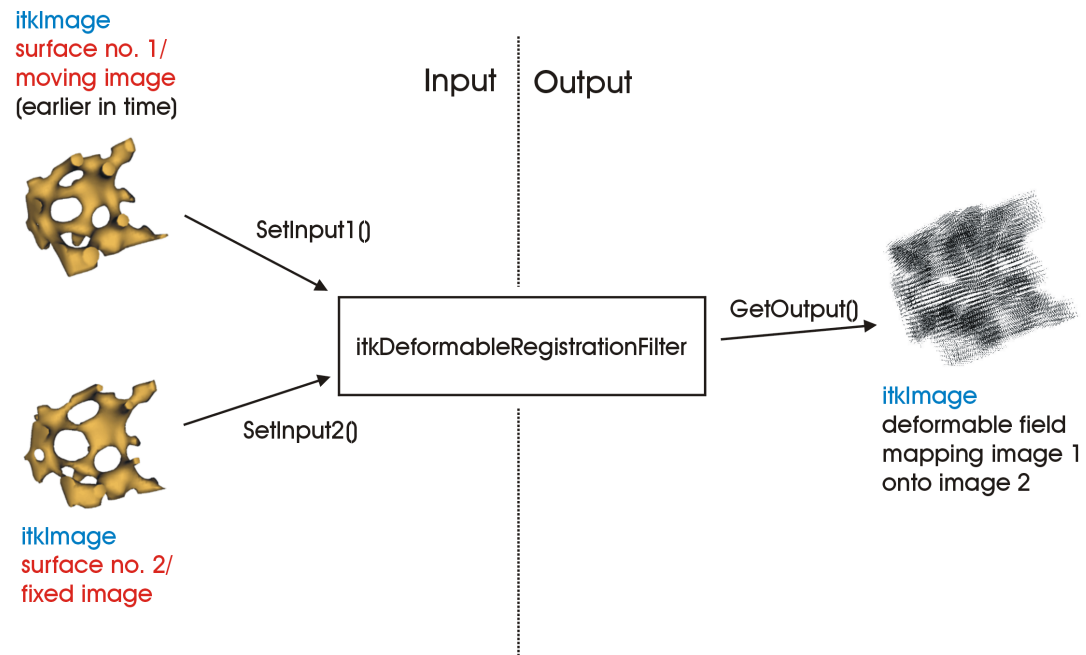


Figure B.8: This graphics shows very simple how to use the DeformableRegistrationFilter.

The two images are then registered with demons algorithm resulting in a deformable field mapping the image earlier in time (image 1) onto the image taken later (image 2). This registration can be done with one single resolution or with using a multi-resolution pyramid which means that the image-resolution is reduced for a first number of iterations. By doing so, a first result can be found faster. Thereafter, the image

resolution is risen and the resulting deformation field is set as the initial deformation field for the next level.

The output of the filter is a deformation field mapping the moving image onto the fixed image. It contains a vector of the size "image dimension" in each image point.

The filter is templated over the two input images (which have to be the same) and the output image.

In the following, it is shown how an instance of the filter can be defined. It is shown for a *short*-type input image and an output deformation field of type float:

```
#include "DeformableRegistrationFilter.h"
typedef itk::Image<short, 3> ImageType;
typedef itk::Vector<float, 3> VectorPixelType;
typedef itk::Image<VectorPixelType, Dimension> DeformationFieldType;
typedef itk::DeformableRegistrationFilter<ImageType, DeformationFieldType> DeformableRegistrationType;
DeformableRegistrationType::Pointer DeformableRegistration = DeformableRegistrationType::New();
```

As our filter is implemented in the itk environment (namespace itk) you have to define the type with `itk::DeformableRegistrationFilter`.

B.7.2 Settings of the filter

In the following section, `myobject` is used for an object of the class `itk::DeformableRegistrationFilter`.

`myobject->SetInput1(const InputImageType *image1)`

`myobject->SetInput2(const InputImageType *image2)`

With `SetInput1` and `SetInput2` the inputs of the filter can be chosen whereas input 1 is the one taken earlier in time.

Often when talking about registration the terms "fixed and moving image" are used with the meaning that the moving image is registered onto the fixed image. For our use this means that the moving image corresponds to the one taken earlier in time and would be input number one in our filter. We chose to use numbers for the inputs to correspond them to the time-axis and therefore to make it easier to remember.

Both input images have to be of the format `itkImage`. The filter has been tested for 3-dimensional char- and short-data but it should work with other data types as well, even with 2-dimensional data.

`myobject->SetNumberOfLevels(int NumberOfLevels)` (default = 1)

With this function the number of levels of the multi-resolution pyramid can be chosen (see section 6.2.3). Recommended is 1 level as with this choice it is simplest to get good results.

`myobject->SetStartingShrinkFactors(unsigned int *StartingShrinkFactors)` (default = [1, 1, 1])

With the starting shrink factor you can define by how much the image has to be reduced at the lowest level of the pyramid. Therefore, the input to this function needs to be an array of the same size as the *image dimension*. If you have a 3-dimensional image, the input to this function must be an array of 3 elements.

The shrink factors for the higher levels of the pyramid are internally calculated by dividing the shrink factors by 2. Thus, the starting shrink factors should always be a power of 2!

If you chose for example 3 levels and set the starting shrink factors to [2, 8, 4], the shrink factors at the next higher level will be 1 in the x-direction, 4 in y-direction and 2 in z-direction. At the highest level the image would be reduced by [1, 2, 1].

myobject->SetNumberOfIterations(unsigned int *NumberOfIterations)
(default = [5])

With SetNumberOfIterations the number of iterations at each level of the pyramid can be chosen. Therefore, the size of the input array has to be the same as the *number of levels*. For a single-resolution registration, 5 iterations seem to be a good choice for simulated data (for real data 10 to 20 iterations have to be used).

By using 2 different resolutions, the decision about the number of iterations is not so simple anymore. It seems that the level with the highest resolution needs at least 2 or 3 iterations to register the not quite perfect result from the lower level (for simulated data). For real data, at least 5 to 7 iterations are needed. A good choice for the number of iterations at the lower resolution seems to be 2.

More information can be found in chapter [6.4.9](#).

myobject->SetStandardDeviations(float *StandardDeviations) (default = [1.0])

The size of the input array has to correspond to the *number of levels* of the pyramid. An explanation what the standard deviation is needed for has already been given in section [6.2.4](#).

myobject->SetMaxRMSError(double MaxRMSError) (default = [0])

When performing an iteration, a temporary deformable field is calculated containing the changes which are added to the resulting field. The mean square value of these changes is saved in the variable "RMSChange". If RMSChange gets smaller than the variable MaxRMSError, the registration is stopped.

Therefore, this value can be used as a stop criterion to minimize the number of iterations. However, it has to be tested if the RMSChange is about the same for all good registered bones. For a first test we used 0.01 which seemed to be a good choice for the images we registered.

With GetRMSChange() and GetElapsedIterations() it can be checked which criterion (maximum number of iterations or MaxRMSError) made the filter stop.

myobject->GetMetric()

The metric value is the mean square difference in intensity between the fixed image and transforming moving image. If this value was zero, the registration would be perfect. Yet as this is never achieved, this value should be as small as possible.

B.8 `itkSymmetricForcesDemonsRegistrationFilter` and `itkSymmetricForcesDemonsRegistrationFunction`

This is a filter which was developed during this project and which we contributed to itk. As it is described in [6.2.1](#), the implemented method in itk had a problem with large changes. That's why we changed the existing filter (`itkDemonsRegistrationFilter` and `itkDemonsRegistrationFunction`) to using the symmetrical formula. The explanation of all the settings and how to use it can be found in the insight doxygen (http://www.itk.org/Insight/Doxygen/html/classitk_1_1SymmetricForcesDemonsRegistrationFilter.html).

B.9 `itkExtrapolateDeformationFieldFilter`

B.9.1 Input/ Output

This filter needs several deformable fields. Based on those fields, it calculates a prediction of how this field looks like in the future. If the output of this filter is applied to the last image (`itkWarpImageFilter`), a prediction of the structure of the bone is the result.

B.9.2 Settings of the filter

In the following section, `myobject` is used for an object of the class `itk::ExtrapolateDeformationFieldFilter`.

```
myobject->SetInput(int numberOfInput, const TInputImage *image,  
double startTime, double endTime)  
myobject->SetNumberOfInputs(int numberOfInputs)  
myobject->GetNumberOfInputs()
```

`SetInput` requires as a first parameter an input number (e.g. 0 for the first deformable field, 1 for the second one,...). **Note**, that the first field in time has to be inputted with number 0,...

Parameter 2 is a pointer to the deformable field.

Parameter 3 and 4 specify the time of the two images which are mapped onto each other by the deformable field (the time can be in any unit).

```
myobject->SetExtrapolationTime(double ExtrapolationTime) (default =  
0)  
myobject->GetExtrapolationTime()
```

Sets the point in time of the extrapolated image. This time has to be in the same units as the input times.

```
myobject->SetExtrapolationMethod(int ExtrapolationMethod) (default =  
1)
```

This sets the method how the future deformation field is calculated. Currently implemented are:

- 1: linear interpolation of the length; in the direction of the last vector
- 2: linear extrapolation of each component separately
- 3: direction: linear extrapolation of each component separately
length: linear extrapolation of the length (as 1)
- 4: 3 dimensional linear extrapolation: $P = t * A + B$
- 5: 3 dimensional quadratic extrapolation: $P = t^2 * A + t * B + C$

myobject->SetInversionNumberOfIterations(int InversionNumberOfIterations) (default = 10)

myobject->GetInversionNumberOfIterations()

In a first step, the deformable field can just be calculated as a forward field (where is a point mapped to). However, in itk the deformable field has to be a backward field (where does the point come from). Starting from the forward field, one has several possibilities to calculate an approximation for the backward field (as described in [13.2.4](#)):

A first method is to apply the calculated deformable field to itself. This is a rather imprecise approach which is very fast! To get the backward field with this approach, the InversionNumberOfIterations has to be set to zero!

Another method is to calculate an inversion of the forward field. This is more precise than the first method but also much slower. With InversionNumberOfIterations, the number of iterations - and with this the accuracy of the inversion - can be chosen!

myobject->SetStopValue(double StopValue) (default = 0)

myobject->GetStopValue()

This is a parameter of the itkIterativeInverseDeformationFieldImageFilter: If the error (in mm) between forward and backward mapping is smaller than the StopValue, the algorithm stops.

myobject->GetReport()

This returns a pointer to the report.

B.10 itkIterativeInverseDeformationFieldImageFilter

This is a filter which was developed during this project and which we contributed to itk. The basic idea behind this filter is described in section [13.2.4](#). The explanation of all the settings and how to use it can be found in the insight doxygen (http://www.itk.org/Insight/Doxygen/html/classitk_1_1IterativeInverseDeformationFieldImageFilter

Appendix C

Writing an itk-filter

C.1 Templates

ITK is set up to use templates. This makes the code very flexible and usable with almost any type of data. If you want to write an itk-filter, it is very likely that you have to use templates. This is handled like this: If the filter contains any templates, a txx-file is used instead of a cxx-file. This is principally the same as a cxx-file except that you have to include the txx-file in your h-file!

C.2 Necessary Functions

The filter executes when the *GenerateData()* method is invoked. Therefore, this is the "main-function" which has to be implemented. If multi-threading is desired, *ThreadedGenerateData()* would have to be implemented instead of *GenerateData()*.

Other functions which should be implemented are: a *Constructor*, a *Destructor* and a *PrintSelf()* method. These three functions are usually defined as "protected".

Also important are the shared typedefs like *Self*, *Superclass*, *Pointer* and *ConstPointer* which have to be defined in the h-file.

The following functions can be overridden if the input and output data do not have the same parameters (origin, spacing, region, ...):

- *GenerateOutputInformation()*
- *EnlargeOutputRequestedRegion(DataObject *object)*
- *GenerateOutputRequestedRegion(DataObject *object)*
- *GenerateInputRequestedRegion()*

For more information see chapter 13 "How To Write A Filter" in [\[3\]](#).

C.3 Set up the pipeline

One of the most difficult things to find out when you want to write your first own filter in itk is "how to get an output from the filter". This can be done in two different ways.

- If you just need to connect the pipeline to the output of the filter, the following code has to be used:
filter->GraftOutput(outputPtr);
filter->Update();
this->GraftOutput(filter->GetOutput());
- If you have to change your data and to copy it to the output, you need to use a image iterator (see chapter 11 in [3]).

C.4 How to enable syntax highlighting for txx-files in Visual C++

Visual C++ does not automatically color a txx-file like a cxx-file: All letters are just displayed in black. If you need to write or to make changes to a txx-file, this is quite troublesome. To color a txx-file in the same way as a cxx-file, the following instructions should be carried out [17]:

- Step 1. Run Regedit: Type regedit in the Run window or from a command line.
- Step 2. Find the tag HKEY_CURRENT_USER\Software\Microsoft\DevStudio\6.0\Text Editor\Tabs\Language Settings\C\C++\FileExtensions
- Step 3. Add ;txx to the end of the tag. So the Value data should look something like this: cpp;cxx;c;h;hxx;hpp;inl;tlh;tli;rc;rc2;txx

Note! that Visual C++ has to be closed when you change the registry (otherwise it will not change) and that also the txx-file in Visual C++ has to be closed!

A guidance for the .NET-Version can be found in [18]

Appendix D

Table of the used test data

During this project different kinds of algorithms and methods have been used for site-specific measurements. It is important to have an instrument for the evaluation of these methods. This chapter provides a list of the data we have been using during this project.

The data has been chosen in a way that it could be shown that the methods work for different types of bone (e.g. plate-like and rod-like).

The following subsection titles represent the directory name of the data sets and each of them contains some comments about the data.

/datasets/alu_15

This is an aluminum foam scan with SIBA applied in order to achieve the loss. REG-IST-ALU-POS_02_r02_IMA_THRESH-SUB_01_GFILT.aim is the thickest and REG-IST-ALU-POS_02_r02_IMA_THRESH-SUB_15_GFILT.aim the thinnest sample.

/datasets/dog

This is an canine bone scan with SIBA applied in order to achieve the loss. The bone is very dense. dog01_gauss.aim is the thickest and dog15_gauss.aim the thinnest sample.

/datasets/fix

This scan of aluminum foam has been scanned five times without touching the sample.

/datasets/invivo

This folder contains the registered in-vivo data received from Norvartis, Switzerland. The used procedure to register the data is written down in the file `/datasets/invivo/HowToRegister.txt`.

/datasets/MorphSamples

This morphological samples have been used in order to evaluate the thickness measurement.

/datasets/siba

This is an aluminum foam scan with SIBA applied in order to achieve the loss. `siba4_gfilt.aim` is the thickest and `siba0_gfilt.aim` the thinnest sample.