

Development of Intra-Modality Image Registration Methods for Assessment of Micro-Structural Bone Adaption

Semester Project

Summer 2003

Final Report

Authors: Michael Kuhn, Stephan Moser

Advisors: Prof. Dr. Ralph Müller, Dr. Steven Boyd

Calgary, September 12, 2003

Contents

1	Introduction	1
1.1	Documentation	2
2	Project Definition	3
2.1	Purpose	3
2.2	Project Goal	3
2.3	Project Tasks	3
2.4	Requirements of the tools	4
2.5	Data Sources	5
2.6	Further Topics	5
2.7	Documentation	5
2.8	Existing Software Packages	6
3	Image Registration Principles	7
3.1	Rigid Registration	8
3.2	Non-Rigid Registration	8
3.3	Inherent Problems of 3D Image Registration	9
4	Software Framework	11
4.1	Future Framework	11
4.2	External Software packages	11
4.2.1	ITK	11
4.2.2	VTK	12
4.3	Modules Developed	13
4.3.1	Overview	13
4.3.2	Dataset	13
4.3.3	Registration Module	13
4.3.4	Registration Controller	15
4.3.5	RegistrationControllerMultiResolution	15
4.3.6	Picker Module	16
4.4	Transforms and Unit Space	16

4.4.1	Preprocessing Operations	16
4.4.2	Transform Types	19
5	Evaluation of Registration Methods	21
5.1	Metric types	21
5.1.1	Mean Square Metric	21
5.1.2	Mutual Information Based Metrics	21
5.1.3	Normalized Correlation Metric	23
5.2	Interpolators	24
5.2.1	Nearest Neighbor Interpolation	24
5.2.2	Linear Interpolation	25
5.2.3	B-Spline Interpolation	25
5.3	Optimizer-Transform pairs	25
5.3.1	QuaternionRigidGradientDescentOptimizer and QuaternionRigidTransform	26
5.4	Validation Method	26
5.4.1	Target Registration Error (TRE)	27
5.4.2	Fiducial Mark Registration	27
5.4.3	AB - BA Registration	27
5.4.4	Artificially Transformed Image Data	27
5.5	Evaluated Configurations	29
5.5.1	Parameters Estimation for Mattes Mutual Information Metric	29
5.5.2	Accuracy and Performance Comparison	29
5.6	Memory Considerations	31
5.7	Conclusion	34
6	Further Topics	35
6.1	Registration - a problem for distributed computing?	35
6.1.1	Case study: Parallel execution on ETHZ Tardis cluster	35
6.1.2	Distributed Metric Calculation	38
6.2	Image Registration on Living Rats and Implications of an Application in Osteoporosis Studies	39
A	Software Manuals	41
A.1	AppITKRegistration - Single Resolution Registration	41
A.2	AppMultiResolutionRegistration	42
A.3	AppLandmarkPicker	43
A.4	AppAIMTransformer	43
A.5	AppRegistrationVisualizer	44
A.6	AppEvaluationTool	44

A.7 Config File Format	45
B Glossary	53
C Gallery	55

Chapter 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.

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 (μ CT) 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 μ CT has been limited to in vitro analysis, new in vivo systems are now emerging. To fully exploit the use of in vivo μ CT, appropriate quantitative analysis methods need to be developed that take full advantage of the sequential measures of bone micro-architecture.

One aspect of sequential μ CT measures is the need to align 3D images in space in order to compare bone structures. This process is called *image registration* and is a common task in computer vision and biomedical imaging. Due to its high computational requirements, registration of high resolution 3D image data has just recently become feasible. The goal of this project was to develop and evaluate methods for 3D image registration.

1.1 Documentation

This document is a report to the advisers of the project. It also serves as a working documentation for future development of the image registration software created in this project.

Chapter 2

Project Definition

This chapter presents the initial task definition as it was set up at the beginning of the project.

2.1 Purpose

The purpose of this project is to develop and validate methods for intra-modality image registration so that bone micro-architecture as measured from in vivo μ CT can be quantitatively assessed in a time-serial fashion. The image registration methodology will form a basis for future studies using a murine model for OP to investigate the preventative role of exercise and diet in juvenile mice on the long term development of OP. In these future studies, serial in vivo μ CT data will be obtained, and the image registration methodology developed in this project will be used to quantitatively determine site-specific bone adaptations over time.

2.2 Project Goal

The main goal is an evaluation and implementation of 3D image registration methodologies for time-sequential μ CT images. For this purpose, a set of computational tools will be developed using C++.

2.3 Project Tasks

- Set up a detailed time table.
- Review literature on image registration algorithms and in-vivo micro-architectural bone research (primary μ CT).

- Familiarize with AIM image data format, ITK, VTK.
- Develop a tool for interactive picking of landmarks on 3D isosurface rendering of images for initial input into registration algorithms.
- Perform intensity-based rigid registration using above acquired 3D landmark data as initial guess to determine the transform between two images.
- Evaluate different interpolation methods (nearest neighbor, linear, cubic splines), metrics, transformation types and optimization strategies for the registration process.
- Evaluate accuracy of the registration methods using fiducial marks in μ CT image data.
- Evaluate registration methods using mouse and rat bone μ CT data previously acquired.
- Determine sensitivity of registration to VOI (volume of interest) in target and source image.

2.4 Requirements of the tools

- The tools will run on state-of-the-art PC hardware.
- Memory efficiency shall be maximized (e.g. avoidance of data duplication in memory).
- The tools are to be designed with possible distributed computing techniques in mind.
- Command line interface driven, but potential integration with future GUI.
- Proprietary and patented software is to be avoided.
- Platform independence shall be maintained.

2.5 Data Sources

- Subsets of previously acquired μ CT data will be used initially. The registration methods will be tested by registering arbitrarily transformed data by known rigid transformations, and on sequential images of the same bone (in vitro).
- For accuracy evaluation, μ CT data of a mouse femur with fiducial marks is in preparation.
- To emulate future sequential in vivo data, methods to artificially age μ CT bone data will be considered.
- There is an option of getting actual sequential in vivo data from external sources, but this will only be exercised once the currently available data has been successfully registered.

2.6 Further Topics

- Discuss feasibility of performing image registration on live mice and rats.
- Discuss possible effects of osteoporotic bone changes in sequential μ CT images on registration methods (as per potential in vivo data set above).
- Discuss feasibility of distributed computing in image registration.
- Discuss the potential for a 3D visualization cave for use in registration methods.

2.7 Documentation

- Provide two intermediate and a final reports.
- Well documented source code for future use.
- Prepare a presentation to be held at the University of Calgary and ETH Zürich.

2.8 Existing Software Packages

- Insight Tool Kit (ITK), www.itk.org (open source; commercial use permitted).
- Visualization Toolkit (VTK), www.vtk.org (open source; commercial use permitted).

Chapter 3

Image Registration Principles

Image registration is a technique used to align related image data from different sources [5]. The image data might originate from a single type of image acquisition technique - so called *intra-modal registration* - or from different types in *inter-modal registration*. Registration can be done for 2D image data as well in the three-dimensional space.

There is a multitude of algorithms in use. Common to all though is an overall feedback structure:

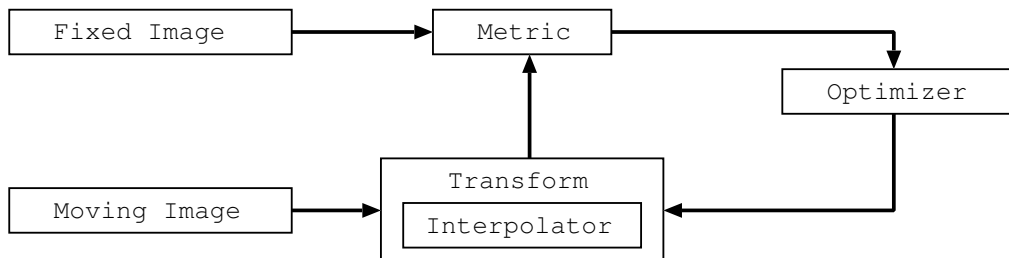


Figure 3.1: Registration feedback loop

There are two data sets as input to a registration process, called the *fixed image* and the *moving image*. The fixed image remains an unchanged reference data set while the moving image is transformed and interpolated for re-sampling in the image space. A *metric function* yields a similarity measure between the two data sets. The *optimizer* searches for a transform that optimizes the metric value, so the final output of a registration process is a transform.

3.1 Rigid Registration

Rigid registration restricts the search space to rigid transforms . A rigid transform is one that preserves angles as well as the Euclidean norm [7]. Thus, there are only translations and rotations allowed, but no scaling or skew operations.

Rigid registration is applied whenever the objects are inelastic and distortion or scaling caused by the data acquisition method can be neglected. Since in this case only rotational and translational components are allowed to appear in the resulting transform, the search space for the registration has to be restricted to rigid transforms. Allowing other transform components during the registration process does not only increase the computational complexity of the task, but might also generate incorrect results.

Rigid registration is basically an optimization problem over a search space of three parameters in 2D space (two translation coordinates and a rotation angle) or six parameters in 3D (three translation coordinates and three rotation components), corresponding to the *degrees of freedom* of the transform. Because of this restriction on the number of parameters, the computational complexity is much lower than for registration methods using more complex transforms, e.g. an affine transform which has 12 degrees of freedom in the 3D case.

However, even on this relatively small search space, the optimization problem can be difficult to solve. An initial guess can be provided to the optimizer to facilitate the task. Using this initial guess increases the chances of finding the correct local minimum (maximum) of the cost function and saves a lot of computing time.

3.2 Non-Rigid Registration

Whereas the transformation between the two image spaces is restricted to translations and rotations in the case of rigid registration, in *non-rigid registration* this transform can be arbitrarily complex. These transforms might be linear (or affine) as well as non-linear.

Non-Rigid registration is of interest for different kinds of applications. It might be applied if distortions occur during image acquisition. Further, registration tasks concerning deformable objects are usually based on *non-rigid transforms* . The deformable structures that have to be registered can originate from the same object as well as from different objects. Since the characteristics of these different applications vary greatly, many different approaches have been developed to solve the registration problem. However,

these approaches are not discussed here, since the registration of μ CT bone images is not a non-rigid task.

3.3 Inherent Problems of 3D Image Registration

3D medical images as acquired by a μ CT scanner are generally large. The image data used in this project were roughly $1000 * 500 * 500$ pixels in size; if each pixel is represented by a 2-byte value, this amounts to roughly 500MB of data. Even though this data can be massively compressed (e.g. ZIP), it must be held in memory in uncompressed state for efficient processing.

At the time of this writing, a state of the art computer has about 1GB of RAM. Processing two 500MB images on such a machine is extremely slow, since it calls for an extensive use of swapping.

Registration software is usually made up from an optimizer that maximizes or minimizes a joint metric value of the moving and the fixed image. This metric shows a certain gradient toward the optimum within the so-called *catch range*. To make sure registration already starts out within this catch range, human intervention is needed: An *initial guess* is required that is basically a rough estimate for the final solution. This guess is usually obtained manually, since automatic approaches are very time-consuming. Starting the registration from outside this catch range, the optimizer might get stuck in a local optimum instead of the globally correct solution.

Image data is usually represented by intensity values mapped to a pixel grid. During the registration, intensities from a transformed image have to be compared to intensities of a non transformed image. In general, the transformed pixel does not exactly map onto a grid position of the other image, which makes interpolation necessary. There is a trade-off between accuracy and performance regarding interpolation methods.

Determining the accuracy of a registration method is not a trivial task. Refer to section 5 for details.

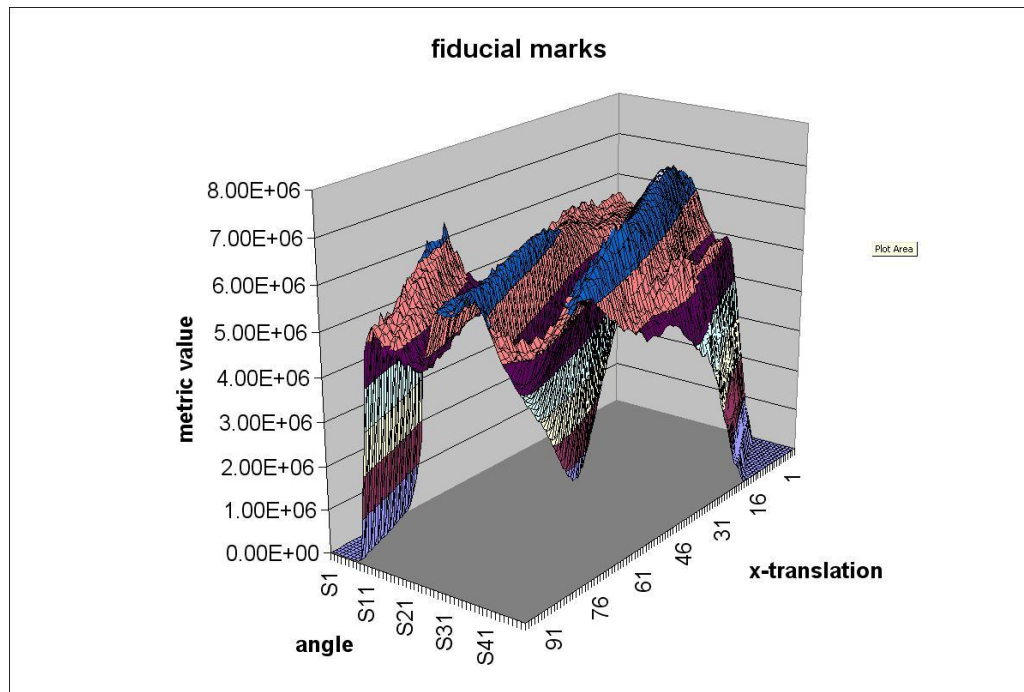


Figure 3.2: Alignment of corresponding structures

Chapter 4

Software Framework

The software modules created in this project are part of a software framework for medical image processing and finite element modeling developed at the University of Calgary.

The starting point was a set of VTK command line tools that performed different tasks on image data. The images are represented in *AIM image format*. This format represents the data in an uncompressed way, in either char or short values. The file also contains meta-information about the processing steps already performed on the data as well as about patient data. Data input/output is provided in form of an AIM reader and an AIM writer class written to be compatible with the VTK framework.

4.1 Future Framework

The registration functionality as well as the landmark picking will be integrated in an overall framework consisting of a GUI. Image processing will be organized in *streams* and the major data source will be AIM files. The framework will be extended with FEM methods in the near future.

Within this framework the landmark picking and registration components are mere intermediate preprocessing steps for further image analysis.

4.2 External Software packages

4.2.1 ITK

The *Insight Toolkit (ITK)* is an open source C++ library used for image registration and segmentation . ITK is funded by the National Library of Medicine and developed by six principal organizations (compare www.itk.org)

for more information). At the time of this writing, ITK is still being developed quickly and frequent bug reports and fixes make regular CVS checkouts necessary.

ITK offers a wide range of functionality in the area of image processing, registration and segmentation. There is an image processing pipeline present, which allows to connect numerous filters in order to apply them on the input data. To profit from the visualization capabilities of the VTK Toolkit, there are classes that allow to connect the ITK pipeline to the VTK pipeline and vice versa.

The ITK Toolkit makes intensive use of *generic programming* techniques. It is supposed to compile on any platform that supports C++. Further, most filters can be applied on a wide variety of image data. There are no restrictions on the number of dimensions or on the data type used for the representation of pixel intensities. As pointed out by the developers, one reason for generic programming is that it allows the programmer to discover e.g. type casting and rounding problems 'at compile time while enough coffee is available' instead of during an ongoing surgery.

However, the techniques (mainly C++ templates) used to make the classes as generic as they are, add a certain difficulty to understanding the ITK documentation and code. Further, a lot of type definitions have to be done to setup even a simple program. It takes quite some time to allow a developer to profit from the capabilities of the toolkit.

4.2.2 VTK

The Visualization Toolkit (VTK) [3] [6] is an open source package that is being maintained by Kitware, Inc. [1]. It consists of a set of C++ classes to visualize a wide variety of data. The package is kept platform-independent and uses existing visualization libraries like OpenGL or Mesa. It incorporates object-oriented principles and event-observer mechanisms. The newsgroup [4] is well frequented and the developers of VTK answer many postings themselves.

Within the project the package is mainly used for the landmark picking module and for visualization of AIM data files and registration processing steps.

4.3 Modules Developed

4.3.1 Overview

There are five major software modules present in the registration tool. These modules are a landmark picker module, a single resolution registration module, a single resolution registration controller, a multi-resolution registration module and a multi-resolution registration controller. Beside these module, there exists a data set class and several helper classes. The data set class is mainly responsible for the communication between the five main modules as well as for the storage of all relevant parameters and results of the registration. Figure 4.1 gives a general overview of the interplay of the mentioned classes which are described in more detail below.

4.3.2 Dataset

The dataset class is basically a collection of all the parameters that are used to set up a *registration run* as well as of all the results calculated by this registration run. It contains a `set()` and a `get()` method for all of these values, and allows to write them to and to read them from disk. The file produced by the dataset class is *human readable and editable*.

Whenever a module has calculated anything that is relevant for the registration process, it can write its output to the dataset. Succeeding modules are able to read this data from the dataset. If desired, the intermediate data can be stored on disk and be manipulated by the user.

4.3.3 Registration Module

To simplify the evaluation and execution of different registration methods, a registration module class has been developed. The main idea of this class is to make it easy to change the different components (transform, metric, interpolator and optimizer) used in registration. Besides these components, the module takes a fixed and a moving image as well as several parameters as input. Its output simply consists of a transform.

The registration takes care of the different template type names and instantiates the according components. Except for the input image types, it is therefore free from the sometimes cumbersome C++ templates.

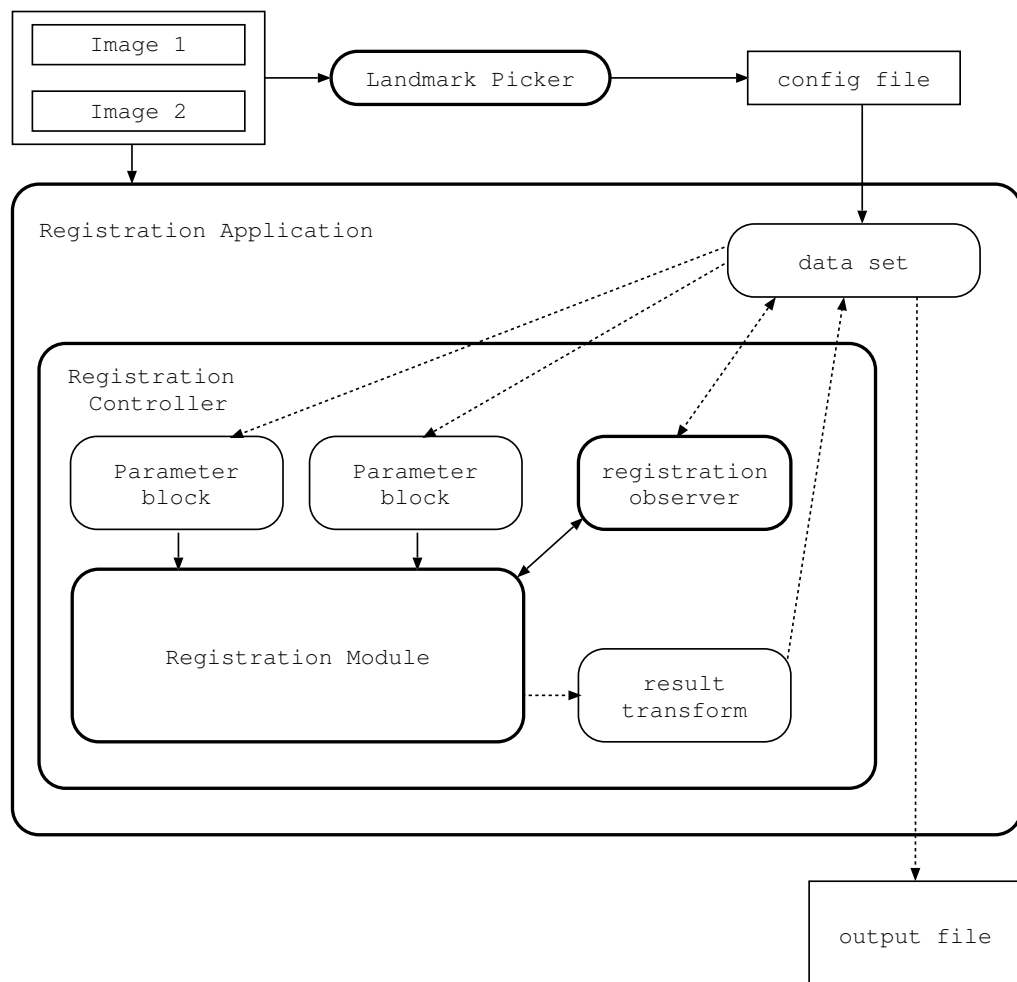


Figure 4.1: evaluation module diagram

4.3.4 Registration Controller

The registration controller controls the overall registration process. It is able to configure a registration module according to the information found in the data set. It assigns a transform, an optimizer, a metric and an interpolator. Further, it reads two AIM-images and pre-processes them (see section 4.4). The pre-processed images are then fed into the registration module, and all image related data is written to the data set.

To make the registration process observable, the controller adds a specialized observer to the registration module. This observer is capable of reading and writing a data set object which is needed to save a log file of the registration process.

The interface to the registration controller is free from any templates and mainly consists of a data set object. Since it might be desired to directly feed it with AIM data, instead of only providing the file names by means of the data set object, a direct interface for these data has been included.

4.3.5 RegistrationControllerMultiResolution

The `RegistrationControllerMultiResolution` class is derived from the `RegistrationController`. It extends the functionality of the `RegistrationController` by allowing to switch the resolution of the fixed and the moving image at run time. The resolution is defined by the shrink factor applied to the fixed and the moving image. Besides changing the resolution, the `RegistrationControllerMultiResolution` class also allows to change the metric and interpolator during an ongoing registration. The `AppMultiResolutionRegistration` application (see section A.2) instantiates the `RegistrationControllerMultiResolution` class instead of the `RegistrationController`.

The `RegistrationControllerMultiResolution` works closely together with a special optimizer that is derived from the quaternion optimizer. This optimizer calls the necessary methods as soon as the resolution level has to be changed.

Several problems have to be addressed when changing the resolution. First of all, the current position of the optimizer has to be adjusted to the new resolution. Further, the values of the optimizer's scale array have to change, and, in case another metric is used, the learning rate has to be adopted as well.

There are some multi resolution schemes present in the ITK toolkit. The main reason why it was decided not to use these classes was, that they would have required to use a floating point representation of the pixel intensities. The floating point data type that takes the least memory is a 'float'

type, but it still consumes twice as much memory as the short representation that is used throughout the registration applications implemented during this project. Because of the large amounts of data present in 3D registration it was considered to be worth doing the extra work of implementing an own multi resolution architecture to cut memory consumption down.

4.3.6 Picker Module

The landmark picker module implements landmark picking functionality. It is mainly used to visually determine initial guess transforms for registration. Due to its architecture it can be extended for other applications like the picking of numerical data from any rendered data model. The module implements an event-observer interface, which allows other modules to register for event notification.

The picker module has been wrapped into a command line tool which is able to read and write a data set object. Thus it can store its result on disk for further processing by the ITKRegistration application.

4.4 Transforms and Unit Space

4.4.1 Preprocessing Operations

Before any registration is done, several preprocessing steps are performed on the images. This includes 3D geometrical transforms to put the images into a state better suited for registration.

At the time of writing, the ITK toolkit still has troubles registering images with spacings not equal to 1.0. This makes it necessary to rescale the images, setting all spacings to 1.0. In addition, the images can be shrunk by a user-defined factor in order to reduce performance and memory requirements. Since optimizers usually work on linearized variables in an n-dimensional space, the absolute 'working point' matters. For registration it is favorable to have the images placed close to the center of the coordinate system. These steps lead to the following transforms:

$\mathbf{G}$ is the rigid initial guess transform obtained by the picker. It is defined as the transform from the landmark set of the fixed image into the landmark set of the moving image. This matrix is obtained from the picker tool.

For testing and evaluation purposes a pre-transform for the moving image has been introduced. $\mathbf{P}$ transforms the original moving image $\mathbf{M}_0$ to $\mathbf{M}_{pt}$. During normal registrations this transform is the identity. The registration application allows to turn the pre-transform on and off by a data set flag.

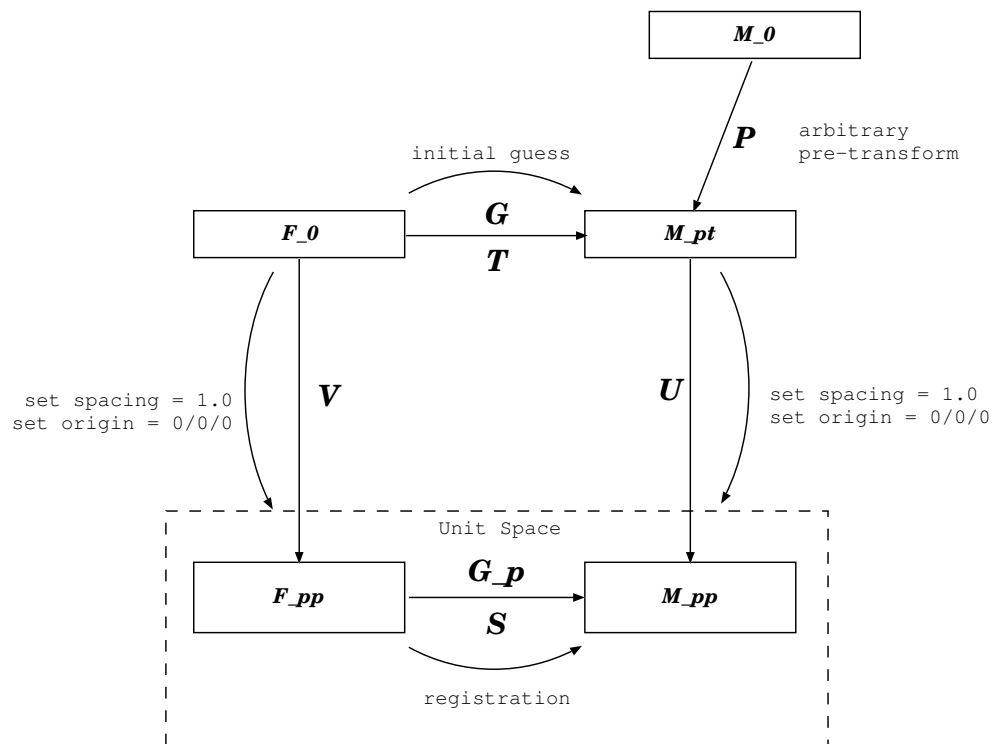


Figure 4.2: geometrical transforms for image pre-processing

$\mathbf{V}$ and $\mathbf{U}$ are transforms which take care of the dimensions and spacings of the input images. These transforms shrink or expand the images so that a unit spacing (1.0) in world coordinates results. The shrink factor β is calculated from

$$\beta = \frac{1}{\beta_{shrink} * \beta_{spacing}} \quad (4.1)$$

where β_{shrink} is a user-defined shrink factor to reduce memory and increase performance and $\beta_{spacing}$ is the minimum spacing found along the two times three coordinated axes of the input images. As data is re-sampled, the memory requirements are reduced drastically (e.g. for a shrink factor of 2.0, only $\frac{1}{8}$ of the original memory is needed).

$\mathbf{G}_p$ is the registration initial guess transform, derived from $\mathbf{G}$, adjusted to the unit space where the images are registered in. Since this transform has to be applied within the unit space, it is calculated from $\mathbf{U}$ and $\mathbf{V}$ as well (see figure 4.2)

$$\mathbf{G}_p = \mathbf{U} * \mathbf{G} * \mathbf{V}^{-1} \quad (4.2)$$

With transform matrices $\mathbf{U}$ and $\mathbf{V}$:

$$\mathbf{U} = \begin{pmatrix} \beta & 0 & 0 & -\beta * o_{ux} \\ 0 & \beta & 0 & -\beta * o_{uy} \\ 0 & 0 & \beta & -\beta * o_{uz} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.3)$$

$$\mathbf{V} = \begin{pmatrix} \beta & 0 & 0 & -\beta * o_{vx} \\ 0 & \beta & 0 & -\beta * o_{vy} \\ 0 & 0 & \beta & -\beta * o_{vz} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.4)$$

(o_u and o_v are the respective image origins in world coordinates).

The fixed image must be transformed by

$$F_{pp} = \mathbf{V} * F. \quad (4.5)$$

To get the pre-processed moving image, the following transform must be applied:

$$M_{pp} = \mathbf{U} * M_{pt}. \quad (4.6)$$

$\mathbf{S}$ is the registration result within the unit space. $\mathbf{T}$ is the final output transform in world coordinates. It is obtained from the registration result matrix $\mathbf{S}$ by

$$\mathbf{T} = \mathbf{U}^{-1} * \mathbf{S} * \mathbf{V} \quad (4.7)$$

Since within ITK all transforms are interpreted as being applied from the output to the input (e.g. the moving to the fixed image, see [2]), $\mathbf{V}^{-1}$ must be provided. To avoid numerical problems related to calculating the inverse, $\mathbf{V}^{-1}$ can be provided from scratch:

$$\mathbf{V}^{-1} = \begin{pmatrix} \frac{1}{\beta} & 0 & 0 & o_{vx} \\ 0 & \frac{1}{\beta} & 0 & o_{vy} \\ 0 & 0 & \frac{1}{\beta} & o_{vz} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.8)$$

The same holds for $\mathbf{U}^{-1}$.

Unlike $\mathbf{U}^{-1}$ and $\mathbf{V}^{-1}$, the matrix $\mathbf{P}^{-1}$ must be obtained from $\mathbf{P}$ by matrix inversion. The VNL library is used for this purpose.

4.4.2 Transform Types

Withing the registration software, 3 dimensional transforms are represented by different kinds of parameters. These representations are briefly explained below.

Matrix Representation of Homogeneous Coordinates

Homogeneous coordinates expand the dimension of the space by one. This allows to represent any affine transform in 3 dimensions as a linear operation in 4 dimensions, which can be described by a 4x4-matrix ([12]).

Basically, such a matrix contains scaling, skew and rotation operations in its top left 3x3 part and translations in the first 3 elements of the right most column. The bottom row represents projections which are not used within the registration software.

A rigid transform is a special case of an affine transform where neither scaling nor skew is present. Thus the top left part of the matrix turns out to be a normal 3x3 rotation matrix representing the rotational part of the transform.

Quaternion Parameters

Quaternion parameters are used by the ITK toolkit as the input to the quaternion rigid 3D transform. They consist of 7 values that represent a rotation (values 0 to 3) and a translation (values 4 to 6).

The translational part is straight forward, i.e. the values 4, 5 and 6 simply represent the translation along the x, y and z axis, respectively. The rotational part, on the other hand, is expressed by a rotation quaternion, which requires some further explanations.

A quaternion is a hyper complex number containing three imaginary and a real part:

$$q = r + xi + yj + zk$$

A unit length quaternion can be written as

$$q_n = \cos(\phi/2) + (n_x i + n_y j + n_z k) * \sin(\phi/2)$$

where $\vec{n} = [n_x, n_y, n_z]$ is a vector of unit length. This representation can be interpreted as a rotation by angle ϕ around $\vec{n}$. ([12])

The parameter representation takes the values $n_x * \sin(\phi/2)$, $n_y * \sin(\phi/2)$, $n_z * \sin(\phi/2)$ and $\cos(\phi/2)$ as values 0 to 3.

Human Displayable Parameters

Since from the representations described above, a person can not quite easily understand what the transform does, a third representation has been introduced, which is used whenever a 3 dimensional rigid transform is written to the screen or onto disk. These parameters are referred to as rigid 3D transform parameters. The *rigid 3D transform parameters* consist of 7 values, exactly as the quaternion parameters do. Values 0 to 2 still represent the axis of rotation, which does not need to be of unit length, however. Value 3 directly shows the angle ϕ and values 4 to 6 remain the translation component.

Chapter 5

Evaluation of Registration Methods

Apart from the development of the software, the registration methods have been evaluated for practical usability. The implemented metrics and interpolators have been analyzed for their registration accuracy, memory requirements and performance needs. This section describes the components, defines accuracy measurement methods and gives estimates on accuracy and parameters for lab use.

5.1 Metric types

5.1.1 Mean Square Metric

The MeanSquaresImageToImageMetric class within ITK implements a straightforward metric that calculates the differences between corresponding pixel intensities and sums up their squares. The obtained value is divided by the number of pixels for normalization. This metric has a smooth nature (see fig. 5.1) and a relatively large catch range.

5.1.2 Mutual Information Based Metrics

Determining a metric value using mutual information is a concept proposed by Viola and Wells [10]. It is based on the information content of the two images and works on probabilistic measures. The basic idea is to express a 'common information content' of the two images which the optimizer must maximize. A measure for information content is *entropy* as defined by

$$E(x) = - \int p(x) \log p(x) dx \quad (5.1)$$

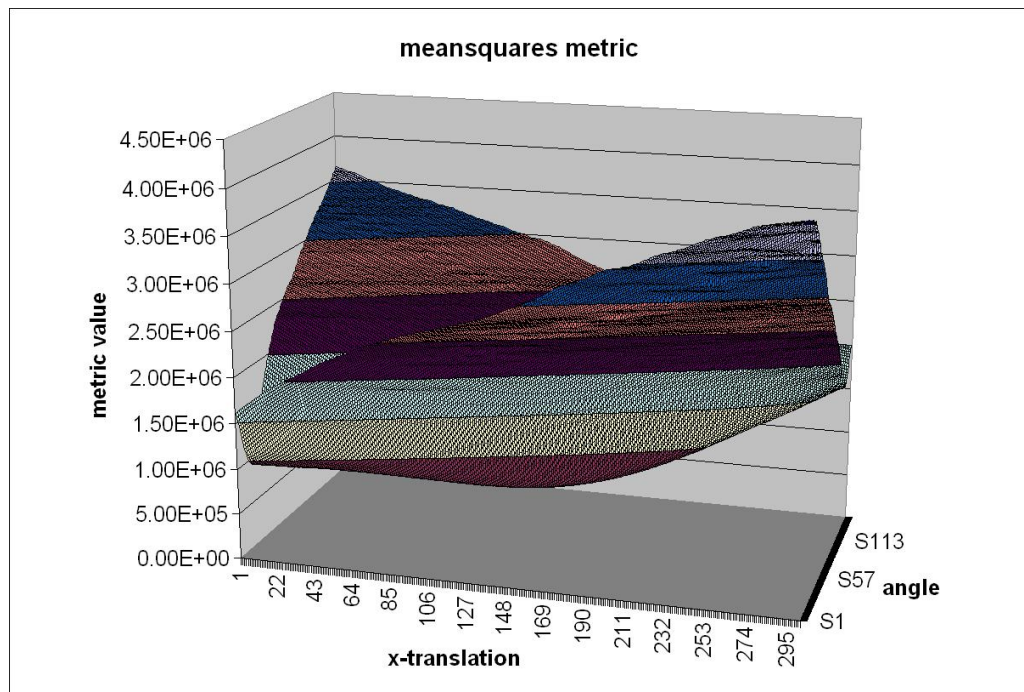


Figure 5.1: MeanSquares Metric for (x,y)-displaced bone images

where $p(x)$ is the probability density function of a random variable x . This measure can be extended to a common probability density function of two random variable by the *joint entropy*:

$$E(x, y) = - \int p(x, y) \log p(x, y) dx dy \quad (5.2)$$

Mutual information is defined as the differential entropy of two random variables. So,

$$I(x, y) = E(x) + E(y) - E(x, y). \quad (5.3)$$

The first two terms on the right contain the separate entropies of the two random variables. The third term holds the joint entropy which is greater the more correlated the two variables are. In image registration the random variables are the density distribution functions of the images, or the joint density distribution function respectively. The joint entropy of the fixed image F and the transformed moving image $M_t = T(M)$ is greater the better the images are aligned.

As a similarity measure function, mutual information is generally used for inter-modal registration since it is based on statistical image data rather than on absolute intensity values. It can be used in intra-modal registration though as will be shown later.

The *MattesMutualInformationMetric* is an ITK implementation of a mutual information metric. It is based on a paper by [11] and is a computationally efficient implementation since it uses only one set of samples from the image as opposed to the originally proposed method by [10] where a sample set for the calculation of the density and a second set for the calculation of the entropy is taken.

It takes two parameters: The *number of bins* describes a division of the intensity range into discrete bins for which the probability density functions are evaluated. The *number of spatial samples* says how many samples from the actual image data are taken to calculate the density functions.

5.1.3 Normalized Correlation Metric

The normalized correlation metric calculates a cross correlation function of two images and normalizes its value so that the metrics can be compared even though they have been calculated on partial images. The function is described as

$$f(F, M) = -1 * \frac{\sum_i F_i * M_i}{\sqrt{\sum_i F_i^2 * \sum_i M_i^2}} \quad (5.4)$$

with F and M as the fixed and moving images and M_i and F_i as its pixels.

Normalized Correlation is a well-known similarity measure in signal processing and versatile in its applications.

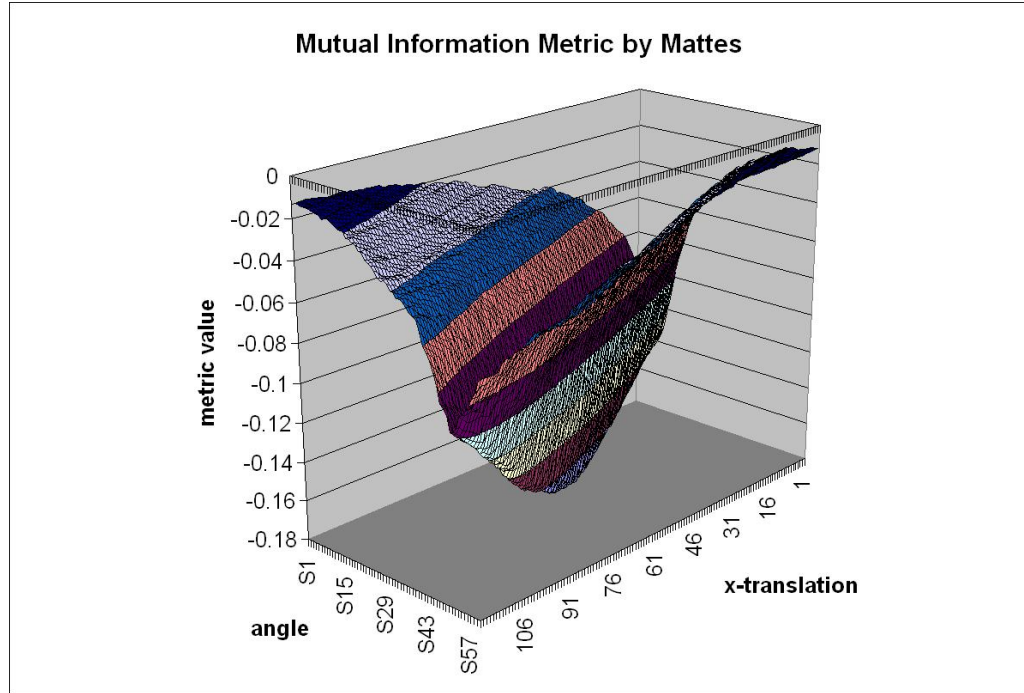


Figure 5.2: MattesMutualInformationMetric for (x,y)-displaced images

5.2 Interpolators

During the registration process, the fixed image is transformed and a metric is calculated by comparing pixel intensities between the fixed and the moving image. Since most likely a transformed pixel does not exactly lie on the pixel grid of the other image, interpolation has to be used to determine the intensity at that position. ITK offers three interpolation methods: a *nearest neighbor interpolation*, a *linear interpolation* and a *b-spline interpolation*, which are described in more detail below.

5.2.1 Nearest Neighbor Interpolation

The nearest neighbor interpolation simply assumes that the intensity at a certain position is equal to the intensity of the closest pixel. This method is very fast, since only index calculations have to be carried out. Nearest

neighbor is not very accurate though and only intended for rough registration or as a first quick approximation method.

5.2.2 Linear Interpolation

The linear interpolation considers the pixels directly neighboring the exact position. It then linearly interpolates these pixel intensities. This method is relatively slow but yields quite appropriate results. Since the trade-off between calculation time and accuracy is quite good, it is a reasonable option for image registration.

5.2.3 B-Spline Interpolation

The b-spline interpolation is the most accurate, but also the slowest one. It uses b-spline base functions to do the interpolation. The order of the base function can be chosen by the user, it is restricted to values of 0 up to 5. For order 0, the b-spline interpolation corresponds to the nearest neighbor, for order 1 to the linear interpolation. Higher order base functions have a larger support range and the interpolation becomes more accurate. The b-spline interpolation is considered to be a good approximation of the theoretically exact interpolation using *sinc* functions (see [5]). Hence it is generally expected to yield the most accurate results. This assumption could not be confirmed within this project, as pointed out later on in this chapter.

5.3 Optimizer-Transform pairs

Within ITK, the optimizer is closely related to the transform used for the registration. For the rigid 3D case, there are basically three transforms around. These are the *quaternion rigid transform*, the *Euler 3D transform* and the *versor rigid 3D transform*. All of them do a rotation and a translation in the 3D space. They only differ in the way they represent the transform, that is, the parameter space they operate on. This parameter space is the interface between the transform and the optimizer. Thus transform and optimizer should be considered as an interacting pair.

ITK only provides a suitable optimizer for the quaternion rigid transform. For the rigid 3D transforms, existing optimizers would have to be adapted. Most of the optimizers work on an algorithm that is based on moving along the gradient of the metric function. The influence of the optimizer-transform pair on the registration result is most likely quite small, since the main difference merely is the interface in between them. Therefore, it was considered

reasonable to only use the quaternion transform-optimizer pair and to do without the extra work of adapting optimizers for other transforms.

5.3.1 QuaternionRigidGradientDescentOptimizer and QuaternionRigidTransform

The quaternion transform uses seven parameters to represent the underlying transform (see section 4.4.2). The degree of freedom of a rigid 3D transform, however, is six. The extra parameter is thus redundant, but does not cause any problems.

The quaternion optimizer consequently operates on a seven-parameter search space. During each iteration it moves on by a vector pointing in the direction of the gradient of the metric function. The length of this vector is proportional to the gradient and to a specified learning rate. Some experiments showed, that the optimal value for that learning rate depends on the current position of the optimizer, i.e. how close it is to the correct result. Thus, a simple scheme has been introduced to adjust the learning rate during the optimization process. It basically checks the sign of the scalar product between the last two optimizer steps to see whether a major change in direction has occurred. If so, the learning rate is reduced, since most likely the correct extrema has been passed. If not, the learning rate is slowly increased, which allows to leave from local extrema after some iterations.

Since rotation and translation components of the transform are measured in different units, they cannot be treated the same during the optimization. Therefore, a scales array is passed to the optimizer, that weights each parameter. These scales depend on the image size, since the angle of rotation is independent, but the length of a translation vector is dependent on the image size. Therefore a scales array that is normalized to a unit cube was introduced in the configuration file for our registration runs. The application takes care of the necessary conversion between the array used in the config file and the one passed to the optimizer.

5.4 Validation Method

Evaluating registration methods is a tricky task since there is no goal data where the exact reference transform is known. There are several methods that still allow a determination of accuracy though.

5.4.1 Target Registration Error (TRE)

A useful measure for registration accuracy determination is the TRE (see [5] p. 120). It is a measure expressed in a real-world unit which expresses the average distance of two corresponding points from the fixed and moving images after registration. If the fixed image point is named f and the moving image point m then the measure amounts to

$$TRE = T(m) - f \quad (5.5)$$

where T is the output transform of the registration process. *Note that in order to calculate this measure, the theoretically exact solution transform has to be known.* The following sections describe methods that allow to make use of this measure even though no image material is available where the exact result transform is known.

5.4.2 Fiducial Mark Registration

The idea behind fiducial mark registration is to have reference data of objects with fiducial marks attached to it. The marks used in this project have been spheres of a higher density than bone. Segmentation by thresholding allows to create pairs of images where one image contains only the fiducial marks and the other one contains only the bone data. As the fiducial marks tend to have sharper contours than bone and tissue data, it is expected to register more accurately (see figure 5.3). A high resolution registration using the costliest method yields a transform that can be used as a reference for TRE calculations.

5.4.3 AB - BA Registration

Another way to evaluate a registration process by TRE is the AB-BA registration. Two images A and B are registered twice. One run is performed registering image A to image B and a second run goes vice-versa. Since $T_{AB}(A) = B$ and $T_{BA}(B) = A$ in the optimal case, $T_{AB}(T_{BA}(B)) = B$ and therefore $T_{AB} = (T_{BA})^{-1}$. So comparing T_{AB} with $(T_{BA})^{-1}$ allows an estimate of the accuracy of a registration process based on the assumption that the direction of registration has no effect on the accuracy of the registration process.

5.4.4 Artificially Transformed Image Data

In order to get images where the transform for optimal registration is known, one can apply a pre-transform on an image A : $A_p = P(A)$. This image

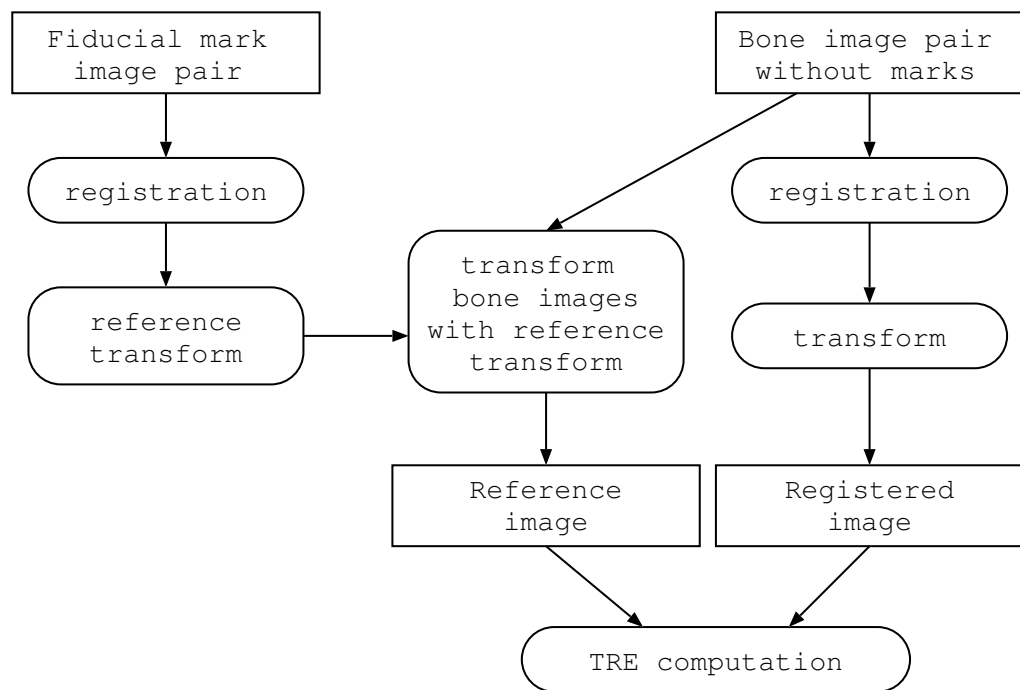


Figure 5.3: Evaluation metric definition

can now be registered to the original image A . The difference of the result transform and the pre-transform can be interpreted as an accuracy measure: $T_{dev} = P - T_{result}$. The problem with this approach is that a successful registration ends up aligning two images that differ only by the interpolation errors introduced by the pre-transform, which is different from the case with actual data that has undergone e.g. some changes in bone structure.

5.5 Evaluated Configurations

5.5.1 Parameters Estimation for Mattes Mutual Information Metric

The *MattesMutualInformationMetric* takes two statistical parameters: The *number of bins* and the *number of spatial samples* (see section 5.1.2). The outcome of a registration run depends strongly on a good estimate of these parameters.

As far as the *number of spatial samples* is concerned, the recommendation in the ITK software guide [2] states '1%..20% of the total number of pixels' as a rough estimate. Table 5.1 shows registration results with different numbers of samples set for the mutual information metric. The accuracy of the results rises significantly up to a number of roughly 15% of the image pixel number. For higher values, there is only a slight improvement, whereas the computation time increases linearly with the number of spatial samples.

The number of histogram bins is recommended to be around 50 according to the software guide. This figure has mainly been confirmed (see table 5.1). Note that this number has no direct influence on the overall calculation time.

5.5.2 Accuracy and Performance Comparison

Six combinations of metrics and interpolators have been considered with respect to accuracy and performance: Meansquares-, normalized correlation and the mutual information metrics have been combined with the linear and bspline interpolations. The runs have been performed on three different image pairs and evaluated using the fiducial registration method, the ABBA registration and artificially transformed images. Table 5.2 shows typical results. The images contained thresholded femur data. The original AIM file size was between 40M and 60M. Since the available memory did not allow a full-resolution registration, a shrink factor of 4 had been used as a first-level approximation using linear interpolation, and a shrink factor of 2 was applied for the respective configuration. In the table, note that the TRE values for

Parameter Evaluation MattesMutualInformationMetric			
Number of spatial samples (% of total number of pixels)	Fiducial TRE (mm)	Metric value (MSQ metric)	Computation time (s)
1	0.0483	428158	127
5	0.0460	413406	200
10	0.0392	366699	285
15	0.0385	364149	369
20	0.0388	367270	456
25	0.0380	366438	546
40	0.0383	364712	827
Number of bins	Fiducial TRE (mm)	Metric value (MSQ metric)	
10	0.0447	405945	
20	0.0378	359035	
30	0.0391	369207	
40	0.0393	371287	
50	0.0384	364149	
60	0.0399	373563	
70	0.0409	383285	
80	0.0401	373944	
90	0.0451	408814	
100	0.0453	412773	

Table 5.1: Number of spatial samples and number of bins evaluation

the bspline interpolated registration runs are generally larger than the linear ones.

Evaluation Results (typical values)				
Metric / Interpolation	Comp. time[s]	Fiducial TRE[mm]	ABBA TRE[mm]	art. Data TRE[mm]
MSQ / linear	1650	0.0119	0.0126	0.00361
MSQ / Bspline	2700	0.0152	0.0176	0.00428
NC / linear	2350	0.0115	0.0115	0.0191
NC / Bspline	4800	0.0148	0.0365	0.0242
Mattes / linear	700	0.0266	0.0337	0.00949
Mattes / Bspline	2400	0.0460	0.0813	0.00991

Table 5.2: TRE values for different metric/interpolator pairs

5.6 Memory Considerations

The μ CT images of mouse femurs used in this project have an approximate extent of 500x500x1000 pixels. Every pixel is represented by a short value, which results in a (uncompressed) data size of roughly 500MB. For one registration run, at least two of these images, or 1GB of data, need to be stored in memory. This is in about the amount of memory that a state-of-the-art desktop computer contains at the time of writing.

As described in section 4.4.1, the images go through several pre-processing steps before they are actually registered. During these steps, different intermediate images are created. Figure 5.4 gives an overview of these intermediate images. First, the AIM data stored on disk is read by the AIM reader, which propagates its output to the VTK-to-ITK interface. This interface converts it to an ITK image representation. The moving image is then optionally pre-transformed by a re-sample image filter. The images are shrunk, moved to origin (0, 0, 0) and converted to unit spacing by another re-sample filter. The output of this filter is finally fed to the registration module. For all of the re-sampling steps, b-spline interpolation is used.

To achieve memory efficiency, the main goal is to eliminate all intermediate images in memory as soon as they are not used anymore. This, however, is not quite easy to achieve within the ITK toolkit, because of different reasons:

ITK makes use of a pipeline mechanism to process data. This mechanism allows to connect several filters to each other, such that filter $n + 1$ processes the output from filter n and so on. Within this generic pipeline, there exists a trade-off between computing time and memory efficiency. The reason for this trade-off is, that if filter $n + 1$ wants to update its output, it needs the output of filter n . Filter n might already have calculated its output, so at a first glance, there is no need to recalculate it. That way, computing time is kept low. However, if the output is not recalculated, it has to be held in memory, which is not memory efficient.

Because of this trade off, ITK as well as VTK filters provide a ReleaseDataFlag. If this flag is set, the filter is allowed to free its memory, as soon as its output data has been processed by the following filter. Unfortunately, this mechanism does not work in every case. Within our project, we came across two situations when the ReleaseDataFlag could not be used as desired. It seems that the VTK-to-ITK interface does not correctly inform the preceding VTK filter as soon as its output has been processed. Therefore, the ReleaseDataFlag set on this filter does not cause it to release its data. The second situation was when b-spline interpolation was used. Since the b-spline interpolator internally makes use of the output of the preceding resample filter, this filter is not allowed to release its data. Unfortunately, the VTK-to-ITK interface and b-spline interpolation make up a major part of the pre-processing pipeline.

Since we expect that these problems related to the release data flag might vanish within later ITK versions, the effort of finding a well designed workaround was not made. Instead, a shrinking mechanism (as mentioned in 4.4.1) was introduced that allows to cut down memory requirements. However, the registration results become less accurate when shrunk images are used. Shrinking was inevitable though since it is impossible to work on the original image resolution, that demands for a multitude of 500MB images in memory. The main problem with these data sizes is, that as soon as the limit of the physical memory is exceeded, the operating system starts disk swapping and the calculation time literally explodes. Luckily, the same shrinking mechanism could be reused for multi resolution registration.

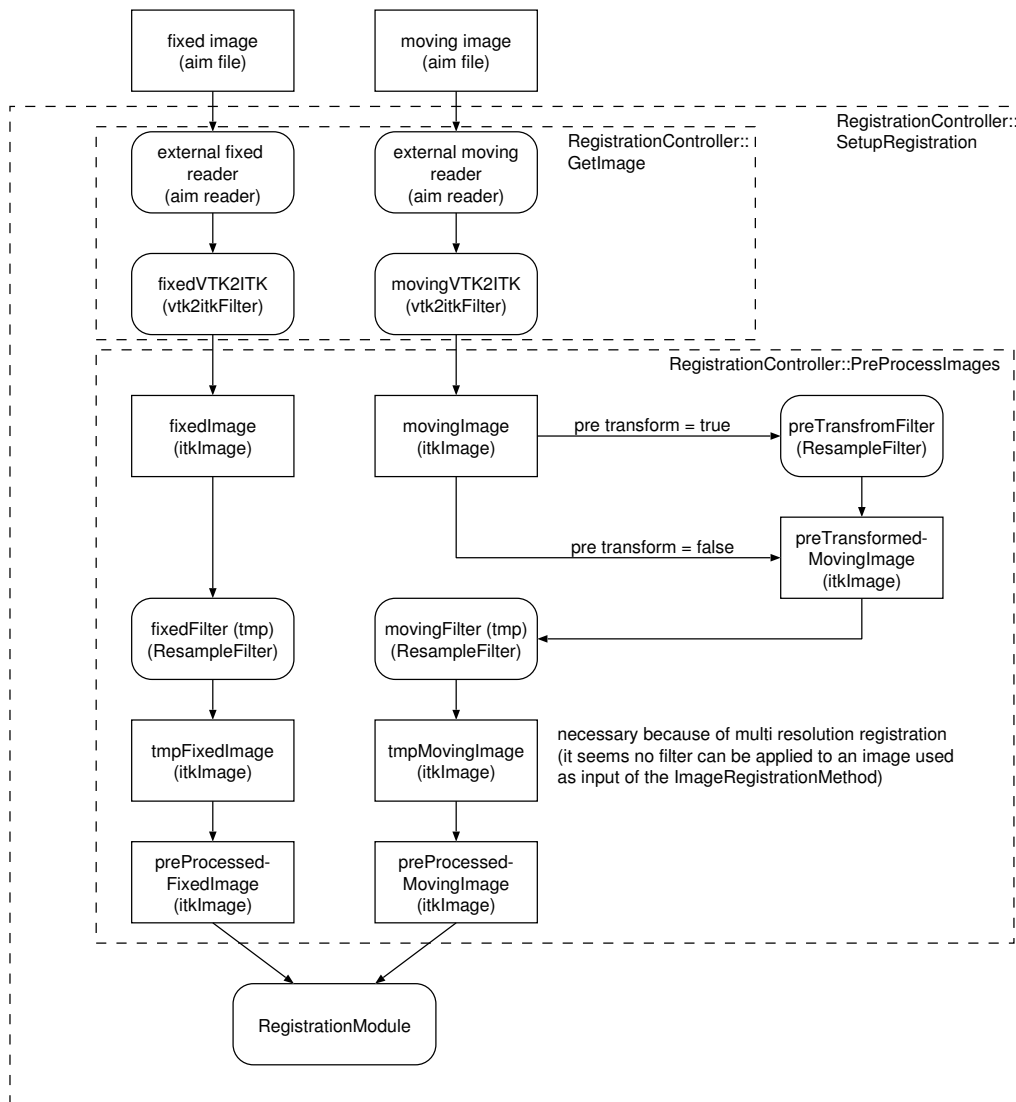


Figure 5.4: Memory Overview of the Registration Module

5.7 Conclusion

There are a few conclusions to be drawn from the above evaluation:

- The three evaluated metrics do not show fundamental differences with respect to an application for intra-modal image registration. They are relatively smooth and show a reasonable catch range and mainly differ in their calculation time.
- For the Mattes mutual information metric, reasonable values for the number of histogram bins are around 50. The number of spatial samples is best chosen between 10% and 20% of the overall number of image pixels.
- Since the mutual information metric has shown to be less accurate but much faster than the other metrics, it is recommended to use it for first approximations within multi resolution registration.
- For final refinement of the results, a meansquares metric or a normalized correlation metric in combination with a linear interpolation is proposed as they have shown to yield the most accurate results.
- Note that due to memory restrictions, a full-resolution registration on 40M or 60M sized images has not been performed. Since in this case no re-sampling during pre-processing would have to be done, a first interpolation step could be avoided and might lead to more accurate results.
- In contrast to the assumption that a bspline interpolation would yield better results in any case than any of the other interpolation methods, the most accurate results were obtained using a combination of a normalized correlation metric with linear interpolation. A reasonable explanation for this phenomenon could not be obtained.

Chapter 6

Further Topics

6.1 Registration - a problem for distributed computing?

6.1.1 Case study: Parallel execution on ETHZ Tardis cluster

If a series of registration runs have to be carried out, the task can be distributed on a networked cluster of computers. The ease of this approach lies in the untouched integrity of the registration application itself: The calculation is not multi-tasked, but many registration runs are performed in parallel. This is a method of reducing computational time that is used frequently in science and engineering.

For Unix and Windows NT clusters this job can be done using *Condor* [8]. The most evaluation runs for this project have been performed in this way. There were a few limitations on these runs however:

- The Tardis cluster at ETHZ contains SUN workstations with either 256MB RAM or 512 MB RAM. The smaller RAM computers were not able to cope with 60MB images that were downscaled with a shrink factor of less than 4.0. The 512MB workstations could handle larger data junks, but non of them was comparable to the lab desktop computer at the lab in Calgary, which had 1GB of RAM and 2GB of swap space. The Solaris OS would react with aborting jobs as soon as memory went short on the workstations.
- Condor runs in two modes: The *Standard Universe* and the *Vanilla Universe*.

In order to use the Standard Universe, the application would have to be compiled with a tool called *condor_compile*. This is a compile tool that adds certain interprocess communication functionality to the application. This allows Condor to do *job migration*, meaning that a process could be removed from a machine that e.g. gets busy due to user interaction. The job would simply be moved to another machine and continue running.

The Vanilla Universe on the other hand takes pre-compiled binaries and sends them to other Condor workstations. The disadvantage is that if a job has to be interrupted, the process state would be lost and the application would have to be restarted on another machine. The Tardis Condor setup was not configured for the Standard Universe functionality, so Vanilla Universe had to be used.

- Since the user accounts for students are usually restricted in storage size (100MB up to 400MB), image data could not be stored on there. This restricted Condor in the use of the network file system (NFS). Image data was stored on a local scratch partition on Tardis-a22.ethz.ch. The image data had to be sent to each host running a Condor client for each job, creating considerable network traffic and making the config file setup difficult, since paths had to be adjusted to be relative to the local Condor temporary directory.

Condor on Tardis: An Example

There are three important commands out of a set of many tools related to Condor to start a cluster of jobs:

`condor_q` prints out a list of all jobs currently queued for execution. The jobs are listed with their status, execution time, memory consumption etc.

`condor_rm` allows to remove single jobs or entire clusters of jobs. The jobs will be marked for removal and eventually be stopped and removed from the peers.

`condor_submit` is the command to start a cluster of jobs. It takes a configuration file that contains all the information about the jobs to run. An example is shown below:

```
universe = vanilla
transfer_files = ALWAYS
my_prefix = Series_CatchRange_
rank = memory
coresize = 1000
```

```

executable=/home/stepmose/calgary/bin/AppITKRegistration
my_run_number=x2.00
app_config      = \$(my_run_number).config
app_out         = \$(my_prefix)\$(my_run_number).app_outfile
app_log         = \$(my_prefix)\$(my_run_number).app_log
log             = \$(my_prefix)\$(my_run_number).condor_log
output          = \$(my_prefix)\$(my_run_number).stdout
error           = \$(my_prefix)\$(my_run_number).err
fixed           = /scratch/regist_sw_01_fe_pos_01_r02.aim
moving          = /scratch/regist_sw_01_fe_pos_02_r02.aim
transfer_input_files = \$(fixed) \$(moving) \$(app_config)
arguments       = -config \$(app_config) -out \$(app_out)
                -log \$(app_log) -visualize none
queue 1

```

```

my_run_number=x2.10
app_config      = \$(my_run_number)
app_out         = \$(my_prefix)\$(my_run_number).app_outfile
app_log         = \$(my_prefix)\$(my_run_number).app_log
log             = \$(my_prefix)\$(my_run_number).condor_log
output          = \$(my_prefix)\$(my_run_number).stdout
error           = \$(my_prefix)\$(my_run_number).err
fixed           = /scratch/regist_sw_01_fe_pos_01_r02.aim
moving          = /scratch/regist_sw_01_fe_pos_02_r02.aim
transfer_input_files = \$(fixed) \$(moving) \$(app_config)
arguments       = -config \$(app_config) -out \$(app_out)
                -log \$(app_log) -visualize none
queue 1

```

...

- **universe** is either **VANILLA** (pre-compiled binaries, no job migration is possible) or **STANDARD** where all the sources are compiled using the **condor_compile** command that includes IPC (interprocess communication) functionalities. Within the standard universe, jobs can be imaged completely including the overall process state and be moved to another peer for further processing.
- **transfer_files** tells condor to send all files to the peers. The alternative would be to use the NFS. Student user accounts are too small though to store AIM files, so they are copied from the 'scratch' partition of the condor master.

- All tags starting with `my` are user-defined variables.
- `rank` tells condor to go for the peers with the largest memories.
- `transfer_input_files` is a list of files that are transferred to the peers before job execution. They are stored in a local directory where the job itself runs in.
- `arguments` is the command line to the executable previously defined.
- `queue 1` puts one process into the local queue of the condor master. The process will be scheduled as soon as resources allow it. The command refers to the previously set variables.

Note that the configuration file is basically a state machine with tags getting defined and keeping their values until they are changed again. The output of the processes are all together redirected toward the master machine, so if an output file is opened by the application, it is in effect opened on the master machine. The same applies for the stdout which is output into a file on the master.

6.1.2 Distributed Metric Calculation

Since three dimensional image registration requires a lot of computing resources, it is worthwhile to consider the feasibility of using distributed computing methods. Both, computing time and memory consumption are an issue in image registration. Using distributed computing, either problem can be addressed.

Unfortunately, there exist many data dependencies that have to be dealt with. Image registration is basically an iterative process, and during each iteration the same computations are carried out. However, since each iteration depends on the results of the preceding one, it is not possible to take advantage of these similarities. Therefore it is necessary to think of a way to distribute the calculations that make up a single iteration.

During one iteration, the derivatives of a metric function as well as its value at a certain position have to be computed. Depending on the results the new position has to be defined. Most metrics are based on a pixel-wise comparison and are therefore well suited to work on partial images. Thus, a metric value for the whole image region can be computed from the metric values of multiple subregions. The same holds for the derivatives. Consequently, it is possible to distribute the calculations needed during one iteration step, if the underlying metric allows to have its value and derivatives

being computed based on partial images (which is, for example, possible for a mean squares metric).

One strategy for distributing the registration process thus is to distribute partial images for calculation on a cluster of computers. However, it has to be carefully analyzed which data are needed by each computer to carry out its calculations. Unfortunately, it is not sufficient to simply use the same region of the moving and the fixed image, because of the different relative positions the images pass through in the course of the optimization. Therefore, either the corresponding regions have to be calculated for every iteration, or the complete fixed image has to be stored on each machine. Since in the first case, new image data is presented to every computer during each iteration, this approach is not quite optimal with respect to the network traffic. Therefore, the most reasonable solution would probably be to store the whole fixed image (or at least a region that is large enough that it will never be left by the current position of the optimizer) on all computers.

This strategy would roughly speed up the registration by a factor n when n computers are doing the distributed metric calculation. Since memory can only be saved for the moving image, this strategy is less efficient with respect to the memory consumption. Here, a factor of roughly $\frac{2}{1+\frac{1}{n}}$ can be saved per machine. During the actual registration, only the new position would have to be transmitted to the participating computers for every iteration. The initial network traffic would be considerable though. Thus network traffic is kept at a minimum during the ongoing optimization process.

6.2 Image Registration on Living Rats and Implications of an Application in Osteoporosis Studies

The goal of this project was to provide image registration methods that allow the study of osteoporosis from a series of images taken from living rats. This application has some impacts on the techniques described throughout this document.

Taking μ CT images from living rats might cause some problems during the image acquisition. First of all, the radiation dose applied while scanning bones of living rats cannot be as high as for the studies on death bones. Therefore, the data might be slightly more vulnerable to noise. Further, even though the animal can be narcotized and fixed within the scanner, it is likely to move due to heart activity and respiration. Since the acquisition of a μ CT scan takes from several minutes up to hours, the images might get

additional distortions.

Another impact on the registration is caused by the analyzed data itself. Osteoporotic bone changes its structure over time. It is difficult to estimate the influence of these changes on the registration methods. The character of the structural adaption caused by osteoporosis is not known very well. If the bone mass is, for example, reduced in an asymmetric manner (i.e. bone mass is for example only reduced at the top, but not at the bottom of the inner structures), then registration results might be affected by these asymmetries.

The accuracy of registration with in-vivo data showing asymmetries will be determined by the type of metric being used. Of the three mentioned metrics, the mutual information metric is suspected to be the choice for future in-vivo registration, as it is widely in use for intra-modal registration tasks.

Appendix A

Software Manuals

A.1 AppITKRegistration - Single Resolution Registration

Note: AppITKRegistration is the predecessor of AppMultiResolutionRegistration. The multi resolution application extends the functionality of the AppITKRegistration and can therefore be used for single resolution registration as well.

For the evaluation task, the registration controller has been wrapped by the *ITKRegistration application*, to allow automatic execution of registration runs. ITKRegistration reads in a config file that contains all data necessary to complete one registration calculation.

All calculated data is written out to a log file for further processing. These log files are intended to serve as extensive lab reports. Post processing of these files allows to determine e.g. the accuracy of a certain registration method even when the accuracy measure function changes during the course of the project.

Besides its functionality to write a report file, the command line wrapper allows to visualize intermediate steps during data processing and to calculate some basic evaluation results of the registration.

Parameters:

-*config*: name of the configuration file (see section A.7)

-*out*: name of the output file (which looks basically the same as the configuration file, but contains some output parameters calculated during the registration run)

-*log*: name of the log file. The log file keeps track of the optimizer steps.

-*visualize*: decides which images should be displayed on the screen. There is always the moving (red) and the fixed image (green) displayed in the

same window. The options are *all*, *end*, and *none*. '*all*' displays (almost) all intermediate images (the original images, the images with the applied initial guess, the pre processed images and the final result of the registration). '*end*' only displays the final result and '*none*' doesn't display any images.

-factor: defines a shrink factor used for visualization. This factor has been introduced for performance reasons.

-eval: tells whether the metric values for all three metrics should be calculated to evaluate the resulting transform found during the registration process. Possible values are *true* and *false*.

A.2 AppMultiResolutionRegistration

The multi resolution registration application extends AppITKRegistration by a *resolution pyramid*. It is a command line wrapper of the MultiResolutionRegistrationController, implementing the necessary functionalities to take advantage of the pyramid capability of the controller. It allows to change the image resolution during an ongoing registration run. Thus, an approximate solution can be found at a coarse resolution. Based on this approximate solution, a more accurate result can then be computed using higher resolutions. That way, time can be saved since the duration of one iteration step is basically proportional to the image volume. In addition to that, the application also allows to change metrics and interpolators during a registration run.

The multi resolution registration application is command line driven. It takes a configuration file, an output file name, a log file name and some display options as its command line parameters.

Parameters:

-config: name of the configuration file (see section A.7)

-out: name of the output file (which looks basically the same as the configuration file, but contains some output parameters calculated during the registration run. It can be used as a configuration file for other runs)

-log: name of the log file. The log file keeps track of the optimizer steps.

-visualize: decides which images should be displayed on the screen. There is always the moving (red) and the fixed image (green) displayed in the same window. The options are *all*, *end*, and *none*. '*all*' displays the original images, the images with the initial guess applied, the pre-processed images and the final result of the registration. '*end*' only displays the final result and '*none*' doesn't display any images.

-factor: defines a shrink factor used for visualization. This factor has been introduced for performance reasons.

-*eval*: decides whether the metric values for all three metrics should be calculated to evaluate the resulting transform found during the registration process. Possible values are *true* and *false*.

A.3 AppLandmarkPicker

AppLandmarkPicker is an interactive tool that lets the user pick initial guess landmarks on 3D AIM data. From the set of landmarks on the fixed and the moving image a transform is calculated that fits the sets onto each other using a mean-squares approach. In order to function properly, both sets must have the same number of landmarks and they must be ordered so that the n'th landmark from the fixed set corresponds to the n'th landmark in the moving set. The application assigns colors to the marks so that the order can be verified visually. At least three landmarks are required to completely define a rigid transform. Picks should be made as exactly as possible, but going down to pixel resolution is not necessary.

The tool takes a config file as its single input parameter. It reads the names of the image files from this file and stores the transform in the initial guess section of the file when finished picking.

The following key commands apply:

'P': pressing 'P' sets a landmark if the object was picked. If the mouse pointed to another landmark, this mark will be removed and a recoloring of the remaining landmarks takes place.

'T': toggles the trackball functionality which might be easier to handle the object.

'Q': quits the actual picking window. Within the first picking window it will open up the second window. Pressing 'Q' within the second window initiates the calculation of the transform and writes it into the config file.

Parameters:

-*config*: the config file

-*factor*: a shrink factor for faster visualization

A.4 AppAIMTransformer

The AIM transformer project takes a 3D AIM image, transforms it by a rigid 3D transform and writes the result to disk as a new AIM file. The output image size is enlarged so that the transformed input volume exactly fits in it. Further, the AIM transformer ensures that the origin is located on the pixel

grid, which is required for AIM images (the origin has an integer index within the pixel grid which can correspond to a real value in world coordinates).

The application is command line driven. It takes an input image name, an output image name and a transform as command line parameters. The transform is described in rigid 3D parameters representation.

Parameters:

- in*: input file name
- out*: output file name
- ax*: x-component of the rotation axis
- ay*: y-component of the rotation axis
- az*: z-component of the rotation axis
- phi*: rotation angle in rad
- tx*: x-translation in real world coordinates
- ty*: y-translation in real world coordinates
- tz*: z-translation in real world coordinates

A.5 AppRegistrationVisualizer

Visualizes the result of a registration given by the output file or the registration. It allows to define two planes and a sphere and only visualizes the part of the registered images that lies within the intersection of these three objects.

Parameters:

- config* filename that the images and the best registration transform is taken from
- orig1* *x y z* origin of implicit plane function one q -*normal1* *x y z* normal vector of plane one
- orig2* *x y z* origin of implicit plane function two
- normal2* *x y z* normal vector of plane two
- center* *x y z* origin of implicit sphere function
- radius* *r* radius of the implicit sphere function
- factor* *f* shrink factor applied on the images for fast parameter tryout

A.6 AppEvaluationTool

This tool takes registration output files and calculates different measures from it: the MSQ, NC and MATTES values for the best registration result and the TRE value for the reference transform.

The output is written to stdout and consists of the following numbers: the best transform from the evaluated result file (7 numbers), the difference between the reference and evaluated transform (7 numbers), the TRE (1 number), the MSQ/NC/MATTES values (3 numbers) plus the calculation time (1 number).

Parameters:

-ref ax ay az angle tx ty tz explicit reference transform OR
-dataset filename result file from where the best result transform is taken as the reference transform
-MSQ true initiates an MSQ calculation
-NC true initiates an NC calculation
-mattes true initiates a mutual information metric calculation
names of files to be evaluated are appended to the command line

A.7 Config File Format

```
dataset_filename:          example.config
purpose_of_registration_run: mouse femur 12 weeks
computer_type:            Athlon 700
```

INPUT SECTION:

=====

```
fixed_image_filename:      C57_03_FE_pos_02_r02_FID.aim
moving_image_filename:     C57_03_FE_pos_04_r02_FID.aim
initial_guess_trafo_axis_x: 0.00299614
initial_guess_trafo_axis_y: -0.00171628
initial_guess_trafo_axis_z: 0.999994
initial_guess_trafo_angle: 1.83402
initial_guess_trafo_trans_x: 8.88283
initial_guess_trafo_trans_y: 1.12631
initial_guess_trafo_trans_z: -0.344254
pre_transform_on:          0
pre_transform_axis_x:      0
pre_transform_axis_y:      0
pre_transform_axis_z:      1
pre_transform_angle:       0
pre_transform_trans_x:     0
pre_transform_trans_y:     0
pre_transform_trans_z:     0
shrink_factor:             2
```

```

multi_resolution_levels:      3
multi_resolution_shrink_factors: 5 2 2
multi_resolution_iterations:   500 100 10
multi_resolution_metrics:      meansquare mattes mattes
multi_resolution_interpolators: linear linear bspline
meansquare_learning_rate:      5e-11
mattes_learning_rate:          3e-05
ncorr_learning_rate:           3e-05
optimizer_type:                quaternionrigid
optimizer_minstep:             0.01
optimizer_maxstep:             4
optimizer_learningrate:        5e-11
optimizer_scales_0:            1
optimizer_scales_1:            1
optimizer_scales_2:            1
optimizer_scales_3:            1
optimizer_scales_4:            0.035
optimizer_scales_5:            0.035
optimizer_scales_6:            0.035
max_number_of_iterations:      300
metric_type:                   meansquare
number_of_histogram_bins:      50
number_of_spatial_samples:    0.15
interpolator_type:             linear
bspline_order:                 3
transform_type:                quaternion

```

OUTPUT SECTION:

```
=====
```

```

registration_start_time:      Wed Aug 22 17:02:13 2003
registration_execution_time:   568
actual_number_of_iterations:   300
error_message:
peak_memory_usage:            -1
fixed_image_size_x:           40
fixed_image_size_y:           40
fixed_image_size_z:           40
fixed_image_spacing_x:        1
fixed_image_spacing_y:        1
fixed_image_spacing_z:        1
fixed_image_origin_x:         0

```

```

fixed_image_origin_y:      0
fixed_image_origin_z:      0
moving_image_size_x:       44
moving_image_size_y:       44
moving_image_size_z:       40
moving_image_spacing_x:    1
moving_image_spacing_y:    1
moving_image_spacing_z:    1
moving_image_origin_x:     -4
moving_image_origin_y:     0
moving_image_origin_z:     0
result_metric_value:       -1.24208
result_transform_axis_x:    0.0312073
result_transform_axis_y:    0.0278015
result_transform_axis_z:    0.999126
result_transform_angle:     0.0929704
result_transform_xtrans:    0.106729
result_transform_ytrans:    0.664279
result_transform_ztrans:    0.0212211
result_best_metric_value:   -1.24208
result_best_transform_axis_x: 0.0312073
result_best_transform_axis_y: 0.0278015
result_best_transform_axis_z: 0.999126
result_best_transform_angle: 1.7929704
result_best_transform_xtrans: 8.106729
result_best_transform_ytrans: 1.664279
result_best_transform_ztrans: 0.0212211

```

dataset_filename : File name of the dataset. This parameter is automatically set by the application.

purpose_of_registration_run : String representing the purpose of the registration run. This parameter does not affect the registration. It is only used as a reminder for the user and has to be set manually.

computer_type : A string representing the computer on which the registration was carried out. This parameters has to be set manually by the user.

fixed_image_filename : File name of the fixed image. This filename is only used, if the SetupRegistration() method of the RegistrationController is called with parameter NULL for the FixedImageReader (which is the case in the command line tools).

moving_image_filename : File name of the moving image. This filename is only used, if the `SetupRegistration()` method of the `RegistrationController` is called with parameter `NULL` for the `MovingImageReader` (which is the case in the command line tools).

initial_guess_trafo : The initial guess for the registration. 'Axis' is the rotation axis, 'angle' is the rotation angle measured in rad and 'trans' is the translation component of the transform measured in mm. These parameters are automatically written by the landmark picker application.

pre_transform_on : Sets a flag that decides whether a pre transform is performed or not. If true, the pre transform is performed, if false it is not. This flag has been introduced to save the time consumed by applying an identity transform.

pre_transform : The pre-transform for the registration run. It is a transform applied to the moving image prior to the registration. 'Axis' is the rotation axis, 'angle' is the rotation angle measured in rad and 'trans' is the translation component of the transform measured in mm. The pre-transform is only executed if the `pre_transform_on` flag is set to true.

shrink_factor : Sets the shrink factor used to shrink the fixed and the moving image during the pre processing step. The shrinking is done to reduce the memory consumption and to speed up the registration. The shrink factor can be any positive floating point number and it defines the rescaling along the axis with the smallest voxel spacing. This parameters is only used if 'multi_resolution_levels' is set to 1.

multi_resolution_levels : Sets the number of levels used for multi resolution registration. If the value is smaller or equal to one, single resolution (with the shrink factor set by the 'shrink_factor' tag) registration is performed. If the value is larger than 1, these number has to match the number of values entered for the other multi resolution parameters.

multi_resolution_shrink_factors : Sets the shrink factors for the different levels of the multi resolution registration. They have to be delimited by a space character. The first entry is the one for the lowest resolution (i.e. the largest shrink factor).

multi_resolution_iterations : Sets the number of iterations for each level of the multi resolution registration. This means, the first entry denotes

the number of iterations performed on the lowest resolution level, the second entry the number of iterations performed on the second lowest resolution level, and so on. The values have to be separated by a space character.

multi_resolution_metrics : Sets the metrics used for the different resolution levels separated by space characters. Valid metrics are 'mean-square', 'mattes' and 'normalizedcorrelation'.

multi_resolution_interpolators : Sets the interpolators used for the different resolution levels separated by space characters. Valid interpolators are 'nearest', 'linear' and 'bspline'.

meansquare_learning_rate : Sets the learning rate used for the mean-squares metric if multi resolution registration is used (i.e. if the number of multi resolution levels is set to a value larger than 1). This parameter is not used for single resolution registration. For single resolution registration, 'optimizer_learning_rate' is used instead. The correction factor calculated by the observer during the registration is also applied to the value entered here when the resolution level is switched. The values have to be separated by a space character.

mattes_learning_rate : see meansquare_learning_rate.

ncorr_learning_rate : see meansquare_learning_rate (ncorr means normalizedcorrelation)

optimizer_type : Sets a string representing the optimizer used during registration. So far, only 'quaternionrigid' is supported.

optimizer_minstep : Sets the minimum step size that is the terminating condition for some optimizers like the regular step gradient descent optimizer. Since no such optimizer is supported at the moment, this parameter does not have any effect.

optimizer_maxstep : Sets the maximum step size of an optimizer, which is the starting step size for some optimizers. Since no such optimizer is supported at the moment, this parameter does not have any effect.

optimizer_learningrate : Sets the learning rate for the optimizer, if it supports this parameter. The 'quaternionrigid' optimizer (which is the only one implemented at the moment) does support this parameter.

optimizer_scales : Sets the scales for an optimizer. The scales define a weight for each parameter of the transform used during optimization. The lower a scale value, the more intensively the optimizer will adjust this parameter during optimization. In the implemented case ('quaternionrigid' optimizer), the last three scales denote the translation component of the transform, which usually has to be weighted differently than the rotation component. The scales are normalized to a unit cube to make them independent on the image size.

max_number_of_iterations : Sets the maximum number of iterations that are performed by the optimizer. This parameter is only used in case of single resolution registration. For the multi resolution registration 'multi_resolution_iterations' is used instead.

metric_type : A string representing the metric used during registration. Valid strings are 'meansquare', 'mattes' and 'normalizedcorrelation'. This parameter is only used for single resolution registration. For multi resolution registration 'multi_resolution_metrics' is used instead.

number_of_histogram_bins : parameter for mattes mutual information metric. See section 5.1.2 for details. Reasonable values for 60MB AIM files are around 50.

number_of_spatial_samples : parameter for mattes mutual information metric. See section 5.1.2 for details. This value is expressed in percent of the total number of pixels and shows good results for roughly 0.1 to 0.2

interpolator_type : Sets a string representing the interpolator used during registration. Valid strings are 'nearest', 'linear' and 'bspline'. This parameter is only used for single resolution registration. For multi resolution registration 'multi_resolution_interpolators' is used instead.

bspline_order : Sets the spline order used for the bspline interpolation. If another interpolator than bspline is set, this parameter does not have any effect.

transform_type : A string representing the transform used during registration. So far, only 'quaternion' is supported.

registration_start_time : The date and time when the registration was started.

registration_execution_time : The duration of the registration in seconds.

actual_number_of_iterations : The number of iterations that were carried out.

error_message : not implemented.

peak_memory_usage : not implemented.

fixed_image_size : The size of the fixed image in pixels.

fixed_image_spacing : The spacing of the fixed image in mm.

fixed_image_origin : The origin of the fixed image in mm.

moving_image_size : The size of the moving image in pixels.

moving_image_spacing : The spacing of the moving image in mm.

moving_image_origin : The origin of the moving image in mm.

result_metric_value : The metric value that was calculated for the last iteration.

result_transform : The transform (i.e. position) of the last iteration. 'Axis' is the rotation axis, 'angle' is the rotation angle measured in rad and 'trans' is the translation component of the transform measured in mm.

result_best_metric_value : The best metric value that was achieved during the registration. In case of multi resolution registration, only the last resolution level is considered.

result_best_transform : The transform that belongs to the 'result_best_metric_value'. 'Axis' is the rotation axis, 'angle' is the rotation angle measured in rad and 'trans' is the translation component of the transform measured in mm.

Appendix B

Glossary

AIM Image Format: This 3D density image format has been developed at ETH Zurich and is the main input format for the registration module developed in this project.

Catch Range: A region within the parameter space of the optimizer where the optimizer is guaranteed to converge toward the optimum. If the initial guess to an optimizer is outside of its catch range, the optimizer will converge into a sub-optimum.

Euclidean Norm: A measure defined on vectors and matrices. It is defined as the square root of the sum of absolute squares of its elements.

Fiducial Mark: In order to define initial guesses or evaluate the accuracy of registration methods, artificial marks can be attached to the objects that are scanned. They can be designed to have a homogeneous density and sharp contours and can therefore easily be segmented from the actual data and registered separately.

Quaternion: A quaternion is a four-dimensional complex number representation and frequently used in computer graphics and 3D data processing. Within ITK it is one way to represent transforms (see section 4.4.2).

Initial Guess: To make sure the registration starts of within the *catch range* of its metric, an initial guess transform can be provided. The optimizer takes this transform as its starting point.

Landmark: Landmarks are selected points on the fixed and moving image objects. They are obtained by picking within an interactive picking tool. If two sets of landmarks are available, a transform can be calculated that brings one set as close to the other as possible and therefore represents a possible initial guess transform for a registration process.

Learning Rate: This is a scalar value that says how big the steps are that the optimizer takes in each iteration of a registration process. The optimizer multiplies its vector of the assumed direction of the optimum with this value. Within ITK, observer instances take care of the adjustment of this value, which should get smaller the more the optimum is approached.

Reference Counting: A technique to keep track of instances of objects in object oriented programming (OOP). Each object is assigned a reference count that is increased as soon as a reference (pointer) to that object is created. Whenever a reference to that object is eliminated, the reference count is decreased. If it hits zero, then the object is removed from memory since it is not accessible anymore. This technique eliminates memory leaks if used properly. ITK has reference counting built in.

Rigid Transform: This transform type is restricted to rotations and translations and therefore is defined with six parameters, three for the rotations around the coordinate axes and three translational parameters.

TRE: Total Registration Error. A measure commonly used to assess registration methods. See section 5.4.1 for details.

VOI: Volume of interest. This denotes a subpart of an image that a user is interested in.

Appendix C

Gallery

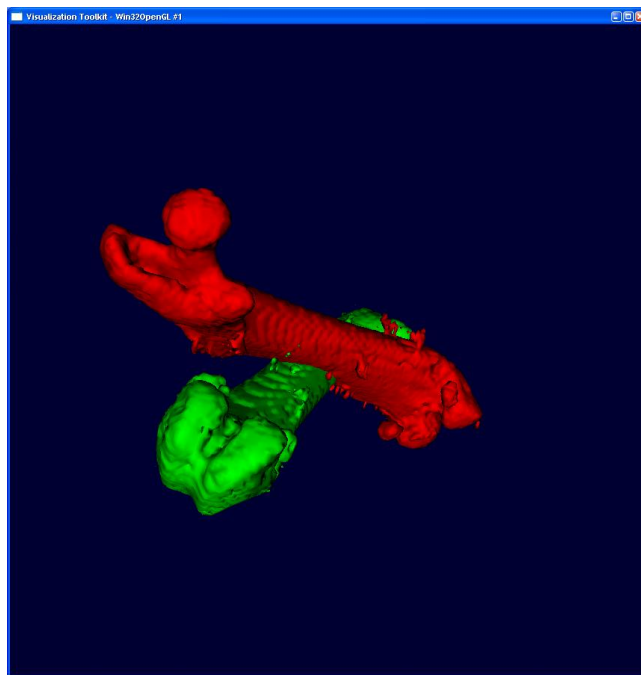


Figure C.1: Two mouse femurs as they are originally located in space before registration

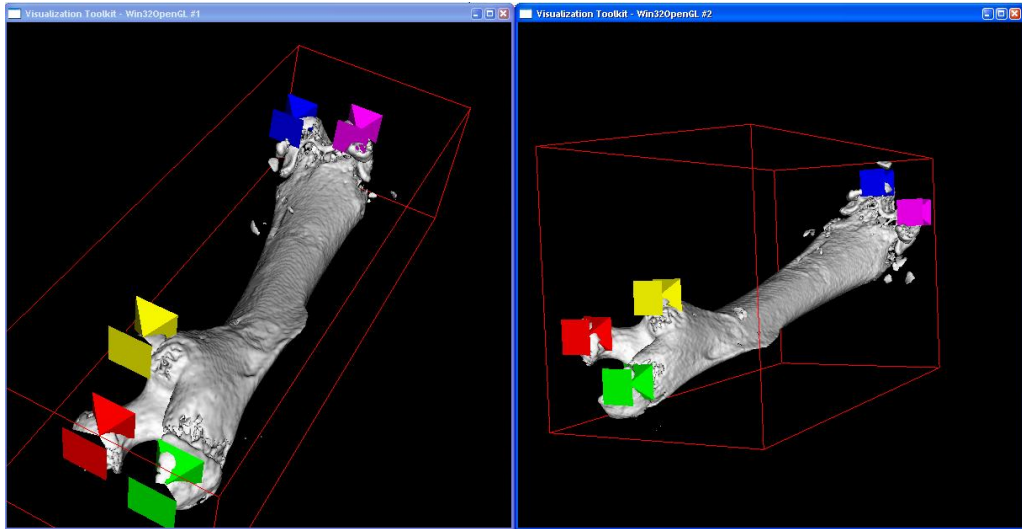


Figure C.2: The Landmark Picker Windows

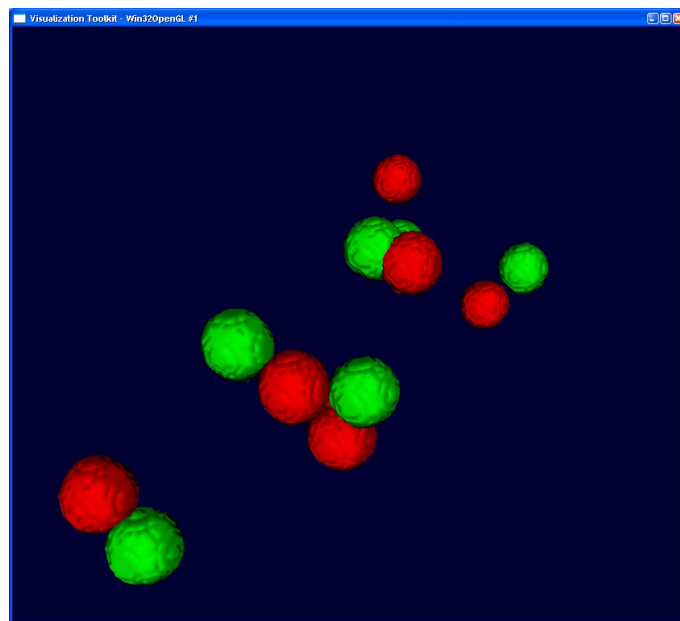


Figure C.3: Unregistered beads (fiducial marks)

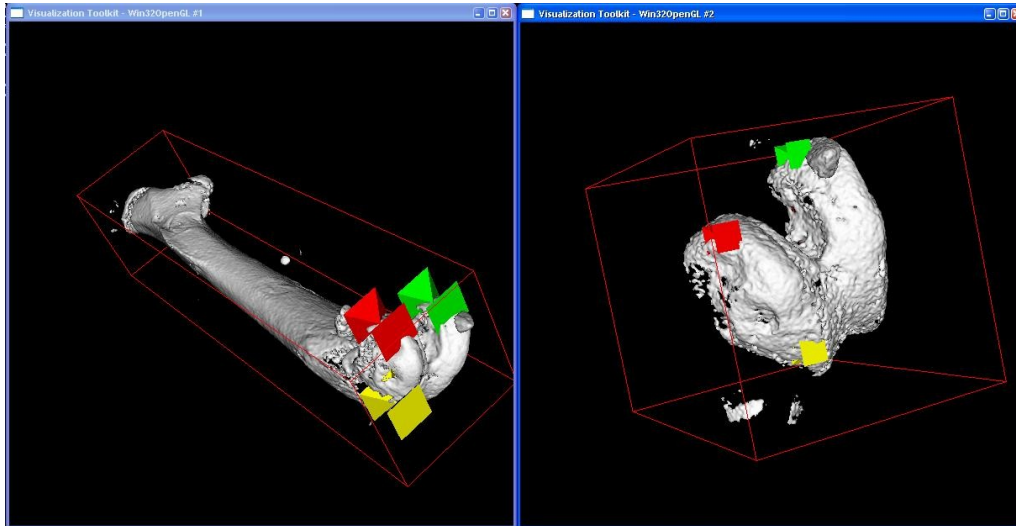


Figure C.4: Sub volume picking

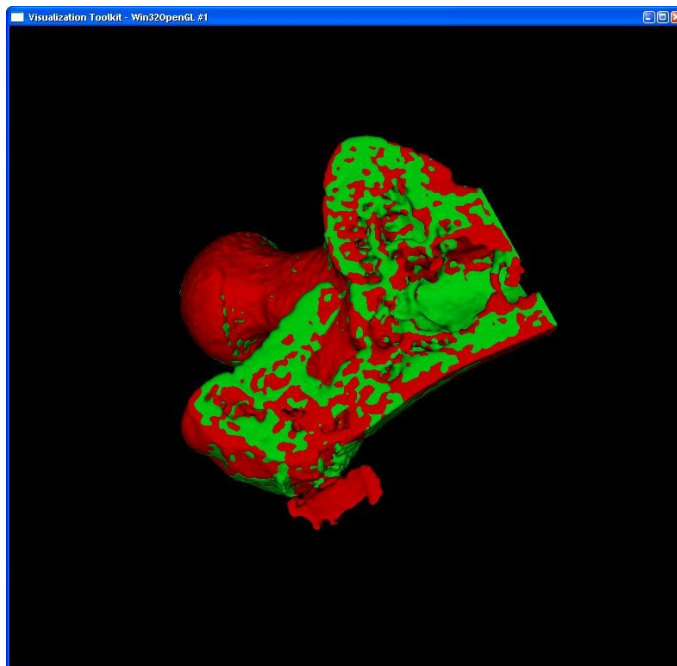


Figure C.5: Registered hip region of a mouse femur: The cut allows to see the internal structure of the two bone images.

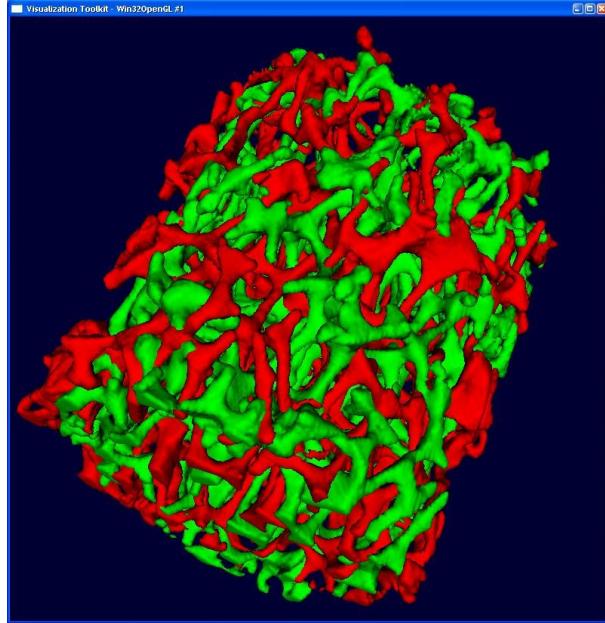


Figure C.6: Two samples of an aluminum foam that are to be registered

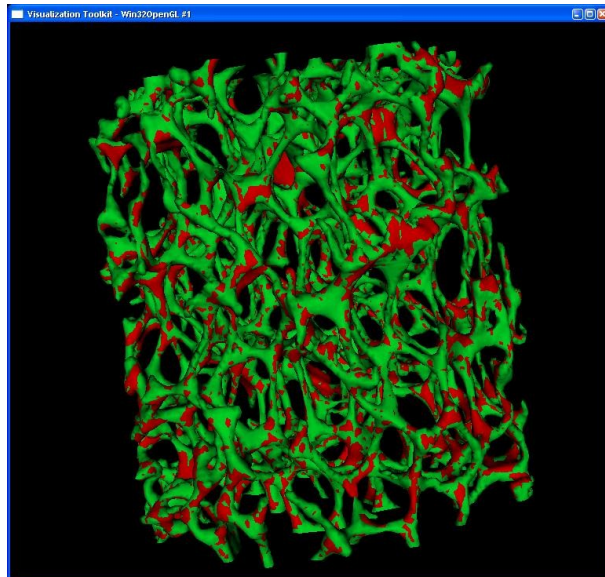


Figure C.7: Aluminum foam samples after registration - full view

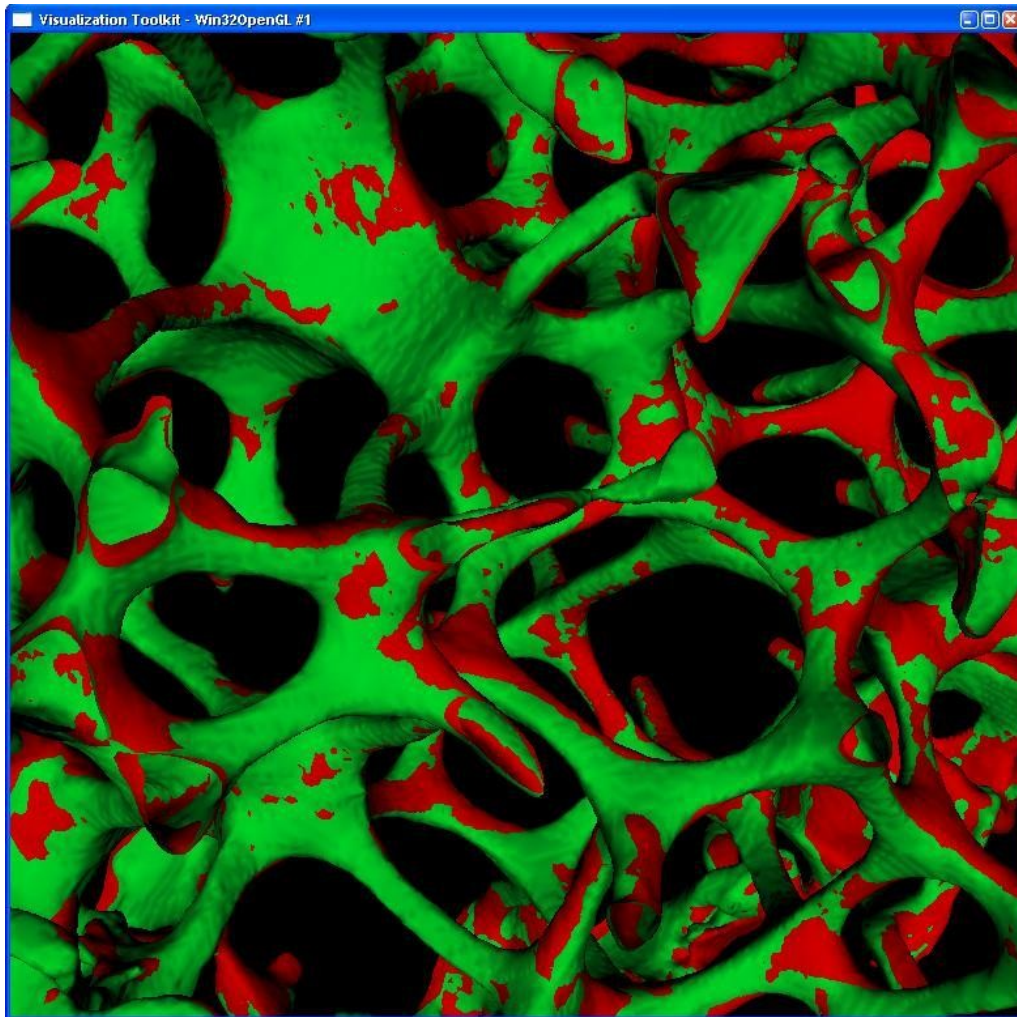


Figure C.8: Aluminum foam samples after registration - close-up view showing detailed structures

Bibliography

- [1] <http://www.kitware.com>
- [2] Luis Ibanez, Will Schroeder and the Insight Consortium *The ITK Software Guide*, The ITK on-line guide @ www.itk.org, constantly updated
- [3] <http://www.vtk.com>
- [4] vtkusers@vtk.org; <http://public.kitware.com/mailman/listinfo/vtkusers>
- [5] Joseph V. Hajnal, Derek L.G.Hill, David J.Hawkes *Medical Image Registration*, CRC Press LLC (The Biomedical Engineering Series), ISBN 0-8493-0064-9, 2001
- [6] Kitware Inc. *The VTK User's Guide - Updated for Version 4.2*, Kitware Inc., ISBN 1-930934-08-4, 2003
- [7] www.seas.upenn.edu/cis580/Lectures/Lecture2.ppt
- [8] www.cs.wisc.edu/condor/
- [9] Jacob Beutel, Harold L. Kundel, Richard L. Van Metter *Handbook of Medical Imaging*, SPIE - The International Society for Optical Engineering, Bellingham, Washington, ISBN 0-8194-3622-4, 2000
- [10] Viola P., Wells W.M. Alignment by maximization of mutual information. *International Journal of Computer Vision* 24(2):137-154. 1997
- [11] D.Mattes, D.R.Haynor, H.Vesselle, T.K.Lewellen, W.Eugank. Non-rigid multi-modality image registration. *Medical Imaging 2001: Image Processing*, 1609-1620, 2001.
- [12] Markus Gross, *Graphische Datenverarbeitung I*, ETH Zürich, script of the lecture

Index

- AB-BA registration, 27
- AIM image format, 11, 53
- AIM reader, 11, 31
- AIM transformer, 43
- AIM writer, 11
- b-spline interpolation, 24
- catch range, 9, 21, 53
- Condor, 35
- condor_compile, 36
- condor_q, 36
- condor_rm, 36
- condor_submit, 36
- config file, 41
- cross correlation, 23
- dataset class, 13
- entropy, 21
- Euclidean norm, 8, 53
- Euler 3D transform, 25
- event-observer, 16
- fiducial mark, 53
- fiducial mark registration, 27
- fiducial marks, 56
- fixed image, 7
- generic programming, 12
- homogeneous coordinates, 19
- image registration, 1, 7, 11, 38, 39
- initial guess, 8, 9, 43, 53
- inter-modal registration, 7
- interpolation, 24
- intra-modal registration, 7
- ITK, 11, 31, 32
- ITKRegistration application, 41
- job migration, 36
- joint entropy, 23
- landmark, 54
- landmark picker module, 13, 16
- landmarks, 43
- learning rate, 26, 54
- linear interpolation, 24
- local optimum, 9
- MattesMutualInformationMetric, 23, 29
- memory efficiency, 32
- metric function, 7, 25
- moving image, 7
- multi resolution registration, 42
- multi-resolution registration controller, 13
- multi-resolution registration module, 13
- MultiResolutionRegistrationController, 42
- mutual information, 23
- nearest neighbor interpolation, 24
- non-rigid registration, 8
- non-rigid transform, 8
- normalized correlation metric, 23
- number of bins, 29
- number of spatial samples, 29

observer, 15
optimizer, 7, 9, 26
optimizer-transform pair, 25
osteoporosis, 1

pipeline mechanism, 32

quaternion, 53
quaternion optimizer, 26
quaternion parameters, 19
quaternion rigid transform, 25

reference counting, 54
registration controller, 15
registration module class, 13
registration run, 13
ReleaseDataFlag, 32
resolution pyramid, 42
rigid 3D transform parameters, 20
rigid registration, 8
rigid transform, 8, 19, 43, 54

segmentation, 11, 27
single resolution registration controller, 13
single resolution registration module, 13
Standard Universe, 35
swapping, 9

thresholding, 27
TRE, 27, 44, 54

unit spacing, 31

Vanilla Universe, 35
versor rigid 3D transform, 25
VOI, 54
VTK, 11, 12, 31
VTK-to-ITK interface, 31, 32