Institute for Computer Graphics and Vision
Graz University of Technology

Master's Thesis

GPU-Based 2D/3D Registration of X-Ray and CT Data

by

Tobias Gross

Advisor: Dipl.-Ing. Dr.techn. Markus Grabner

Graz, January 2008

Abstract

By means of 2D/3D registration, high-quality CT patient scans can be integrated into medical interventions, which allows surgery to be guided by images and additional information, such as the location of tumorous tissues, to be stored and recovered during operations. For this purpose the exact pose of the patient in the operating room has to be established. This can be accomplished by digitally reconstructing radiographs from the patient scan in an emulated imaging setting and comparing them to intraoperative X-ray images. Once aligned with the patient, the CT data can be used to augment the information available to the surgeon, thus facilitating minimally-invasive surgery. However, the registration has to be fast, accurate, and reliable to be useful for medical application.

The method proposed in this work is based on an intensity-based similarity measure which is inoffensive to the patient and does not require any user interaction. The *automatic* or *algorithmic differentiation* technique is used to generate the derivative of the volume rendering and similarity measuring code. The resulting program evaluates the gradient with respect to the transformation parameters on execution. As opposed to the numerical approximation of the gradient, its performance is virtually insensitive to the number of transformation parameters. Additionally, all computationally and memory intensive parts of the program are computed by means of programmable graphics hardware, which is laid out to perform simple and repetitive tasks very efficiently.

The quality of the method was assessed by means of a standardized evaluation method involving target registration error, capture range, and time measurements. An approach based on numerical approximation served as reference for comparisons. For a $512 \times 512 \times 346$ voxels scan of a human abdomen and three corresponding 512×512 pixels X-ray images, a speed-up factor of 2.4, which resulted in a mean registration time of 28.5 seconds, was observed. The mean target registration error averaged 0.6 millimeters for successful alignments. Although the reference approach proved more accurate, both methods were about equally robust. For a maximum initial error of 33 millimeters, the presented approach converged successfully in 90 percent of all cases. The concept, which still leaves a lot of room for further improvement, is thus considered as qualified for medical application.

Zusammenfassung

Mit Hilfe von 2D/3D Registrierung kann ein hochqualitatives CT Bild des Patienten in chirurgische Eingriffe einbezogen werden. Dies ermöglicht einerseits die bildgesteuerte Navigation, und andererseits das Speichern und Abrufen von Information, wie etwa die Lage von tumorösem Gewebe, während der Operation. Dazu muss jedoch die genau Lage des Patienten im Operationssaal bestimmt werden. Dies kann anhand einer digitalen Rekonstruktion von Röntgenbildern aus einem CT Bild innerhalb der nachgebildeten Aufnahmekulisse und eines anschließenden Vergleichs mit Röntgenbildern, die während der Operation aufgenommen werden, erreicht werden. Sobald die Bilder in Deckung gebracht wurden, kann das CT-Volumen dazu verwendet werden, den Informationsgehalt, der dem Chirurgen zur Verfügung steht, zu vergrößern und so minimal-invasive Eingriffe erleichtern. Um für eine medizinische Anwendung geeignet zu sein, muss die Registrierung jedoch schnell, genau und zuverlässig funktionieren.

In der vorliegenden Arbeit wird eine Methode vorgestellt, die auf einem intensitätsbasierten Ähnlichkeitsmaß, das unschädlich für den Patienten ist und keine Interaktion mit Anwendern erfordert, beruht. Zur Ableitung der Prozedur, die das Volumen rendert und das Ähnlichkeitsmaß bestimmt, wird das Verfahren der *automatischen Differenzierung* verwendet. Das resultierende Programm wertet den Gradienten bezüglich der Transformationsparameter aus. Im Gegensatz zur numerischen Approximation des Gradienten ist die Laufzeit dieses Programms praktisch unabhängig von deren Anzahl. Zusätzlich werden alle rechen- und speicherintensiven Teile des Programms mit Hilfe von programmierbarer Grafikhardware, die für die effiziente Verarbeitung von einfachen und repetetiven Aufgaben ausgelegt ist, ausgeführt.

Die Qualität der Methode wurde anhand eines standardisierten Evaluierungsverfahrens unter Einbeziehung des Registrierungsfehlers, des Einfangbereichs und der Laufzeit bewertet. Als Referenz für Vergleiche diente ein auf der numerischen Annäherung des Gradienten basierender Ansatz. Für ein $512 \times 512 \times 346$ CT Bild eines menschlichen Unterleibs und drei dazugehörige Röntgenbilder mit je 512×512 Bildelementen wurde ein Beschleunigungsfaktor von 2.4 gemessen, was eine mittlere Registrierungsdauer von 28.5 Sekunden ergab. Der mittlere Registrierungsfehler betrug 0.6 Millimeter für erfolgreiche Registrierungen. Obwohl sich der Referenzansatz als genauer erwies, funktionierten beide Methoden etwa gleich robust. Bei einem maximalen Anfangsfehler von 33 Millimetern konvergierte der vorgestellte Ansatz in 90 Prozent aller Fälle. Die Methode, die noch reichlich Raum für weitere Verbesserungen lässt, wird deshalb als geeignet für die medizinische Anwendung erachtet.

I hereby certify that the work presented in this thesis is my own and that work performed by others is appropriately cited.

Ich versichere hiermit, diese Arbeit selbständig verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Acknowledgements

First of all, I would like to thank Markus Grabner for providing valuable advice and feedback, producing new ideas, and being available for questions at almost any point of time. Thanks to Dieter Schmalstieg, Horst Bischof, and the staff at the Institute for Computer Graphics and Vision at Graz University of Technology for supporting this work and providing all necessary facilities to realize this thesis. Special thanks to Thomas Pock for his aid on algorithmic differentiation and segmentation issues as well as Matthias Rüther and Werner Trobin for carrying out camera calibrations.

Finally, I would like to thank Prof. Bartsch from the Urology Clinic of the University Hospital Innsbruck for partly financing this work, giving medical advice, and providing all necessary X-ray and CT data to carry out the experiments presented in the thesis.

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Thesis Outline 3

2 Related Work **5**
2.1 Rigid-Body 2D/3D Registration 5

3 2D/3D Registration **11**
3.1 Introduction 11
3.2 Rigid-Body 2D/3D Registration 17
3.3 Digitally Reconstructed Radiographs 19
3.4 Camera Calibration 26
3.5 Intensity-Based Similarity Measuring 28
3.6 Optimization 34

4 GPU-Based Computation **41**
4.1 Introduction 41
4.2 GPU Capabilities 41
4.3 General Purpose GPU Programming 44
4.4 GPU-Based Registration 47

5 Algorithmic Differentiation **55**
5.1 Introduction 55
5.2 Algorithmic Differentiation 56
5.3 Differentiation of the Registration Algorithm 63

6 Results **73**
6.1 Implementation and Operation 73
6.2 Validation 73
6.3 Convergence and Accuracy 75
6.4 Performance 83

7 Conclusion **85**
7.1 Summary 85
7.2 Discussion 86
7.3 Future Perspectives 87

Chapter 1

Introduction

1.1 Motivation

Medical interventions often cause serious operative traumata to the patient. In many cases, however, the body is affected to a larger extent than is actually required to achieve the desired outcome of the intervention. Immoderate damage to the body as a byproduct of surgery has many causes. One of the key problems is the inability to locate positions and regions of interest, such as tissue that has been priorly diagnosed with tumor, precisely. Additionally, the surgeon has a limited field of view when performing invasive interventions. Above all, the applicability of automatic tools assisting the operation such as surgical robots is limited when no reliable information on locations within the patient's body is available.

Medical imaging provides more and more insight into the human body. The quality of the images that can be acquired is ever growing. On one hand, this allows one to draw more accurate conclusions about the condition of the patient. On the other hand, the computational requirements to process these data is increasing along with the images' resolution. Three-dimensional datasets such as produced by *computed tomography* (CT) or *magnetic resonance imaging* (MRI) represent highly accurate models of the patient's body. They may act as a reference for tagging points and regions of interest, thus augmenting the information conveyed by the original data. When visualized, they provide very realistic images of the body which can be used for diagnosis and to guide surgery. However, as long as there is no reliable means to make them available during interventions without impeding the work flow of medical stuff either physically or by delays, they are useless in the context of minimally-invasive operations.

Minimally-invasive surgery is a very promising field in medicine. It aims at causing less traumata to the patient by limiting both the region that is affected while accessing the operation target and the region where surgery is applied. Hence, interventions are more accurate and have less adverse effects and complications arising afterwards. Although the operations may be more complicated to conduct, the patient is strained less and recovers faster. Minimally-invasive surgery thus encourages a trend towards ambulant routine interventions, which often total to lower treatment costs. An example for a minimally invasive operation is the local treatment or removal of tumorous tissue without removing the whole organ affected. This reduces the trauma caused by surgery and may allow the organ to keep up its function.

To enable a broad range of minimal-invasive surgery, the exact pose of the patient within a common frame of reference in the operating room has to be known. This allows positions within the body to be filed, retrieved, and referred to by all participants, be it humans or machines. Furthermore, the visualization of patient data may be used in order to virtually extend the field of view and the amount of information available to the surgeon.

This work aims at building a reliable basis for minimally-invasive and image-guided surgery by presenting a method to efficiently integrate high-quality preoperative information into the operation. The task consists in establishing a relationship between the patient and a three-dimensional scan serving as a model of the their body. Given this relationship, any point in the volume can be mapped to a location in the patient and vice versa. This enables the use of the scan as a reference for filing locations of interest within the body, for example, positions where tissue samples where taken, and recovering them at a later point of time, for example, to treat regions diagnosed with tumor. Additionally, the image can be integrated visually to guide or assist the intervention. However, the procedure does not only align the model with the patient, but also yields the pose of the patient in the operating room with respect to some predefined coordinate system. It may serve as a common frame of reference for all devices involved in the procedure.

X-ray images taken during the intervention reflect the current pose of the patient. In order to establish the alignment, the whole X-ray imaging setting in the operating room is emulated with a three-dimensional scan acting as a model of the patient. The result of simulating the acquisition of an image is a so-called *digitally reconstructed radiograph* (DRR), which constitutes the virtual counterpart of an X-ray image. The model is placed into the emulated operating room and its pose is adapted until an optimal match between the original and the reconstructed radiograph(s) has been found. Once established, the patient and the scan share the same pose within the predefined coordinate system. The entire process of identifying a transformation in three-dimensional space by means of two-dimensional images is referred to as *2D/3D registration*.

Since we are dealing with a medical application that is aimed at minimally harming the patient, speed, accuracy, reliability, and ease of use constitute the main criteria for success. Additionally, the procedure has to be inoffensive to the patient, which is an absolute necessity considering the fact that the application is supposed to be used in the course of routine examinations.

To achieve a major speedup of the alignment, we want to exploit the computational power of modern *graphic processing units* (GPUs), which are optimized for performing repetitive tasks on large amounts of data. The use of high-resolution data, in turn, promotes accuracy. Moreover, *algorithmic* or *automatic differentiation* (AD), which is novel in the context of medical image registration, is to be used to guide the search towards an optimal match between the original and the reconstructed radiographs in an efficient and accurate manner. This work acts on the assumption that 2D/3D registration still provides a lot of room for improvement, particularly in terms of speed and robustness. GPU-based computation and algorithmic differentiation are promising approaches to significantly reduce the registration time while not impairing the application's accuracy, robustness, and ease of use, thus qualifying it for the use in minimally-invasive surgery.

1.2 Thesis Outline

In Chapter 2, selected work in the field of 2D/3D registration is presented and shortly discussed. The focus lies on different approaches to DRR generation and GPU-based methods.

Chapter 3 elaborates on the nature of rigid-body transformation and subsumes the typical steps of 2D/3D registration, namely the generation of *digitally reconstructed radiographs* (Section 3.3), the calibration of the X-ray camera (Section 3.4), measuring similarity between the original and the reconstructed radiographs (Section 3.5), and finding a set of transformation parameters that minimizes the cost function (Section 3.6).

In Chapter 4 the capabilities of modern graphics hardware and its use for general purpose programming (Section 4.3) are discussed. Section 4.4 presents the GPU-based implementation of the registration algorithm and describes how programming concepts are mapped to the GPU.

Chapter 5 provides an overview of the *algorithmic differentiation* technique. First its methodology is established theoretically (Section 5.2), then it is applied to the registration algorithm (Section 5.3). Eventually the GPU-based evaluation of the derivative code is presented.

Chapter 6 presents the results of an experimental evaluation conducted with the implemented algorithm. First it shortly describes the implementation and the operational environment of the registration application, then it presents the results with respect to convergence and accuracy (Section 6.3), as well as performance (Section 6.4).

In the concluding Chapter 7 the presented work is summarized. Then the relevance of the findings and the meaning of the experimental results is discussed. Eventually the work will be provided with a future perspective.

Chapter 2

Related Work

2.1 Rigid-Body 2D/3D Registration

Rigid-body 2D/3D registration was extensively dealt with in scientific literature. Although there is a number of different approaches to the alignment of preoperative volume data and intraoperative images, no method combining speed, accuracy, robustness, inoffensiveness, and ease of use in a satisfactory manner has been found. Usually, the presented work focuses on some of these aspects while disregarding others. A general survey of medical image registration including a classification by nine distinctive criteria can be found in [Maintz and Viergever, 1998]. In order to make a direct comparison of different methods possible, a standardized evaluation schema for 2D/3D registration was proposed in [Van de Kraats et al., 2004]. It elaborates on a common definition of the registration error, starting positions, and the capture range.

According to the typical tasks within the rigid-body 2D/3D registration process, we state the following list of potential optimization targets:

- **DRR generation**
- **Similarity measuring**
- **Optimization**

where the former usually constitutes the most costly task because of potentially high memory and computational requirements. Since similarity measuring and optimization are fairly well-established, DRR generation is frequently the most promising target for registration enhancements.

2.1.1 DRR Generation

From work that has been presented in the field of rigid-body 2D/3D registration, we extracted the following classes of approaches to enhancing DRR generation, which are not mutually exclusive:

GPU-based approaches These approaches try to exploit the arithmetic power and bandwidth of modern graphics hardware. Essentially, DRR generation is reduced to a volume rendering task. In [LaRose, 2001] one of the first consumer-level GPU-based DRR generation applications is presented. It uses the two-dimensional texture mapping feature to render the DRRs. Similar approaches, yet with three-dimensional textures, are described in [Gong et al., 2006, Ino et al., 2006]. The latter additionally computes gradient images and the *normalized cross correlation* similarity measure on the GPU.

Programmable kernels in the rendering pipeline allowed for more complex volume rendering techniques to be performed on consumer-level graphics boards. In [Wein, 2003, Khamene et al., 2006a] DRRs are generated on the GPU using fragment shaders. In [Chisu, 2005, Kubias et al., 2007b] intensity-based similarity measures are evaluated on the GPU in addition. Both use texture *mipmapping* to reduce the pixel-wise results to single values. They observed a speed-up factor of around 4.5 compared to CPU-based rendering approaches. For a combination of eight similarity measures, a mean *target registration error* (TRE) of about 1.4 millimeters was measured in [Kubias et al., 2007b].

In [Khamene et al., 2006b] a general registration approach that reduces the number of image dimensions by means of a projection step is presented. Both the projection and similarity measuring are performed with the help of *shaders* in order to speed up computation. One approach averaging the pixel-wise results directly on the GPU, one averaging them on the CPU, and a completely CPU-SSE-based approach are compared. Experiments with a CT scan of the pelvis showed that the former was more than four times faster than the latter for certain similarity measures while being significantly less accurate. Yet, the hybrid approach could keep up with the CPU-based approach in terms of accuracy and still perform considerably better.

Another approach which does not deal with DRR generation yet is still worth mentioning is described in [Köhn et al., 2006]. For a registration in two or three dimensions the presented algorithm computes the derivative of the simple *sum of squared differences* (SSD) similarity measure, which is subsequently used for a *gradient descent* optimization procedure, on the GPU. Insofar, the underlying idea is similar to ours, yet the differentiation is performed symbolically, which is limited to very simple expressions such as the SSD formulation. Eventually, the volume of derivatives (for the three-dimensional case) is reduced by first rendering all slices to one texture with blending enabled and then reducing the texture by means of GPU-based image pyramid creation.

Gradient-based approaches This class of approaches uses volume gradients to produce some sort of gradient DRRs, thus incorporating spatial information. In [Wein et al., 2005] a gradient volume is used to render the DRRs both with *raycasting* and *splatting*. The resulting DRRs are then fed directly to the *gradient correlation* similarity measure. With additional optimizations like empty space skipping, they achieve a speed-up factor of about ten compared to normal raycasting and the same similarity measure. For real patient data, however, the standard deviation of the estimated transformation parameters as a measure of registration accuracy strongly increased.

In [Livyatan et al., 2003] a *gradient projection* approach is presented. The method is based on the observation that X-ray image gradients are linearly proportional to the weighted volume gradients along the viewing rays. It tries to find the correct alignment by maximizing the volume gradient projection at edge pixels of the radiographs. The experimental evaluation of the three step algorithm conducted with a human pelvis dataset and an initial displacement of 9.8 millimeters yielded a *surface TRE* of 1.7 millimeters.

In [Tomaževič et al., 2003], on the other hand, X-ray image gradients are projected back towards the X-ray source. The cost function then calculates the correspondence between the volume's surface normals, which are computed from the CT coefficients, and the back-projected image gradients. A registration of single vertebrae yielded mean TREs below 0.5 millimeters counting only successful registrations, that is, ragistrations with a TRE below 2 millimeters, and success rates above 91 percent when displacing the volume by a maximum of 6 millimeters and 17.2 degrees.

Alternative rendering approaches Raycasting is a popular rendering technique that provides a good tradeoff between accuracy and speed. Yet, several other approaches to rendering DRRs have been proposed. A *splatting*-based technique is presented in [Birkfellner et al., 2005]. However, splat rendering outperformed raycasting only when a majority of voxels was discarded and efficient anti-aliasing techniques were used. In [Weese et al., 1999] volume rendering based on *shear-warp factorization* combined with their *pattern intensity* similarity measure is used. Comparing their application to raycasting, they report a speed-up factor in the range between 3 and 7, depending on the resolution of the intermediate images, where the accuracy is still acceptable.

To address the DRR generation bottleneck, the *light field* rendering technique, which reduces the common runtime complexity from $O(n^3)$ to $O(n^2)$ by performing most calculation in a preprocessing step, is used in [Russakoff et al., 2003]. The approach is combined with the *mutual information* similarity measures and *best neighbor search* optimization.

Precomputation-based approaches Precomputation is a way to rationalize the DRR computation while entailing an increased memory complexity. A great part of the workload is shifted to a precomputation step, whose results are used to generate the actual DRRs during the online procedure.

Apart from [Russakoff et al., 2003] a precomputation-based approach is presented in [Freund et al., 2004]. *Projection fields* are used to store a certain range of prototype values for every DRR image pixel, which stem from a discrete variation of two rotational parameters. The actual DRRs are then computed using quadri-linear interpolation. This rendering approach showed to be about one hundred times faster than raycasting, while requiring 256 MB of RAM per radiograph (512×512) to align. It entailed only a minor loss of accuracy, yet is limited to a relatively small transformation parameter space.

Another interesting approach to DRR generation can be found in [Rohlfing et al., 2002]. Instead of computing *deterministic DRRs*, thus determining one scalar value per pixel, *probabilistic DRR*s (pDRRs), which record the distribution of volume intensities along the rays, are generated. This procedure conserves information on intermediate tissues, which is otherwise merged into a single intensity value. The *probabilistic mutual information* measure showed a promising convergence behavior when matching pDRRs and *deterministic DRRs*.

Yet, there are also approaches that get by without DRR generation, such as the method described in [Tomaževič et al., 2006]. Instead of projecting the patient scan to image space, a volume is reconstructed from a set of about 100 X-ray images. To cope with the poor quality of the reconstructed volume and the fact that the registration is multimodal, a novel similarity measure called *asymmetric multi-feature mutual information* is used. For a CT scan of a vertebra the results exhibited a mean TRE of 0.37 millimeters for successful registrations and a capture range, defined as the distance for which 95 percent of registrations are successful, of 7 millimeters.

2.1.2 Similarity Measuring and Optimization

Comparative studies on similarity measuring in the context of 2D/3D registration can be found in [Penney et al., 1998, Kim et al., 2007]. In order to solve the 2D/3D registration task, new similarity measures were developed, for example, *pattern intensity* [Weese et al., 1997], and the use of known measures from other fields was proposed, for example, the *correlation ratio* from probability theory [Roche et al., 1998].

In the nineties *mutual information* from information theory evolved as a very promising similarity measure for multi-modal image registration (for example, [Maes et al., 1997]). Recently, there has been an effort to enhance the robustness of the measure by incorporating spatial information using energy minimization based on Markov random fields [Zheng, 2007]. For the registration of a spine segment, the mean TRE generally amounted to 0.8 millimeters. For a maximum displacement from ground truth of 12 millimeters and degrees respectively, a success rate, that is, the percentage of registrations with a TRE below 1.5 millimeters, of 85 percent could be achieved.

Since we are carrying out an *intensity-based* registration, we do not consider *feature-based* approaches here. However, a hybrid approach to similarity measuring involving one fiducial marker is proposed in [Russakoff et al., 2003]. For spine images and a maximum initial TRE of 14 millimeters, the marker was capable of reducing the mean TRE by 0.2 millimeters and increasing the success rate (TRE below 2.5 millimeters) by 13 percent points to 99 percent.

Optimization strategies for use in image registration are evaluated in [Maes et al., 1999]. Different methods are compared with respect to their capability to maximize the *mutual information* of two images. Additionally, the benefit of multi-resolution strategies is evaluated. It was found that Powell's *direction set method* was most accurate and that the downhill-simplex, the conjugate-gradient, and the Levenberg-Marquardt method were most efficient.

In [Kubias et al., 2007a] an optimization strategy with three variable components, namely the optimizer (*local* or *global*), the scale and resolution, and the type of transformation parameters to estimate (*in-plane* or *out-of-plane*), which are parameterized according to the optimization stage, is presented. In the beginning, a global *adaptive random search* optimizer is used to estimate the *in-plane* parameters, which particularly affect the DRR, using a lower

DRR resolution and CT scale. At later stages a local *best neighbor search* is used to estimate the *out-of-plane* parameters at higher resolutions and scales.

Chapter 3

2D/3D Registration

3.1 Introduction

In medical imaging a number of modalities, which differ in the data they produce and the way it is gained, is used to acquire images of the human body. For medical applications it is often desirable to combine the capabilities of more than one imaging modality capturing one and the same object. Likewise, one may want to integrate images that were acquired at different points of time into the intervention in order to augment the information available to the surgeon. The process of spatially aligning two images is referred to as *registration*. It aims at establishing a correspondence between the coordinate systems of the respective datasets.

In [Maintz and Viergever, 1998] the following classification of registration methods is presented:

1. **Dimensionality**: The number of dimensions of the datasets involved (*2D/2D*, *2D/3D* and *3D/3D*). These depend on the modalities the images were acquired with.

2. **Nature of registration basis**: The image information that the alignment is based on. Essentially, there are *intrinsic* and *extrinsic* methods, where the former rely on information as arising solely from the patient's body, and the latter involve artificial objects which are mounted such that they appear on the resulting images. These objects comprise fiducial markers attached during invasive (e.g. screw markers applied to the bone) as well as non-invasive (e.g. skin markers) interventions. *Intrinsic* features can be divided into salient objects (*landmarks*), structures

obtained by segmentation (mostly surfaces), and voxel-based properties.

3. **Nature of transformation**: The simplest transformation is called *rigid*. It consists merely of translation and rotation. Adding scale and shear yields the *affine* transformation. If the transformation is capable of aligning parallel lines regardless of perspective distortion, it is called *projective*. The transformation with the most degrees of freedom is called *curved* or *elastic*, mapping curves to lines.

4. **Domain of transformation**: Specifies whether the transformation is applied to the whole image (*global*) or only parts of it (*local*).

5. **Interaction**: The level of interaction quantifies the amount of interaction of the user with the registration application. It comprises *interactive*, *semi-automatic*, and *automatic*.

6. **Optimization procedure**: This criteria tells whether the transformation parameters that are to be recovered can be computed directly or have to be searched for.

7. **Modalities involved**: Registering two images of the same modality is referred to as *monomodal*. If different devices are involved, it is called *multimodal*. Other options comprise the registration from *modality to model* and *patient to modality*.

8. **Subject**: Defines the origin of the images, where *intrasubject* (subject A to subject A), *intersubject* (subject A to subject B), and *atlas* registrations (subject A to an image generated from more than one subject) are possible options.

9. **Object**: The area of the imaged subject used to carry out the registration (e.g. head, lung, or abdomen).

The problem at hand can be broken down into a *global rigid 2D/3D* registration of the abdomen based on *intrinsic* (*voxel-based*) features. Due to differing dimensionality it is inherently *multimodal* (X-ray to CT). The transformation parameters are to be *searched for* during a preferably *automatic* procedure. This classification will be justified in the next sections.

3.1.1 Medical Imaging Modalities

Medical imaging modalities produce images of the human body. Maintz and Viergever distinguish *anatomical* and *functional* modalities [Maintz and Viergever, 1998]. The former capture mainly the morphology of the human body whereas the latter focus on metabolism. This second group includes SPECT (*single photon emission computed tomography*), PET (*positron emission tomography*), and fMRI (functional MRI). Since our interest lies in salient and rigid objects of the abdomen, we will restrict our considerations to *anatomical* modalities. These include CT (*computed tomography*), MRI (*magnetic resonance imaging*), X-ray (*fluoroscopy*), and ultrasonography. CT and MRI produce three-dimensional scans whereas X-ray and ultrasonography produce two-dimensional images of the human body.

Medical ultrasonography is used to visualize primarily non-rigid structures like organs, foetuses, and muscles by means of acoustic energy. Acoustic impedance of the tissue accounts for the resulting sonograph, which arises from a measurement of the reflected sound. Because of its low resolution and high level of noise, it does not satisfy our accuracy requirements. However, ultrasonography is a popular image acquisition technique since it is widely considered to be harmless for the human body.

Fluoroscopy devices use electromagnetic radiation to obtain information about rigid, but also non-rigid parts of the patient's body. If a region exhibits a certain amount of *radio-density*, it is detectable in the acquired data since there is a direct correlation between the radiation attenuation of material and the resulting image. Modern *fluoroscopes* use X-ray image intensifiers, which convert the X-ray radiation into a visible image allowing the use of lower X-ray doses. The voltage typically applied ranges from 50 to 150 kilovolts (kV) depending on the examination [Seibert, 2004]. Figure 3.1 shows a C-arm X-ray camera. The X-ray source of such cameras can be rotated around the patient's body to a certain extent, which adds great flexibility to intraoperative image acquisition. For the experiments described in this work a mobile C-arm camera was used to obtain different views of the patient.

Computed tomography is closely related to X-ray since the three-dimensional CT images are reconstructed from multiple scans produced by pairs of opposite X-ray sources and sensors rotating around the patient's body. The tube voltage is usually set to a single value between 80 and 140 kilovolts. The resulting three-dimensional dataset is a regular grid of real values, which represents a discretization of the imaged object's radiodensity and spans the volume coordinate system. These values are usually expressed by means of the Hounsfield scale, which refers to the radiodensity of water, being defined

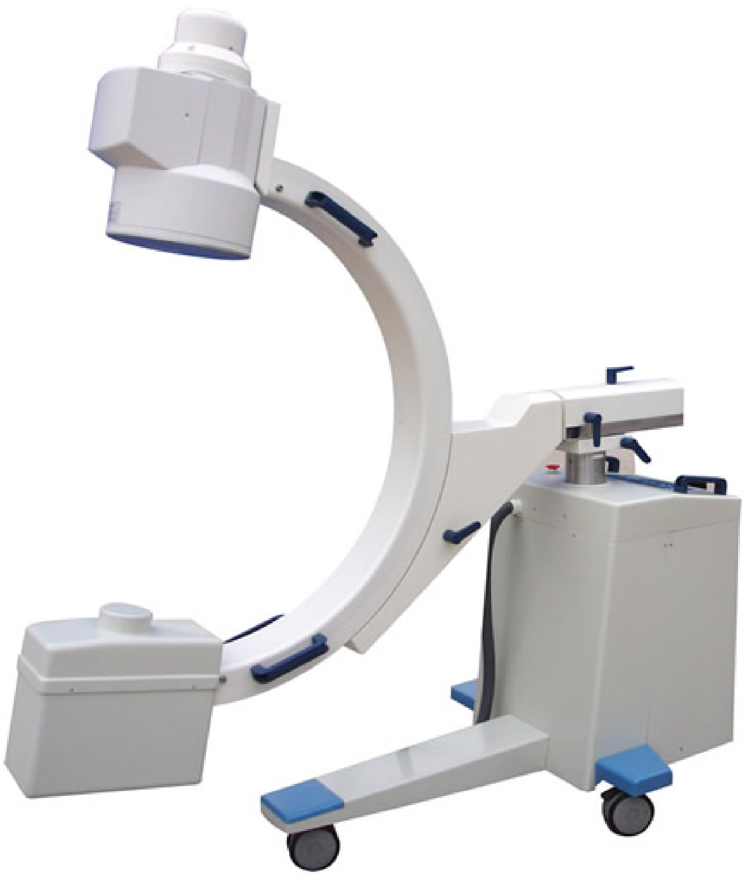


Figure 3.1: A C-arm X-ray camera

as zero (see Chapter 1 of [Jackson and Thomas, 2004] for details). A CT scanner is shown in Figure 3.2.

Figure 3.2: A CT scanner.

Magnetic resonance imaging derives images of the human body from the relaxation properties of hydrogen nuclei. The patient is exposed to a magnetic field which causes the nuclei to align in a defined manner. The relaxation of the nuclei, that is, the transition to the static condition, can be detected and forms the basis of the resulting MRI scans. In contrast to *computed tomography*, which is particularly suitable to acquire images of radio-dense structures of the body, such as bones and cartilages, *magnetic resonance imaging* provides a better means to capture tissues.

The registration task described in this work is carried out with CT scans and X-ray images and is thus *multimodal*. On one hand, the respective data can be acquired in many modern hospitals, on the other hand, a good registration result may be expected. CT scanning inherently produces high-resolution and high-contrast images with different types of tissue clearly distinguishable. Radio-dense structures, which are usually time-invariant and thus particularly suited for our registration task, are salient. By now no three-dimensional scan of the patient whose quality is similar to that of CT or MRI volumes can be produced during medical interventions. The Siemens ARCADIS Orbic 3D [Siemens, 2007] is capable of producing intraoperative volumes, yet the size ($12 \times 12 \times 12$cm) of the scans is still too small to be useful for our purposes. Additionally, it would take too long to acquire a full CT scan, for example, to determine the current position of a surgical instrument, and the radiation exposure for both the patient and the surgeon would be excessively high. Hence, the registration has to be based on two-dimensional images of the patient which can be produced without unnecessarily impeding the intervention. Modern C-arm X-ray cameras satisfy this condition. Furthermore, the aforementioned close relation between X-ray and CT in terms of image acquisition promises an inherently sound registration.

3.1.2 ***Intensity-Based*** Versus ***Feature-Based*** Registration

In contrast to [Maintz and Viergever, 1998], we draw a distinction between *feature-based* and *intensity-based* registrations. These terms are more common in recent literature on medical image registration.

Feature-based methods rely on the presence of salient features in the images to align. These features may be of an inherent or an artificial nature. Natural features comprise points, lines, surfaces, and contours. Artificial features include fiducial markers applied during invasive and non-invasive interventions, stereotactic frames, and the like. To recover salient objects from the images, some kind of segmentation is required. As soon as the desired features have been extracted from both images, they can be aligned potentially faster than when using the whole image information. However, it is necessary to carefully plan the image acquisition, the segmentation may introduce additional errors, and the user interaction often required is generally not desirable during medical interventions [Zöllei, 2001]. Yet, what eventually keeps us from using *feature-based* registration methods is our need for accuracy and robustness, which necessitates the use of invariant fiducial markers applied to the patient's body, for example, their bones, during an invasive intervention. Bearing in mind that the registration algorithm is to be used mainly in the context of routine examinations, the burden on the patient that the application of these markers involves is not justifiable with the registration speed-up and accuracy it may entail. In addition, such an intervention conflicts with our demand for a method that is inoffensive to the patient.

Intensity-based methods rely on the whole information conveyed by the raw pixel or voxel data of both datasets or parts of it. Their basic principle is to maximize a criterion measuring the intensity similarity [Roche et al., 1999]. In contrast to *feature-based* registration, there is no need for a segmentation step prior to the alignment. Hence, the registration can be conducted without any user interaction. Before assessing the similarity of both images and therefrom inferring the quality of the alignment, the dimensionality of the images has to be matched (see Section 3.3), involving a prior calibration of the imaging system (see Section 3.4). Thereafter *objective functions* provide a quantitative measure of the similarity of both images (see Section 3.5). An optimization procedure is used to find a minimizer or a maximizer of the *objective function* (see Section 3.6).

3.1.3 User Interaction

As mentioned in Section 3.1.2, user interaction with the registration system is generally undesirable. It may delay the intervention or require additional staff. The choice of an *intensity-based* approach to carry out the registration supports the request for a system which gets by with as little user interaction as possible. However, a successful registration is only feasible when the initial estimation of the images' correlation is sufficiently accurate. Hence, an initial adaption of the volume's pose by the user may have a significant influence on the convergence behavior of the registration application. Moreover, a coarse user-driven segmentation of regions that impair the registration may be beneficial. This leads us to not categorically ruling out user interaction and aiming at a maximal robustness of the registration instead.

3.2 Rigid-Body 2D/3D Registration

Since we use bones and rigid parts of the human body as a reference for the alignment and these structures are nearly invariant with respect to time, we can consider our task a *rigid-body* registration. Hence, the set of variable transformation parameters reduces to six, three parameters t_x, t_y, and t_z defining the translation along the x, y, and z axis in millimeters and three parameters r_x, r_y, and r_z, which define the rotation around the x, y, and z-axis and represent Euler angles in radians. Thus, the aim of this work is to find the translation matrix:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_x \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

and the rotation matrices:

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(r_x) & -sin(r_x) & 0 \\ 0 & sin(r_x) & cos(r_x) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.2}$$

$$\mathbf{R}_y = \begin{pmatrix} cos(r_y) & 0 & sin(r_y) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(r_y) & 0 & cos(r_y) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

$$\mathbf{R}_z = \begin{pmatrix} cos(r_z) & -sin(r_z) & 0 & 0 \\ sin(r_z) & cos(r_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.4}$$

Since the application of the rotations around the three coordinate axis is not commutative, we define their order to be *roll* (x-axis), *pitch* (y-axis), and *yaw* (z-axis). The complete rotation matrix $\mathbf{R}$ can thus be computed as follows:

$$\mathbf{R} = \mathbf{R}_z \mathbf{R}_y \mathbf{R}_x \tag{3.5}$$

To establish a common frame of reference, we introduce a world coordinate system which originates somewhere in the operating room. In our case this coordinate system is defined during camera calibration, where the pose of the X-ray camera is determined (see Section 3.4).

The operators $\mathbf{T}$ and $\mathbf{R}$ place the volume into the world coordinate system, thus mapping a point $\tilde{\mathbf{p}}_{\text{vol}} \in \mathbb{R}^4$ from the coordinate system defined by the CT volume (in millimeters), that is, the object coordinate system, to a point $\tilde{\mathbf{p}}_{\text{world}} \in \mathbb{R}^4$ in the world. Note that vectors with a tilde are given in homogeneous coordinates. The volume should be rotated around a defined point, by default its center. Hence, we translate the volume's center of rotation into the world coordinate system's origin using the inverse of matrix $\mathbf{C}$ before we carry out the rotation. Only when the volume has been rotated and its original position has been restored using $\mathbf{C}$, we apply the actual translation $\mathbf{T}$. We refer to this chain of transformations as *model transformation*, which stems from the classical rendering pipeline. It is described as a whole by the matrix $\mathbf{M}$, which represents the rigid-body transformation in three-dimensional space:

$$\tilde{\mathbf{p}}_{\text{world}} = \mathbf{M}\tilde{\mathbf{p}}_{\text{vol}} \tag{3.6}$$

$$= \mathbf{TCRC}^{-1}\tilde{\mathbf{p}}_{\text{vol}} \tag{3.7}$$

Now each point in the world is mapped to the coordinate system established at the X-ray camera's source. We call this process *viewing transformation.* The matrix **V** encodes the position and orientation of the camera while it produced the radiograph to be aligned.

$$\tilde{\mathbf{p}}_{\text{cam}} = \mathbf{V}\tilde{\mathbf{p}}_{\text{world}} \tag{3.8}$$

Subsequently, we make use of the knowledge about the *internal camera parameters* gained during camera calibration (see Section 3.4) and packed into the perspective projection matrix **P**. It is the *perspective transformation* followed by the *perspective division* that realizes the mapping of a point in camera space into image space:

$$\tilde{\mathbf{p}}_{\text{img}} = \mathbf{P}\tilde{\mathbf{p}}_{\text{cam}} \tag{3.9}$$

$$\mathbf{p}_{\text{img}} = \begin{pmatrix} \tilde{p}_{\text{img}}.x/\tilde{p}_{\text{img}}.w \\ \tilde{p}_{\text{img}}.y/\tilde{p}_{\text{img}}.w \end{pmatrix} \tag{3.10}$$

where $\mathbf{p}_{\text{img}}$ represents a point in Cartesian image coordinates.

The whole chain of transformations from a point in the CT volume to a point in the X-ray image can thus be represented as follows:

$$\tilde{\mathbf{p}}_{\text{img}} = \mathbf{PVTCRC}^{-1}\tilde{\mathbf{p}}_{\text{vol}} \tag{3.11}$$

followed by the perspective division shown in (3.10).

The only transformation that is considered as variable, thus not being trivial to recover, is the *model transformation* **M**. The others are treated as constants in the course of aligning one X-ray image and a CT volume.

3.3 Digitally Reconstructed Radiographs

3.3.1 Introduction

In order to align the coordinate systems of a CT volume and an X-ray image, their dimensionality has to be matched. From (3.11) we know how a point within the CT volume maps to a point in X-ray image space as well as all intermediate steps. If we use the CT scan as a model of the patient's body and emulate the X-ray image acquisition by applying the known chain of transformations to points in the volume, we obtain a virtual X-ray image commonly referred to as *digitally reconstructed radiograph* (DRR).

If the *viewing transformation* and the *perspective transformation* are considered to be constant, namely in accordance with the effective attributes of the X-ray camera, only the *model transformation* remains variable. Hence, the nature of the DRR depends on the choice of the translation parameters t_x, t_y, and t_z and the rotation parameters r_x, r_y, and r_z. Assuming a bijection between these parameters and a DRR within certain bounds[1], we can state the following: If we succeed in aligning the X-ray image and the DRR, we have found the set of model transformation parameters sought-after.

To adequately simulate the process of X-ray acquisition, we have to understand what the image intensities of the radiograph result from. The X-ray intensity I reaching the detector plane or the image intensifier at a point $(u, v) \in \Omega$ in image space can be expressed by means of the following equation [Wein et al., 2005]:

$$I(u,v) = \int_0^{E_{\max}} I_0(E) \exp\left(-\int_{r(u,v)} \mu(x,y,z,E)dr\right) dE \tag{3.12}$$

where $I_0(E)$ denotes the incident X-ray energy spectrum, $r(u, v)$ a ray from the X-ray source to the image point (u, v), and $\mu(x, y, z, E)$ the energy dependent attenuation at a point (x, y, z) in space. The second integral represents the attenuation of an incident energy $I_0(E)$ along the ray $r(u, v)$. The integral over E incorporates the energy spectrum of X-ray cameras. For the sake of efficient computation, the X-ray source is mostly modeled to be monochromatic and the attenuation to act upon an effective energy E_{eff}:

$$I(u,v) \approx I_0(E_{\text{eff}}) \exp\left(-\int_{r(u,v)} \mu(x,y,z,E_{\text{eff}})dr\right) \tag{3.13}$$

Taking into account that X-ray devices usually provide the logarithm of the measured X-ray intensities, we can state the following correlation:

$$\text{xray}(u,v) \approx ln(I_0(E_{\text{eff}})) - \int_{r(u,v)} \mu(x,y,z,E_{\text{eff}})dr \tag{3.14}$$

In other words, the intensities of the final X-ray image can be mapped linearly to a simple accumulation of the energy attenuation along the ray. More

[1]E.g. r_x and $r_x{+}360°$ will produce the same DRR. Yet, such rotations can be neglected.

elaborate similarity measures do not require an identity relationship between the two signals and are thus insensitive to global additions and multiplications applied to one of the signals (see Section 3.5). Hence, the basis for the DRR generation can be formulated as follows:

$$\mathrm{drr}(u, v) = \int_{r(u,v)} \mu(x, y, z, E_{\mathrm{eff}})dr \tag{3.15}$$

A DRR intensity has now become a quantitative measure for the total attenuation of the incident energy along one viewing ray.

When mapping the highest measured X-ray intensity to the maximum intensity value (e.g. white) and clamping the other intensities to the range of available intensity values, the DRR, according to the above formulation, represents the inverted version of the X-ray image.

3.3.2 DRR Rendering

Digitally reconstructing a radiograph is tantamount to rendering a volume, thus producing a two-dimensional view of a three-dimensional model. However, we are dealing with a greatly simplified version of the rendering equation [Kajiya, 1986] since lighting can be generally neglected. There are essentially two approaches to volume rendering: *Image-order* and *object-order* rendering.

Image-order approaches iterate through the image space and try to find all contributions of the volume to single pixels. The most widely used technique is *volume raycasting* (described in more detail in Section 3.3.3). The performance is more dependent on the number of pixels in the image than on the resolution of the volume. This may be an advantage for volumes with a high number of voxels as well as a drawback for sparsely populated volumes. *Image-order* approaches are particularly suited for GPU-based rendering by means of fragment shaders since they inherently pursue a pixel-based computation strategy (see Section 4.4).

Object-order approaches, on the other hand, iterate through the elements of the volume and determine their contribution to the intensities of certain image pixels. A popular *object-order* technique is *splatting* [Westover, 1991], where the volume's voxels are splatted onto the image plane, thus leaving a footprint in the image to produce. In contrast to *image-order* rendering, the performance of these approaches is more sensitive to the number of volume elements. However, sequential access to the voxels in memory may speedup the computation.

3.3.3 DRR Raycasting

Volume raycasting is a direct *image-based* volume rendering technique. It is based on the work of James F. Blinn on the interaction of light and matter [Blinn, 1982] and James T. Kajiya [Kajiya and Herzen, 1984]. Marc Levoy lay the foundation for volume raycasting as it is known today [Levoy, 1988].

A viewing ray is cast from the observer eyepoint through every pixel in the image. The volume it penetrates is sampled at certain points along the ray to produce a final color value for the pixel. A raycasting algorithm typically involves the following steps:

- **Interpolation**

  Volumes are usually represented by a discrete regular grid of color or density values. To obtain such a value for an arbitrary point within the volume, a set of voxels in the neighborhood and their respective distance from the sampling position are used to generate, that is to say, *interpolate* a new value. Common methods comprise *nearest neighbor* and *linear interpolation.*

- **Classification**

  Classification describes the phase of assigning material properties to a point in the volume. Typically every sampling position is provided with an opacity or a color value. Most often so-called *transfer functions* serve as a look-up table for these values. Levoy initially proposed a classification prior to interpolation (*pre-classification*), which is known to produce faulty colors and is most often replaced by a classification after the interpolation step (*post-classification*).

- **Shading**

  During *shading* a color value is assigned to a sampling point in the volume. This color represents the light which is reflected at that point into the direction of the observer. One of the most popular shading techniques is Phong's *reflection model* [Phong, 1973].

- **Composition**

  The process of computing a final color value for one image pixel from the samples is referred to as *composition.* It is the step where a discrete version of the rendering equation is evaluated, either from front to back or from back to front.

As already mentioned, we are dealing with a greatly simplified version of volume rendering. There is only one point light source, namely at X-ray camera's source, whose light is attenuated along the viewing rays.

To compute the total attenuation of the initial energy, we need to accumulate the attenuation coefficients μ_{world} for every ray by evaluating the discrete version of (3.15). According to [Rohlfing et al., 2002] we formulate:

$$\mathbf{s}(u,v) = \delta \frac{\mathbf{p}_{\text{dest}}(u,v) - \mathbf{p}_{\text{src}}}{\|\mathbf{p}_{\text{dest}}(u,v) - \mathbf{p}_{\text{src}}\|} \tag{3.16}$$

$$\mathbf{p}_i(u,v) = \mathbf{p}_{\text{src}} + (i + 0.5)\mathbf{s}(u,v) \tag{3.17}$$

$$N(u,v) = \frac{\|\mathbf{p}_{\text{dest}}(u,v) - \mathbf{p}_{\text{src}}\|}{\delta} \tag{3.18}$$

$$\text{drr}(u,v) = \delta \sum_{i=0}^{N(u,v)} \mu_{world}(\mathbf{p}_i(u,v)) \tag{3.19}$$

where $\mathbf{s}(u,v)$ is the step vector (of length δ) on the ray from the X-ray source $\mathbf{p}_{\text{src}}$ through a discrete point $\mathbf{p}_{\text{dest}}(u,v)$ on the image plane. The set of sampling positions on that ray is denoted by $\mathbf{p}_i(u,v)$ with $i = 0, \ldots, N(u,v)$. Note that all vectors and lengths refer to the Cartesian world coordinate system.

To get the radiodensity for an arbitrary sampling position $\mathbf{p}_i(u,v)$ within the volume tri-linear *interpolation* of the eight nearest voxels' density values is used. It was mentioned earlier that the same principles underlie the acquisition of X-ray and CT data. The elements of both resulting datasets attribute to the radio-density of the imaged object, which means that the voxels' radio-densities provide a good approximation of the local relative attenuation of radiation. Consequently, the value obtained from the interpolation step can be classified as-is. Shading can be generally neglected. Ultimately, the final pixel intensity is composed as the sum of the samples' attenuation coefficients times the step length.

In order to simulate the acquisition of an X-ray image correctly, the whole chain of transformations in (3.11) has to be reflected in the raycasting algorithm. Figure 3.3 shows a schematic X-ray imaging setting for acquiring radiographs of one patient for three different camera poses. It was reconstructed from the three radiographs that can be seen in the image during camera calibration (see Section 3.4). The six translation and rotation parameters we are trying to recover are incorporated by the position and orientation of the volume in the frame of reference, that is, the world coordinate

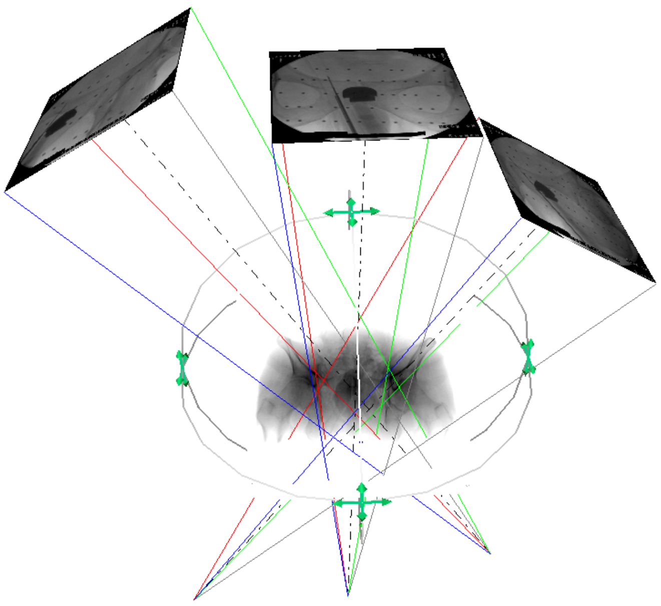


Figure 3.3: The schematic setting of X-ray image acquisition that is emulated in order to generate the DRRs.

system. Likewise, the position of the ray source and the normal vector of the image plane take into account the pose of the X-ray camera whereas the distance between them considers the characteristics of the perspective projection. Hence, we reformulate:

$$\mathrm{drr}(\mathbf{x}, u, v) = \delta \sum_{i=0}^{N(u,v)} \mu_{vol}(\mathbf{M}(\mathbf{x})^{-1}\tilde{\mathbf{p}}_i(u, v)) \tag{3.20}$$

where the vector $\mathbf{x}$ contains the six rigid-body transformation parameters incorporated by the inverse model matrix $\mathbf{M}^{-1}$, which maps any point in the world to a point in the coordinate system of the volume. The function μ_{vol} provides the attenuation coefficients for these points. The pose as well as the projection properties of the camera are reflected in $\tilde{\mathbf{p}}_i$, which is considered a constant within the function drr. Figure 3.4 depicts the DRRs produced by the setting in Figure 3.3, the corresponding X-ray images, and the color-coded addition of the respective original (red channel) and reconstructed (green channel) radiographs.

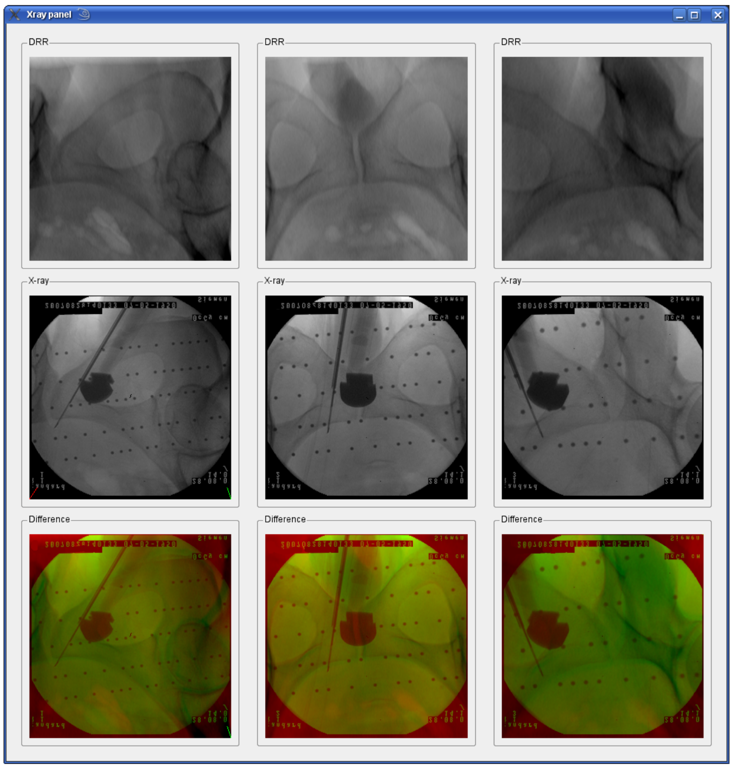


Figure 3.4: A comparison of the DRRs (shown in the upper row) and the X-ray images (shown in the middle row) produced by the setting in Figure 3.3. The last row depicts the addition of the DRRs (red channel) and the radiographs (green channel). Yellow hints at a good match between them.

3.4 Camera Calibration

In order to simulate the acquisition of a radiograph, the exact characteristics of the imaging setting have to be known. These characteristics comprise:

- The pose (position and orientation) of the camera source with respect to the common frame of reference, referred to as *extrinsic* camera parameters.
- The perspective projection and distortion properties of the camera, referred to as *intrinsic* camera parameters.

The *extrinsic* parameters are incorporated by the viewing transformation matrix $\mathbf{V}$. They define a rigid-body transformation from the world coordinate system to the coordinate system established at the X-ray camera's source. The *intrinsic* parameters are reflected by the projection matrix $\mathbf{P}$, thus specifying the transformation from camera coordinates to image coordinates. While those parameters can be determined during an off-line procedure, that is, before any intervention, the *extrinsic* parameters usually have to be recovered ad hoc, that is, for every X-ray image acquired and used for the registration. They can only be retrieved beforehand if the camera's position is fixed relative to the world or it can be moved in a predefined way during the operation. In other words, we have to be able to reproduce certain camera poses which correspond to known sets of *extrinsic* parameters. Since portal X-ray cameras are usually mobile and subject to potential bumps against them, we carry out an online calibration for each image acquired. Alternatively, the pose of camera can be determined by means of a C-arm tracking setup.

The implementation of camera calibration was not part of this thesis. Instead, we use a calibration procedure which essentially relies on the methods described in [Tsai, 1992]. A set of coplanar points visible on the image is used to compute the camera parameters. To obtain these points, a radio-opaque board with spheric metal markers applied to it on both sides is introduced to the image space. The positions of the markers on the board are known, that is, determined from a CT scan of the board. The respective positions of the markers on the X-ray images can be recovered by means of segmentation since they stand out clearly from the rest of the image, as can be seen in Figure 3.5(a). It is conducted using variational methods [Chan and Shen, 2005]. First a total variation L^1 filter (TV-L^1) which is capable of extracting homogeneous regions of a certain scale, that is, the size of the markers on the image, is applied. To find all circle-like regions and discard other objects

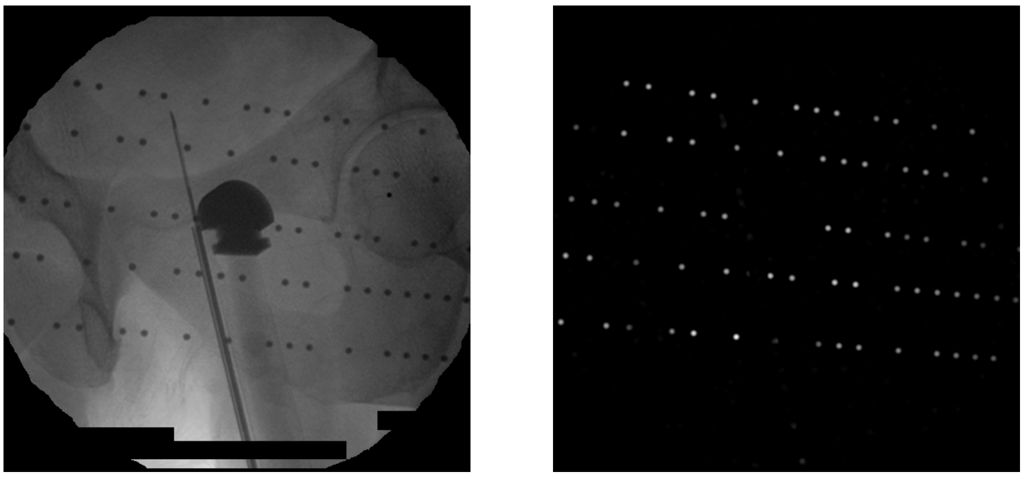

(a) (b)

Figure 3.5: The left image shows a radiograph with the spheric calibration markers clearly visible. The right image shows the result of the marker segmentation.

such as surgical needles, the Eigenvalues of the image's Hessian are taken into account. An absolute value of the smaller Eigenvalue which is still large suggests a circular structure in the image. A final thresholding step yields an image as shown in Figure 3.5(b). As soon as the correspondence between the markers in the image and in the world has been established, the chain of transformations between them can be recovered.

In order to obtain the *intrinsic* parameters, a set of views of the calibration board at different poses, that is, not being fixed to the operating table, is used. First the parameters are approximated by a closed-form solution which disregards lens distortion. Then the reprojection error is minimized using *gradient descent* yielding the final values for the *intrinsic* parameters. These are used to undistort the X-ray images and to compute the *extrinsic* parameters for single radiographs, which are determined in a similar manner as above. To do so, the calibration board is mounted under the operation table. Since the board is fixed during the entire intervention, we use it as a reference for setting up the world coordinate system in the operating room. Hence, the positions of the markers in the world are known too. Initially an approximation of the mapping between the markers' positions in the CT scan of the board, that is, world coordinates, and the respective positions in the camera coordinate system, which is represented by the viewing transformation matrix $\mathbf{V}$, is calculated. This is possible since the mapping from points in the world to points in the image, that is, the result of the matrix multiplication $\mathbf{PV}$, and the projection from camera coordinates to image coordinates, that is, the projection matrix $\mathbf{P}$, are known. Eventually the reprojection error is again minimized using gradient descent optimization.

3.5 Intensity-Based Similarity Measuring

3.5.1 Introduction

Similarity measures or *objective functions* provide a means to assess the similarity of two signals. They are of the following form:

$$y = f(\mathbf{x}) \qquad f : \mathbb{R}^N \to \mathbb{R} \tag{3.21}$$

The dependent variable y, sometimes referred to as *energy*, is a dimensionless quantity expressing the similarity of the signals. The vector $\mathbf{x}$ contains the set of variables that is known to have an influence on the similarity. Usually the graph of f is not known and only local evaluations are available. During optimization (Section 3.6) parts of the objective function are recovered to find extrema of y. We will refer to our task as a minimization problem throughout this work. Hence, we define f to be positive and to decrease towards zero for an increasing similarity of the signals. To emphasize this, we will use the term *cost function* when referring to f.

For the registration problem at hand, f can be considered as follows:

$$f(\mathbf{x}) = \mathrm{sm}(\mathrm{drr}(\mathbf{x}), \mathrm{xray}) \tag{3.22}$$

with sm being the similarity measure. While drr is dependent on the model transformation parameters $\mathbf{x}$, the original radiograph xray is a constant in the context of f.

The choice of the similarity measures is dependent on the nature of the registration basis (see 3.1.2). Since we rely on *intensity-based* methods, we have to use cost functions that process essentially the whole information conveyed by the two images in order to compute their similarity. Furthermore, the imaging modalities involved in the registration play a key role in the choice of similarity measure. The physical model implemented by a modality is reflected in the resulting images. These models may vary greatly for different imaging devices (see Section 3.1.1). Each *intensity-based* similarity measure represents a certain type of relationship between the image intensities. These relationships reflect different interpretations of similarity. A similarity measure will perform the better the more the images actually exhibit the assumed relationship. Hence, it is crucial to the success of the registration to find a similarity measure which adequately incorporates the peculiarities of the imaging modalities involved and their interrelationship.

Roche et al. distinguish the following hypotheses underlying known similarity measures [Roche et al., 1999]:

- **Identity relationship**

  These measures assume the images I and J to be identical when they are perfectly aligned: $I = J$. Most of them are also insensitive to a global offset, though: $I = J + \beta$ ($\beta \in \mathbb{R}$). Popular examples of this class are *sum of squared differences* (SSD) and *sum of absolute differences* (SAD).

- **Affine relationship**

  Measures based on an *affine relationship* assess images I and J to be similar when they are related by a linear mapping: $I \approx \alpha J + \beta$ ($\alpha, \beta \in \mathbb{R}$). This assumption usually holds for images acquired by one imaging modality. *Normalized cross correlation* (NCC) is a widely used measure of this class.

- **Functional relationship**

  The term *functional relationship* comprises any kind of functional mapping between the two images: $I \approx f(J)$. This is a very general definition containing the two hypotheses above.

- **Statistical relationship**

  Similarity measures based on *statistical relationship* involve methods from information theory and the computation of image statistics. For example, *mutual information* provides a quantitative measure of the amount of information an image I, considered a random variable, reveals about an image J. It proved reliable also for *multimodal* registration [Maes et al., 1997, Zöllei, 2001].

Since similarity measures that are evaluated in a pixel-wise fashion such as SSD and NCC inherently match the GPU programming model well and are thus not affected by its limitations (see Chapter 4), we will focus on these measures in the following.

3.5.2 Sum of Differences

This class of similarity measures is based on a per-pixel computation of intensity differences. It assumes an *identity relationship* between the intensities of two correctly aligned images. This assumption hardly holds for images produced by different imaging modalities, but may be adequate for *monomodal* registrations.

To prevent differences with different signs from cancelling each other out, they have to be provided with an uniform sign, which ensures a proper accumulation of differences. The *sum of squared differences* (SSD) measure does so by squaring the intensity difference of each pair of pixels at a position $(u, v) \in \Omega$ in image space:

$$\mathrm{SSD}\,(I, J) = \frac{1}{N} \sum_{(u,v)\in\Omega} \left(I(u, v) - J(u, v)\right)^2 \tag{3.23}$$

where N denotes the total number of pixels.

To reduce the impact of single outliers, for example, salt and pepper noise, on the measure, the *sum of absolute differences* (SAD) may be used:

$$\mathrm{SAD}\,(I, J) = \frac{1}{N} \sum_{(u,v)\in\Omega} \left|I(u, v) - J(u, v)\right| \tag{3.24}$$

A simple example shall demonstrate the limited applicability of these measures. Let us consider a hypothetical imaging modality producing binary images. We may want to align it with a reconstructed image. For the sake of simplicity, the emulation procedure produces the inverse of the original image for a congruent setting, as it is the case for our DRR formulation in Section 3.3. When aligned, the two images will exhibit the highest possible intensity difference. Hence, the similarity measure will report a better match for *any* emulated image other than the correctly aligned one.

Despite the aforementioned constraints, the use of sums of differences is popular since they are easy to implement and quick to compute. In particular, these measures can be evaluated along with an *image-order* generation of DRRs (see Section 3.3.2).

3.5.3 Normalized Cross Correlation

Normalized cross correlation (NCC) or the *correlation coefficient* verifies the existence of an *affine relationship* between images. It provides information about the extent and the sign by which two random variables are linearly related:

$$\mathrm{NCC}\,(I,J) = \frac{1}{\sigma_I\sigma_J}\frac{1}{N}\sum_{(u,v)\in\Omega}\left(I(u,v)-\mu_I\right)\left(J(u,v)-\mu_J\right) \tag{3.25}$$

$$= \frac{\sum_{(u,v)\in\Omega}\left(I(u,v)-\mu_I\right)\left(J(u,v)-\mu_J\right)}{\sqrt{\sum_{(u,v)\in\Omega}\left(I(u,v)-\mu_I\right)^2}\sqrt{\sum_{(u,v)\in\Omega}\left(J(u,v)-\mu_J\right)^2}} \tag{3.26}$$

It has two extrema occurring if $I(u,v) = \alpha J(u,v) + \beta$ for all $(u,v) \in \Omega$ with $\alpha, \beta \in \mathbb{R}$, where a positive value of α, that is, an increasing linear relationship yields a maximum of 1 and a negative value of α, that is, a decreasing linear relationship yields a minimum of -1. Both extrema attest to a full linear relationship between the images I and J. Hence, *normalized cross correlation* is insensitive to global variations of brightness and contrast between the input images. What is more, even an inversion of an image's intensities does not prevent the measure from detecting a potential relationship, which is the condition for the applicability of the inverse DRR formulation in Section 3.3.

In contrast to a simple summation of pixel differences, NCC, according to formulation (3.26), requires the pixel data to be considered at least twice, once for the computation of the arithmetic means μ_I and μ_J and again for the computation of the distances from the means. Note that the expressions in the numerator and the denominator can be evaluated in a parallel manner. The following alternative formulation can prevent this:

$$\mathrm{NCC}\,(I,J) = \frac{\sum_{(u,v)\in\Omega} I(u,v)J(u,v) - N\mu_I\mu_J}{\sqrt{\sum_{(u,v)\in\Omega} I(u,v)^2 - N\mu_I^2}\sqrt{\sum_{(u,v)\in\Omega} J(u,v)^2 - N\mu_J^2}} \tag{3.27}$$

The pixel intensities can be summed up along with the summations above and beneath the slash, which again can be evaluated simultaneously, to eventually compute the means and evaluate the whole formula. Yet, the potential proximity of the minuends and the subtrahends may introduce an additional roundoff error (cp. Sections 1.3 and 14.1 of [Press et al., 1992]), especially when I and J are largely homogeneous.

When assessing the similarity of a DRR and an X-Ray image, we usually want to exclude certain regions that would impair the result, that is, parts that are only visible on the X-ray image, such as the camera's aperture around the edges, calibration targets, and medical instruments. This can be achieved by using pixel weights which specify the contribution that an image position makes to the measure. Hence, we reformulate

$$\mathrm{NCC}\left(I, J, w\right) = \frac{1}{\sigma_{wI}\sigma_{wJ}} \frac{1}{\Sigma_w} \sum_{(u,v)\in\Omega} w(u,v)\left(I(u,v) - \mu_{wI}\right)\left(J(u,v) - \mu_{wJ}\right) \tag{3.28}$$

where w may be a binary mask on every pair of corresponding pixels, that is, $w(u,v) \in \{1,0\}$, or a weight function, that is, $w(u,v) \in [0,1]$. For the sake of flexibility we chose the latter, even though it may be less efficient to compute. However, we may want to reduce the impact of certain image regions without completely discarding them. Σ_w denotes the sum of all weights in Ω:

$$\Sigma_w = \sum_{(u,v)\in\Omega} w(u,v) \tag{3.29}$$

Naturally, the use of pixel weighting also influences the mean and standard deviation of the X-ray and DRR intensities:

$$\mu_{wI} = \frac{1}{\Sigma_w} \sum_{(u,v)\in\Omega} w(u,v) I(u,v) \tag{3.30}$$

$$\sigma_{wI} = \sqrt{\frac{1}{\Sigma_w} \sum_{(u,v)\in\Omega} w(u,v)\left(I(u,v) - \mu_{wI}\right)^2} \tag{3.31}$$

To make the formulation of normalized cross correlation comply with the definition of the cost function f in the beginning of this section, we introduce:

$$\mathrm{NCC}'\left(I, J, w\right) = 1 + \mathrm{NCC}\left(I, J, w\right) \tag{3.32}$$

which maps NCC to the range $[0,2]$ and yields lower values for more similar, that is, stronger correlated images whose intensities vary inversely, as it is the case for the X-ray images and the DRRs according to our formulation in Section 3.3.1.

When dealing with more than one X-ray view, which is virtually always the case, we calculate one common similarity measure as follows:

$$\mathrm{NCC}'_{\mathrm{n_{views}}}\left(\begin{bmatrix} I_i & J_i & w_i \end{bmatrix}_{i=1}^{n_{\mathrm{views}}}\right) = 1 + \frac{\sum_{i=1}^{n_{\mathrm{views}}} \mathrm{NCC}'\left(I_i, J_i, w_i\right)}{n_{\mathrm{views}}} \tag{3.33}$$

In fact, we are not dealing with a *monomodal* registration, what cross correlation usually proofs useful for [Roche et al., 1999]. However, the same principles underlie the acquisition of CT scans and X-ray images, where the effective energies used by tomographs and fluoroscopes differ. This may cause different ratios between the recorded attenuation of bones and soft-tissue. One way to address this is to discard all attenuation coefficients within the CT volume beneath a certain threshold [Penney et al., 1998].

What particularly encourages the use of the NCC measure apart from its speed is the pixel-wise evaluation it is based on. For reasons qualified in Chapter 4, this kind of measure can be evaluated very efficiently by means of programmable graphics hardware.

3.5.4 Image Region of Interest

There are parts in the X-ray image whose intensities are not related to those of the DRRs, even if both images are perfectly aligned. Those parts distort the similarity measure and may thus impair the accuracy and robustness of the registration. Hence, we use the weight function w to mask out the following sources of imprecision:

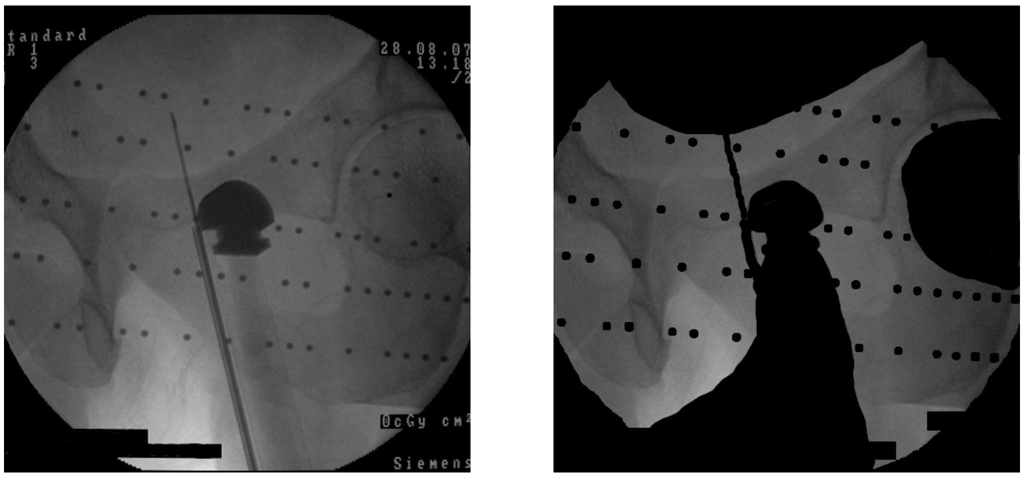
Aperture of the X-ray camera The aperture visible on the edges of the images is almost the same for all radiographs we used. Hence, one mask for the aperture and the writings on the radiographs can be used for all X-ray images if it grants a little tolerance.

Medical instruments We are currently working on the automatic segmentation of medical instruments that are visible on the radiographs, such as surgical needles and ultrasonic probes. For the time being, a mask is drawn manually for each radiograph.

Calibration targets An image showing the calibration targets is a byproduct of camera calibration, where the positions of these markers have to be determined. After inverting, thresholding, and dilating the images, they can be used to mask out the calibration targets from one radiograph each.

Large areas of deformable tissue Currently we do not have a way to automatically identify areas containing merely soft tissues such as intestines, whose pose may vary over time and which are not reliable to guide the registration. In the future we want to evaluate the effect of completely excluding tissues below a certain radio-density threshold. Furthermore, these areas might be segmented automatically. For now, however, masks for these areas are drawn manually.

The final weight mask intensities $w(u,v)$ constitute the minimum intensity of all four masks at position (u,v). Pixels that are discarded by w neither contribute to the mean of the image intensities nor to the final NCC measure. Figure 3.6(a) shows an exemplary radiograph including the artifacts discussed above. The result of masking all potential sources of imprecision out of the image by means of w can be seen in Figure 3.6(b).



(a) (b)

Figure 3.6: The left image shows a radiograph containing different kinds of artifacts such as an ultrasonic probe, a surgical needle, calibration targets, and the camera's aperture. The right image shows the radiograph after the application of w.

3.6 Optimization

3.6.1 Introduction

Optimization describes the process of seeking a set of model transformation parameters $\mathbf{x}$ which induces the cost function f to take on a minimum. The requirement of finding the global minimum is usually over-optimistic, yet it is mostly sufficient to find the smallest value of f in a subset $\mathbf{X} \subset \mathbb{R}^N$ of the independent parameter space, which is then referred to as the *minimizer* of

f in $\mathbf{X}$:

$$\arg\min_{\mathbf{x}\in\mathbf{X}} \mathrm{sm}\left(\mathrm{drr}(\mathbf{x}), \mathrm{xray}\right) \tag{3.34}$$

The subset $\mathbf{X}$ can be considered as a limited range of poses that the patient may take in a surgical setting.

Unfortunately, the cost function's graph can not by retrieved easily and it is too complex to compute the extrema directly, that is, computing the zeros of the derivative $\frac{\partial f}{\partial \mathbf{x}}$ obtained by symbolic differentiation. Hence, the progression of the cost function has to be partly reconstructed from a set of local function evaluations. Evaluating f at a position $\mathbf{x}$ is potentially costly since it involves computing a DRR and determining its similarity to the X-ray image. Hence, it is crucial to choose an optimization strategy that heads for the desired minimum quickly and terminates at the right time, according to a good tradeoff between registration time and accuracy. Most optimization approaches allow for a user-definition of the so-called *tolerance*, for example, by setting a threshold for f or $\|\Delta\mathbf{x}\|$. If f drops beneath a lower bound or the changes that are to be applied to $\mathbf{x}$ are too small, the optimization algorithm terminates.

When stepping through the parameter space $\mathbf{X}$ of a non-smooth cost function, that is, a function that does not decrease monotonically towards its minimum, there is always the risk of getting trapped in a local minimum $\mathbf{x}_{lm}$:

$$f(\mathbf{x}_{lm}) < f(\mathbf{x}) \qquad \forall \mathbf{x} \in \mathbf{X} : \|\mathbf{x} - \mathbf{x}_{lm}\| < \delta \tag{3.35}$$

for some $\delta \in \mathbb{R}$. One way to address this is to smooth the cost function as discussed in Section 3.6.4.

There are essentially two classes of optimization strategies, namely *gradient-based* and *non-gradient-based*. Approaches from the former group use local derivatives of the cost function as a hint for the direction towards the minimum, while the latter rely solely on values of f. *Non-gradient* methods used for image registration include the *Powell-Brent direction set method* [Powell, 1964, Brent, 1973], the *downhill simplex* method [Nealder and Mead, 1965], and a *best neighbor* search. The first two proved to be robust but did not keep up with *gradient-based* methods in terms of speed [Wein, 2003, Zöllei, 2001]. Methods involving gradients include the *Levenberg-Marquardt method* [Levenberg, 1944, Marquardt, 1963] as well as the popular *gradient descent* and *steepest descent* strategies.

In this work we will restrict our focus to *gradient-based* methods since we consider the computation of gradients one of the main targets for the optimization of the registration in terms of speed. In *algorithmic differentiation* (see Chapter 5) we found a means to compute the cost function's gradient both accurately and efficiently.

3.6.2 Gradient Descent and Steepest Descent

Gradient descent is based on the fact that the gradient vector at a point $\mathbf{x}_i$ always points in the direction of the steepest increase of f. Consequently, the parameter vector is updated proportionally to the negative gradient vector at that point in order to approach the next local minimum:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \lambda \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i) \tag{3.36}$$

where $\lambda \in \mathbb{R}^+$ denotes the *step-size* or *learning rate*. It controls the size of the steps to be taken according to the local gradient. The choice of this value is crucial to the success of the optimization procedure. If chosen too small, the search may easily get stuck in a local minimum or take excessively long to converge. If chosen too large, the algorithm may repeatedly miss or oscillate around the minimum sought-after. Possible approaches to deal with this situation are a local adaption or a gradual decrease of the learning rate at later stages (*annealing*). The algorithm is usually specified to terminate when the length of $\frac{\partial f}{\partial \mathbf{x}}$ drops beneath a certain threshold, and hence no major enhancements may be expected anymore. An alternative is to compute f at the updated position $\mathbf{x}_{i+1}$ and decide whether the alignment is accurate enough.

The principle of the *steepest descent method* is similar to *gradient descent*, yet it works without learning rate, which relieves us from the awkward situation of choosing an adequate value for λ. The method is based on a line search along the direction defined by the gradient vector:

$$\phi(v) = f\left(\mathbf{x}_i - v \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i)\right) \tag{3.37}$$

$$\arg \min_{v \in [a,b]} \phi(v) \tag{3.38}$$

where a and b are defined such that

$$\mathbf{x}_i - v \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i) \in \mathbf{X} \tag{3.39}$$

where the bounds of $\mathbf{X}$ may be defined by the user. For determining the scalar v which minimizes f, for example, the *bisection* method may be used.

It first finds a middle point $c \in (a, b)$ such that $\phi(c)$ is smaller than $\phi(a)$ and $\phi(b)$. This is the invariant of the algorithm. If such a point can be found, the algorithm is guaranteed to converge linearly. Then it iteratively narrows down the interval defined by a and b by evaluating ϕ at $v = \frac{c-a}{2}$ or $v = \frac{b-c}{2}$ and updating a, b, and c such that the invariant is not violated. That is to say, the scalar v takes the place of the middle point c if $\phi(v) < \phi(c)$. Otherwise, it is assigned to a or b depending on the side of c on which it lies. The respective other interval boundary remains unchanged. A slightly modified algorithm is the *golden section* method which uses the golden ratio to define a new value for v.

A more elaborate algorithm is Brent's line minimization method [Brent, 1973], which combines the stability of the *bisection method* with the speed of *inverse parabolic interpolation*. The latter is known to converge super-linearly under optimal conditions, but it is not guaranteed to converge for non-smooth functions. It is based on the observation that the graph of a function near an extremum can be approximated by a parabola.

The algorithm first computes the minimum of the parabola fitting f at the positions a, b, and c defined above. This minimum, b, and a or c (whatever produces a smaller value of f) define the new interval. The function values at these points are again fitted by a parabola. This procedure is repeated until the interval is narrow enough.

In Brent's method, if *parabolic interpolation* does not progress fast enough, that is, the size of the current step is less than half the size of then step before the last, or steps out of the boundary interval (a, b), the algorithm falls back to the reliable *bisection method* in order to ensure convergence. It usually converges super-linearly, yet linearly in the worst case, that is, if only the *bisection* method is applied.

3.6.3 The L-BFGS-B Algorithm

L-BFGS-B is a limited memory algorithm for solving large nonlinear optimization problems subject to simple bounds on the variables [Zhu et al., 1997]. It uses a limited memory BFGS matrix $\mathbf{B}$ to approximate the Hessian of the cost function. The matrix is used to define a local quadratic model of f:

$$m_i(\mathbf{x}) = f(\mathbf{x}_i) + \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i)^T(\mathbf{x} - \mathbf{x}_i) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_i)^T \mathbf{B}_i (\mathbf{x} - \mathbf{x}_i) \tag{3.40}$$

This model is minimized with respect to the free variables. Eventually, the line search described in [Moré and Thuente, 1994] is performed along the direction defined by the vector from the current position $\mathbf{x}_i$ to the minimizer of m_i.

The algorithm expects the following user inputs:

- The number of input variables
- The number of corrections to the BFGS matrix saved
- An initial estimation of $\mathbf{x}$
- Upper and lower bounds on $\mathbf{x}$
- The cost function's value and gradient on demand
- The *tolerance* used as termination criteria

We chose the L-BFGS-B algorithm to accomplish the optimization task because it is easy to use, does not depend on the computation of second order derivatives, and does not require any knowledge about the structure of the cost function. Moreover, it is possible to set explicit bounds on the subset $\mathbf{X}$ of the parameter space to search for the optimum.

3.6.4 Cost Function Smoothing

It is generally beneficial for optimization strategies if the cost function progresses smoothly towards its minimum. In that case the local gradient provides a good approximation of the direction in which to advance. When smoothing out extrema of smaller extents, the probability of getting trapped or being mislead by local minima decreases. In addition, optimization algorithms tend to converge faster when applied to smoother functions (for example, *steepest descent* in Section 3.6.2). Naturally, the smoothness of the cost function is dependent on the similarity measure itself, but it can also be influenced by methods addressing the DRR generation.

One way to obtain a smoother cost function is the use of *image pyramids* of the X-ray image and the CT volume for rendering the DRRs [Maes et al., 1999, Wein, 2003, Kubias et al., 2007a]. An *image pyramid* is the representation of an image at different resolutions. Usually the number of image elements per dimension is halved from one level of the pyramid to the next. This results in a down-sizing factor of 2^{-d} for d image dimensions, which is achieved by means of sub-sampling. First the image is convoluted with a filter kernel, such as a four-element *box filter* or a *Gaussian filter*.

Then every second image element in each dimension is taken to form the image at the next coarser pyramid level. The filter kernel causes a smoothing of the image. It removes higher frequencies, such as noise, which are otherwise reflected in the DRR and the similarity measure. It makes sense to scale down the images as well, thus rendering lower-resolution DRRs and comparing them to similarly down-sized X-ray images.

The optimization algorithm starts at the coarsest resolution level. After convergence the optimizer is initialized with the parameters found in the precedent pass and operates on the next higher level of resolution. The procedure is repeated until the highest level of resolution is reached. It is evident that a *multi-scale* approach does not only smooth out the cost function, but may also effect a tremendous registration speed-up [Maes et al., 1999]. Since less pixel intensities have to be computed from smaller volumes and similarity is evaluated by means of a smaller amount of pixels, both the potentially expensive DRR generation and similarity measuring are economized.

We use five global pyramid levels, where 0 represents the highest level of detail with CT volumes of around $512 \times 512 \times 300$ voxels and X-ray images with 512×512 or 1024×1024 pixels. The respective level controls the resolution of the CT volume used for rendering, the number of samples used during raycasting, the resolution of the generated DRR, and the resolution of the X-ray image used for similarity measuring. It defines the coarseness of the current optimization pass and is decremented once the optimizer converges. The parameters found at pyramid level 0 represent the final approximation of the model transformation parameters sought-after.

Chapter 4

GPU-Based Computation

4.1 Introduction

Over the past years modern *graphical processing units* (GPUs) have become extremely powerful computation engines. Until a few years ago they were fast but lacking the flexibility to be useful for fields other than computer graphics and rendering, which they were originally designed for. With the rise of programmable shading units GPUs have become a real alternative to the CPU. The computation speed and memory bandwidth of modern GPUs, which are by far superior to those of the CPU, can now be exploited for a diverse field of scientific applications, such as signal and audio processing, simulations, data compression, and solving linear equations. The use of the GPU for applications other than graphics is commonly referred to as *general purpose computation on GPU* (GPGPU).

4.2 GPU Capabilities

As opposed to CPUs, which follow a serial programming model, GPUs are highly efficient *stream processors* [Owens, 2005]. All data, for example, vertices, triangles, or fragments, is considered a *stream* which is processed by *kernels* that implement different functionalities. Since the processing of one stream element within a kernel is never dependent on computations on other stream elements, it can be conducted in parallel. GPUs exploit *parallelism* in three ways:

- **Task parallelism**. Several kernels, that is, kernels performing different tasks, can process different data at the same time. In other words, there is no need for the processing units to wait until the precedent task has been completed. As soon as a kernel has processed some data, the subsequent kernel can start to work in parallel.

- **Data parallelism**. One kernel can process several stream elements at the same time, applying the same calculations to them. For example, the *fragment processor* can process a large number of fragments at a time.

- **Instruction parallelism**. Within the complex evaluation of a single data element, usually a vector, several simple operations may be evaluated in a single-instruction multiple-data (SIMD) fashion.

Another concept which sets GPUs apart from CPUs is task *specialization.* Where CPUs are laid out for flexibility and to execute a maximal variety of different applications, GPUs optimize certain computation units for specific tasks. Implementing computations such as triangle rasterization or texture filtering by means of dedicated hardware entails an enormous efficiency benefit compared to general-purpose hardware. However, graphics hardware offers more and more programmability, which allows for user-defined programs to be executed on certain kernels in the stream, currently the *vertex* and the *fragment processor.* It is these programmable kernels that enable advanced rendering as well as the execution of a broad class of general-purpose applications.

Modern technology allows more and more transistors to be put on one chip. They can be roughly divided into three categories: *control*, *datapath*, and *storage.* During CPU design many transistors are dedicated to complex control functionality such as branch prediction and to multi-level caching techniques decreasing memory latency. In contrast, GPUs maximize the number of transistors in the *datapath.* Since there is less need to control high level program features, they can focus on achieving peak performances for simple and repetitive tasks. Additionally, high-end GPUs have a higher total of transistors than consumer-level CPUs[1].

[1]A total of about 582 million transistors for the Intel® Core™2 Extreme QX6800 [Chiappetta, 2007], compared to 681 million for the NVidia® GeForce® 8800 GTX [ComputerBase, 2007].

The density of transistors has made GPUs extremely capable arithmetic engines that are by far superior to high-end consumer level CPUs[2]. However, technology trends show that memory bandwidth can not keep pace with the rapid growth of computation power. This can be determined by calculating the number of floating point operations that can be carried out per word of off-chip GPU memory bandwidth[3]. Hence, applications with a higher *arithmetic density*, that is, more arithmetic operations per word of memory, are better suited to exploit the capabilities of modern graphics hardware.

Memory latency, on the other hand, is decreasing slower than memory bandwidth is growing. Where CPUs implement sophisticated multi-level caching techniques to minimize latency, GPUs are optimized for throughput[4]. Modern GPUs address this discrepancy by continuing to work while waiting for off-chip data, which is consequently stored in the texture cache, to arrive.

Figure 4.1: The NVidia® GeForce® 8800 GTX graphics board.

[2]Under optimal conditions the GeForce® 8800 GTX performs 518 GFLOPS [ComputerBase, 2007], in contrast to about 37 GFLOPS for the Intel® Core™2 Extreme QX6800 [Schmid and Töpelt, 2007].

[3]The GeForce® FX 5800 (2002) could perform 2, the GeForce® 6800 (2004) 6, and the GeForce® 8800 GTX (2007) 24 floating point operations per word of off-chip GPU memory bandwidth (computed from the respective GFLOPS and the sequential off-chip memory bandwidth).

[4]The GeForce® 8800 GTX comes up with a bandwidth of 86.4 GB/s for sequential memory reads [ComputerBase, 2007], compared to about 13.5 GB/s for the Intel® Core™2 Extreme QX6800 [Freeman, 2007].

4.3 General Purpose GPU Programming

The observation that the programming model of modern GPUs is not only applicable for computer graphics and the introduction of fully programmable *vertex* and *fragment processors* gave rise to a variety of general-purpose applications that experienced a tremendous speed-up on the GPU. High-level programming languages, such as *Cg* (see [NVidia, 2007a]), which allows developers to implement custom functionality with a C-like syntax, have been introduced to facilitate the development of so-called *shader* programs.

Fragment processors are particularly interesting for the use in GPGPU because they leave more control over the output to the programmer. The output of a *fragment shader* is written to the render target as-is, that is, without any further processing, and can be used as input for further rendering passes. The GeForce® 8800 GTX graphics board (see Figure 4.1), which we use for our experiments, has 128 stream processors, each capable of performing one scalar operation at a time. Eight stream processors together form one *multiprocessor*, which can perform fragment, vertex, or geometry processing according to the current computation requirements. Such architectures are referred to as implementing the *unified shader model*. Within one *multiprocessor* a *warp* of 32 threads carries out the same scalar instruction on different data (SIMD), as opposed to older GPU generations, which actually carry out calculations on vectors of four. As many as 24 *warps* can run concurrently on one *multiprocessor* [NVidia, 2007b]. Hence, a *multiprocessor* can *physically* process 8 scalars at a time, yet is capable of *virtually* processing much more data in parallel due to time-slicing, which is used to hide memory latency. However, only when using the NVidia® CUDA™ technology (see [NVidia, 2007b]), one has full control over these features. By the time writing, CUDA™ did not provide support for three-dimensional textures. Therefore, we stick to the *Cg* shading language (see [NVidia, 2007a]) for the time being since it meets our demands on functionality well without requiring major code re-engineering. Note that some of the constraints of GPU-based computation discussed in the following may become obsolete with future GPU technologies, such as CUDA™.

With different processors at the same task level being unable to communicate, we can state the key attributes of GPU computations to be *data parallelism* and *independence* [Harris, 2005]. Ultimately, the success of porting an application to the GPU and optimally exploiting its capabilities is dependent on how well it matches these concepts. The following list provides an overview of implications of performing general-purpose programming on the GPU:

Arithmetic intensity As mentioned earlier, it quantifies the ratio between the number of arithmetic operations and the amount of data loaded from off-chip GPU memory in a program. We have seen that the arithmetic intensity must get higher for every GPU generation in order to reach peak performances. If the arithmetic intensity is high enough, processors may continue to work while waiting for requested data, if not, the arithmetic units are forced to enter an idle state. Hence, recomputing values should be preferred to storing them, unless the cost of computing is higher than for fetching them. Not least is arithmetic power the feature where GPUs are clearly superior to CPUs.

Locality Locality here refers to the way memory is accessed. Similarly to CPUs, sequential memory access is much faster than random access and should thus be generally preferred. The GPU's texture cache does not play the same role as cache on CPUs. It is used primarily to accelerate texture filtering, thus providing locality in two dimensions. Figure 4.2 depicts the results of a memory bandwidth benchmark carried out on the GeForce® 8800 GTX graphics board (see [Houston, 2007]). Additionally, neither cache nor

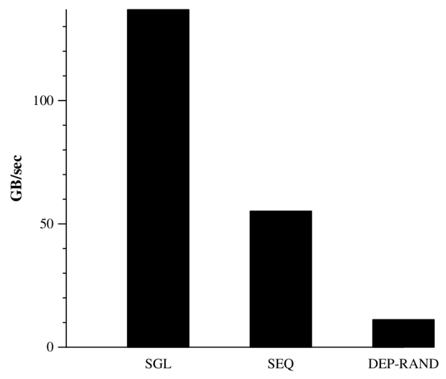


Figure 4.2: Memory performance benchmark of the GeForce® 8800 GTX. The left bar represents the cache bandwidth, the middle bar the sequential off-chip memory access bandwidth, and the right bar the random off-chip memory access bandwidth.

memory on GPUs are writable. The only place where output can be stored is one predefined and unchangeable pixel position in the render target. However, multiple render targets allow the programmer to write to a maximum of four four-channeled targets, which amounts to a total of 16 scalar outputs (18 when including the stencil and the depth buffer). Consequently, programs performing sequential array reads and writes are very likely to profit from GPU-based computation.

Gather vs. scatter As just mentioned, the stream model implemented by GPUs is capable of *gathering* several inputs from arbitrary memory locations, that is, `x = a[i]`, yet is not intended to *scatter* outputs to any location desired, that is, `a[i] = x`. Convolving an image with a Gaussian kernel, for example, is a classical *gather* algorithm. It does not only gather inputs, but also accesses memory in a sequential and 2D-local manner. Such algorithms generally perform very well on the GPU. The CUDA™ technology also provides support for *scattering*, that is, writing to arbitrary locations within a dedicated part of GPU memory, but synchronization techniques are not yet as elaborate as for CPUs. However, the techniques we chose for our registration approach do not rely on *scattering* since rendering and similarity measuring are performed in a per-pixel fashion.

Value precision and range NVidia® GPUs currently support the IEEE single-precision floating point format, where in the end of 2007 AMD® presented the first GPU supporting the double-precision floating-point format [AMD, 2007]. Hence, precision might not be an issue anymore in the future. However, until double-precision processors are largely established, GPU programmers must be aware of the limitations that lower-precision floating-point values may entail. Scientific applications may depend on high-precision data types. It was shown that double-precision floating point types can be emulated on the GPU [Göddeke et al., 2005]. Additionally, GPUs have no built-in integer data types. Shader languages might provide them for reasons of convenience, yet their range is smaller than true integer data types with the same number of bytes. For example, the IEEE four-byte floating point type can represent the contiguous zero-centered range of $\pm 16,777,216$ [Harris, 2005] where a signed four-byte integer type can hold values from $-2,147,483,648$ to $+2,147,483,647$.

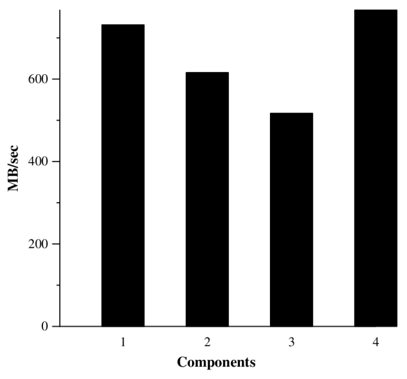
Control flow As mentioned earlier, GPUs are not laid out for complex control requirements. Therefore, conditional statements are to be used with precaution since their use incurs a certain overhead, especially when used in inner loops. For example, for the GeForce® 6 Series an `if/endif`-block causes a loss of 4 clock cycles [Kilgariff and Fernando, 2005]. Additionally, older GPU generations only support *static branching*, which executes both conditional branches anyway. This is because hundreds of pixels are processed in an uniform way and conditional statements are allowed for by writing registers only in the branch that the predicate defines to take. However, newer GPUs such as the GeForce® 8 series support *dynamic branching* as an alternative to *predicated instructions*. Because of smaller blocks of fragments being processed concurrently within one *multiprocessor*, it may be beneficial to evaluate only one branch per fragment. Since the instruction unit still caters one instruction for all processors at a time, some threads have to be stopped until the end of the branch they are not supposed to take. The use of *dynamic branching* pays off only if the cost for synchronization is lower than for the evaluation of the skipped branches. Our GPU computation gets by with one `for`-loop, which is required to sample the volume along the viewing rays.

Download and readback One of the biggest bottlenecks in GPU-based computing is the data communication between GPU and main memory. On-board memory bandwidths are by far superior to off-board bandwidths. Figure 4.3(a) and Figure 4.3(b) show the results of a respective download and readback bandwidth benchmark carried out on the GeForce® 8800 GTX graphics board (see [Houston, 2007]). When developing GPU programs, it is crucial to bear in mind the cost of transferring large amounts of data to and from GPU memory. Yet, there are many cases where it is not even necessary to transfer whole textures. For example, *texture reduction*, which we use to sum up image intensities (see Section 4.4.2), reduces the amount of data to be read back from an arbitrary amount of texels to one single element. Also, in many cases it may be much faster to (re)compute values on the GPU than to download them to GPU memory. Our application requires the CT and X-ray data to be downloaded and only about a dozen of values to be read back.

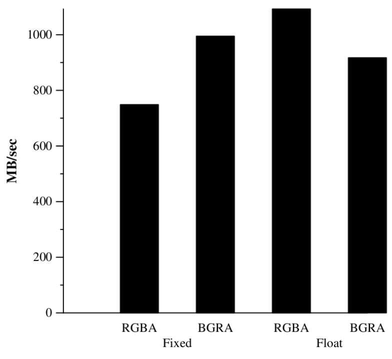
4.4 GPU-Based Registration

4.4.1 GPU-Based DRR rendering

DRR raycasting as discussed in Section 3.3.3 represents a standard *image-order* volume rendering approach and can thus be ported naturally to the GPU. *Image-order* rendering approaches inherently match the stream programming model implemented by GPUs very well since they are performed in a pixel-wise fashion. Within a fragment shader, an intensity value is computed for every screen pixel independently, where data is gathered from a



(a)



(b)

Figure 4.3: Off-board communication benchmark of the GeForce® 8800 GTX. The left chart subsumes the results for data transfers from main to GPU memory (*download*), where the right chart shows the bandwidths for transferring data from GPU memory back to main memory (*readback*).

three-dimensional texture representing the CT dataset. Algorithm 1 depicts the algorithm as implemented by the fragment shader, where $\tilde{\mathbf{f}}_{\text{win}}$ and $\tilde{\mathbf{b}}_{\text{win}}$ denote the front and back intersection of the ray with the volume in window coordinates. The tilde marks homogeneous vectors. The first and the last sampling position are given by $\mathbf{s}$ and $\mathbf{e}$. The step vector, whose length is δ, is referred to as $\mathbf{x}$. The subscripts ”cam”, ”world”, and ”obj” assign the vectors respectively to the camera, the world, or the object coordinate system, that is, the coordinate system of the volume in metric units. The function vol returns the interpolated radiodensity at sampling positions $\mathbf{p}_{\text{obj}}$ within the CT volume. The transformation matrix $\mathbf{H}$, which maps window coordinates to camera coordinates, is defined as follows:

$$\mathbf{H} = (\mathbf{WP})^{-1} \tag{4.1}$$

where $\mathbf{W}$ transforms normalized device coordinates into window coordinates.

In fact, the volume is sampled at fixed positions in the camera coordinate system. Choosing the sampling positions to be independent of the model transformation parameters considerably facilitates the computation of the algorithm’s gradient with respect to these parameters. For, if we define the sampling positions in an object-based manner, the computations performed by the rasterizer in the rendering pipeline, which are beyond the influence of the fragment shader, have to be considered during the differentiation. Since

Algorithm 1 The original DRR algorithm

```
Require: $\mathbf{C}, \mathbf{R}, \mathbf{T}, \mathbf{H} \in M_4(\mathbb{R})\,;\; n_{\text{smpl}} \in \mathbb{N}$
 1: for all $(u,v) \in \Omega$ do
 2:     $\tilde{\mathbf{f}}_{\text{cam}} = \mathbf{H}\tilde{\mathbf{f}}_{\text{win}}$
 3:     $\tilde{\mathbf{b}}_{\text{cam}} = \mathbf{H}\tilde{\mathbf{b}}_{\text{win}}$
 4:     $\mathbf{f}_{\text{cam}} = \frac{\tilde{\mathbf{f}}_{\text{cam}}.xyz}{\tilde{\mathbf{f}}_{\text{cam}}.w}$
 5:     $\mathbf{b}_{\text{cam}} = \frac{\tilde{\mathbf{b}}_{\text{cam}}.xyz}{\tilde{\mathbf{b}}_{\text{cam}}.w}$
 6:     $\mathbf{x}_{\text{cam}} = \delta \text{normalize}(\mathbf{b}_{\text{cam}} - \mathbf{f}_{\text{cam}})$
 7:     $\mathbf{s}_{\text{cam}} = \left\lceil \frac{\|\mathbf{f}_{\text{cam}}\|}{\delta} \right\rceil \mathbf{x}_{\text{cam}}$
 8:     $\mathbf{e}_{\text{cam}} = \left\lfloor \frac{\|\mathbf{b}_{\text{cam}}\|}{\delta} \right\rfloor \mathbf{x}_{\text{cam}}$
 9:     $n_{\text{smpl}} = 1 + \frac{\|\mathbf{e}_{\text{cam}} - \mathbf{s}_{\text{cam}}\|}{\delta}$
10:     $\mathbf{s}_{\text{world}} = \mathbf{V}^{-1}\mathbf{s}_{\text{cam}}$
11:     $\mathbf{x}_{\text{world}} = \mathbf{V}^{-1}\mathbf{x}_{\text{cam}}$
12:     $\mathbf{s}_{\text{obj}} = \mathbf{C}\mathbf{R}^{-1}\mathbf{T}^{-1}\mathbf{C}^{-1}\mathbf{s}_{\text{world}}$
13:     $\mathbf{x}_{\text{obj}} = \mathbf{R}^{-1}\mathbf{x}_{\text{world}}$
14:     $\mathbf{p}_{\text{obj}} = \mathbf{s}_{\text{obj}}$
15:     $I(u,v) = 0$
16:     for $i = 1$ to $n_{\text{smpl}}$ do
17:         $I(u,v) = I(u,v) + \text{vol}(\mathbf{p}_{\text{obj}})$
18:         $\mathbf{p}_{\text{obj}} = \mathbf{p}_{\text{obj}} + \mathbf{x}_{\text{obj}}$
19:     end for
20: end for
```

the discrete intersection points of the rays with the volume's surface it computes are dependent on the model transformation parameters, the rasterizer can not be considered constant with respect to the model transformation. Choosing fixed sampling positions in camera space, on the other hand, allows one to differentiate the DRR rendering statements in Algorithm 1 exclusively. At the same time, this is the reason why we do not use the elegant GPU-based raycasting method proposed in [Krüger and Westermann, 2003]. It encodes the texture coordinates of the volume's front and back faces by means of colors to let the rasterizer compute the pixel-wise intersection points of the ray with the volume's surface. Instead, we use a slightly modified version of the algorithm.

Since the volume takes up only a small part of the window space, the fragment shader implementing Algorithm 1 does not sample it completely from front to back. Instead, the first and the last sample are derived from the intersection points with the volume's front and back faces, namely $\tilde{\mathbf{f}}_{\mathrm{win}}$ and $\tilde{\mathbf{b}}_{\mathrm{win}}$. In *Cg* programs the window coordinates of single fragments is semantically bound to the `WPOS` parameter. In our case it reflects the pose of the volume and the camera (incorporated by means of the OpenGL `MODELVIEW_MATRIX`), as well as the perspective projection properties of the camera (considered by `PROJECTION_MATRIX`). The z-coordinates of the front faces are obtained during a preliminary rendering pass, where the depth component is rendered to texture using back-face culling. This texture is passed as an argument to the main rendering pass, which is triggered by drawing only the back faces of the volume. Within the DRR rendering shader, the intersections are first transformed into homogeneous camera space and converted back to Cartesian coordinates. The first and the last sampling position $\mathbf{s}_{\mathrm{cam}}$ and $\mathbf{e}_{\mathrm{cam}}$ in the camera coordinate system are determined as follows:

$$\mathbf{s}_{\mathrm{cam}} = \left\lceil \frac{||\mathbf{f}_{\mathrm{cam}}||}{\delta} \right\rceil \mathbf{x}_{\mathrm{cam}} \tag{4.2}$$

$$\mathbf{e}_{\mathrm{cam}} = \left\lfloor \frac{||\mathbf{b}_{\mathrm{cam}}||}{\delta} \right\rfloor \mathbf{x}_{\mathrm{cam}} \tag{4.3}$$

which determines the first and the last constant sampling position on the ray that is still located within the volume. From these values the minimal number of samples n_{smpl} can be derived. Next the starting position and the step vector are transformed to world coordinates by means of the inverse viewing matrix $\mathbf{V}^{-1}$. To recover their counterparts in object space, the remaining chain of transformations is applied in reversed order:

$$\mathbf{s}_{\text{obj}} = (\mathbf{TCRC}^{-1})^{-1}\mathbf{s}_{\text{world}} \tag{4.4}$$

$$= \mathbf{CR}^{-1}\mathbf{C}^{-1}\mathbf{T}^{-1}\mathbf{s}_{\text{world}} \tag{4.5}$$

where for the step vector only the rotation is relevant:

$$\mathbf{x}_{\text{obj}} = \mathbf{R}^{-1}\mathbf{x}_{\text{world}} \tag{4.6}$$

Within the inner loop the sampling positions $\mathbf{p}_{\text{obj}}$ are fed to vol, which performs tri-linear interpolation of the eight nearest neighboring voxels in object space. When the loop has finished, $I(u, v)$ contains the accumulated radiodensities for the image pixel (u, v). Eventually, the weighted DRR intensities $w(u, v)I(u, v)$ and the weighted squares $w(u, v)I(u, v)^2$ (see Section 4.4.2) are written to the render target, that is, to texture. The weight mask w is available in the form a two-dimensional input texture which is passed to the DRR rendering shader (and subsequent shaders) as an argument.

To speed up rendering and to obtain smoother DRRs in earlier stages of the optimization procedure, the CT data accessed by vol is made available at different levels of detail. Additionally, the sampling distance δ is increased and the size of the DRRs to generate is reduced when a lower level is requested (see Section 3.6.4).

4.4.2 GPU-Based Similarity Measuring

From the definition of normalized cross correlation (equation (3.25) in Section 3.5.3) it is clear that the measure can not be evaluated along with DRR generation, that is, within one rendering pass. To compute the pixel-wise correlation, the mean DRR intensity has to be known. Note that the mean and standard deviation of the X-ray image to be aligned can be computed beforehand. To obtain the mean over the DRR without transferring it back to CPU main memory a *texture reduction* technique is applied. For this purpose, two textures are alternately read from and written to, where the viewport is gradually scaled down by a factor of two in each dimension. The information of four pixels each is gathered from the read-texture to compute one pixel value of the downscaled image written to the write-texture. Then their roles are swapped and the part of the texture just written to is processed. This is done until the viewport has reduced to one single pixel constituting the result. Reading four texels, filtering them, and writing the result to the render target is implemented by means of a simple fragment shader program.

In order to compute the mean DRR intensity, the four pixel values are summed up to form the new value to write. Eventually, one pixel contains the sum of all DRR intensities. This value is read back to main memory in order to compute the mean. Figure 4.4 shows an example of reducing a texture to its sum. In practise, not only one scalar, but all four texture channels are summed up simultaneously.

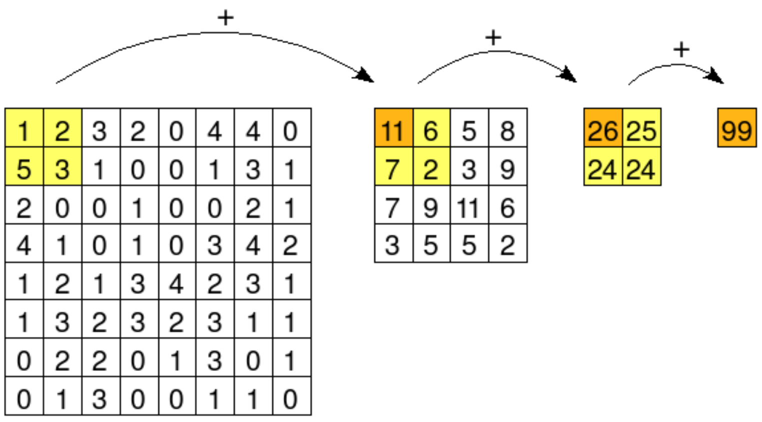


Figure 4.4: An example of *texture reduction* used to sum up the intensities of an image. The viewport written to is scaled down to a forth of its current size at every step.

In order to save an additional rendering pass, the standard deviation is calculated as follows:

$$\sigma_{wI} = \sqrt{\frac{1}{\Sigma_w} \sum_{(u,v)\in\Omega} w(u,v)I(u,v)^2 - \mu_{wI}^2} \tag{4.7}$$

where the weighted intensity squares $w(u,v)I(u,v)^2$ are computed during the DRR rendering pass and subsequently summed up along with the weighted intensities $w(u,v)I(u,v)$. Since both values represent scalars, they fit into one texture. Indeed, this formulation of standard deviation is susceptible to round-off errors, yet the computation is carried out on the CPU, where double-precision floating point types are available.

The DRR texture and the X-ray texture, as well as their mean intensities and the weight mask texture are then fed to the next rendering pass, where the pixel-wise correlations

$$\mathrm{corr}(u,v) = w(u,v)(I(u,v) - \mu_{wI})(J(u,v) - \mu_{J,w}) \tag{4.8}$$

are computed and written to texture. Their sum is obtained by texture reduction, read back, and used to compute the final NCC measure:

$$\mathrm{NCC}(I,J,w) = \frac{1}{\sigma_{wI}\sigma_{wJ}} \frac{1}{\Sigma_w} \sum_{(u,v)\in\Omega} \mathrm{corr}(u,v) \tag{4.9}$$

The whole similarity measuring algorithm is depicted in Algorithm 2. Note that the use of **for**-loops over Ω always hints at the use of GPU-based computation by means of fragment shaders.

Algorithm 2 The similarity measuring algorithm.

```
Require: $I, J, w \in M_n(\mathbb{R})\,;\ \mu_{wJ}, \sigma_{wJ}, \Sigma_w \in \mathbb{R}$
 1: $\Sigma_{wI} = 0, \Sigma_{wI^2} = 0$
 2: for all $(u,v) \in \Omega$ do
 3:     $\Sigma_{wI} = \Sigma_{wI} + w(u,v)I(u,v)$
 4:     $\Sigma_{wI^2} = \Sigma_{wI^2} + w(u,v)I(u,v)^2$
 5: end for
 6: $\mu_{wI} = \frac{\Sigma_{wI}}{\Sigma_w}$
 7: $\sigma_{wI} = \sqrt{\frac{\Sigma_{wI^2}}{\Sigma_w} - \mu_{wI}^2}$
 8: $\Sigma_{\mathrm{corr}} = 0$
 9: for all $(u,v) \in \Omega$ do
10:     $\Sigma_{\mathrm{corr}} = \Sigma_{\mathrm{corr}} + w(u,v)(I(u,v) - \mu_{wI})(J(u,v) - \mu_{wJ})$
11: end for
12: $\mathrm{NCC} = \frac{\Sigma_{\mathrm{corr}}}{\Sigma_w \sigma_{wI} \sigma_{wJ}}$
```

Chapter 5

Algorithmic Differentiation

5.1 Introduction

Algorithmic differentiation (AD) is a method that allows for an automatic computation of exact derivatives for vector-valued functions of the form

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) \qquad \mathbf{f} : \mathbb{R}^N \to \mathbb{R}^M \tag{5.1}$$

consisting of an arbitrary number of elementary sub-statements. The alternative and more commonly used term *automatic differentiation* emphasizes the automatic generation and evaluation of derivative code by means of computer programs. AD represents an alternative to the traditional calculation of derivatives using *numerical approximation* and *symbolic differentiation*. It overcomes the drawbacks of both methods.

5.1.1 Symbolic Differentiation

Symbolic differentiation takes as input an algebraic expression and generates a derivative expression with respect to one of its input parameters. Thus, the cost of computing a function's gradient depends on the size of the input.

Yet, it is inherently hard to map whole computer programs to mathematical expressions. This becomes apparent when taking into account control structures like conditional statements or loops. Furthermore, the size of the resulting expressions tends to grow excessively large for an increasing number of independent input parameters and higher order derivatives. This results in an inefficient computation and evaluation of derivative code, high memory requirements, and hardly maintainable code [Griewank, 1989], which leads to the conclusion that symbolic differentiation is not applicable for our accelerated registration algorithm.

5.1.2 Numerical Approximation

Numerical approximation is based on Newton's difference quotient. It approximates the derivative of a function $\mathbf{f}$ with respect to x_n at a point $\mathbf{x}$ by evaluating the function at two nearby points $\mathbf{x} + h\mathbf{v}_n$ and $\mathbf{x} - h\mathbf{v}_n$ ($h \in \mathbb{R}$):

$$\frac{\partial \mathbf{f}}{\partial x_n}(\mathbf{x}) \approx \frac{\mathbf{f}(\mathbf{x} + h\mathbf{v}_n) - \mathbf{f}(\mathbf{x} - h\mathbf{v}_n)}{2h} \tag{5.2}$$

where $\mathbf{v}_n \in \mathbb{R}^n$ is the unit vector codirectional with x_n.

The accuracy of the approximation depends on the size of h. If chosen too large, the approximation of the gradient will get worse, if chosen too small, the subtraction may cause a loss of significance when using finite-precision floating-point arithmetic. The dilemma is to find a tradeoff between *truncation error* and *cancellation error* [Bischof and Bücker, 2000]. However, numerical approximation of the derivative allows the function $\mathbf{f}$ to be treated as a black-box, which does not require any processing or potentially error-prone modifications of the original function's code.

The cost of numerically approximating the whole gradient vector is linearly dependent on the number of variable input parameters. The ratio between the cost of calculating the gradient and evaluating the function is $2N$ (two function evaluations per input parameter) using central differences, and $N + 1$ using forward or backward differences, where the function evaluation at point $\mathbf{x}$ can be reused for all elements of the gradient vector. Approximating the registration algorithm's gradient with respect to the six model transformation parameters thus requires either twelve or seven evaluations of $\mathbf{f}$. Numerical approximation serves as the reference for comparing our approach in terms of performance, accuracy, and convergence.

5.2 Algorithmic Differentiation

Algorithmic differentiation is a mathematical concept that automatically generates the derivative of a vector-valued function $\mathbf{f}$ represented by a computer program. To do so, it takes the original program's statements, interlaces them with derivative statements, and produces a new program which evaluates the function and its derivative at the position defined by the input parameter vector $\mathbf{x}$. The accuracy of the resulting gradient is only limited by machine precision.

Essentially, algorithmic differentiation corresponds to a gradual application of the chain rule from differential calculus to a program. Let s_i ($s_i : \mathbb{R} \to \mathbb{R}$, $i = 1, \ldots, I$) be a list of consecutive statements of the simple program p ($p : \mathbb{R} \to \mathbb{R}$), each referencing the output of its predecessor:

$$v_i = s_i\left(v_{i-1}\right) \tag{5.3}$$

$$p\left(v_0\right) = \left(s_I \circ s_{I-1} \circ \ldots \circ s_1\right)\left(v_0\right) \tag{5.4}$$

where v_0 is an input parameter, then

$$p'\left(v_0\right) = s'_I\left(s_{I-1}\left(s_{I-2}\left(\ldots\right)\right)\right) s'_{I-1}\left(s_{I-2}\left(\ldots\right)\right) \ldots s'_1\left(v_0\right) \tag{5.5}$$

Note that we need the program variable's intermediate value v_{i-1} to compute the derivative

$$\frac{\partial s_i}{\partial v_{i-1}} \tag{5.6}$$

if the chain rule applies for the elementary statement s_i, that is, a statement implementing a nonlinear function, for example, $s_i(v_{i-1}) = v_{i-1}^2$.

Likewise, a subsequent program evaluation induces the program to evaluate the chain rule. The two basic modes of algorithmic differentiation, namely *reverse* or *adjoint mode* and *forward* or *tangent-linear mode*, differ in the direction the chain rule is evaluated. The implications of the way derivative information is propagated through the program are fundamental. In the following we will generalize (5.5) for N inputs and M outputs and establish the prerequisites for a well-founded description of the basic modes of algorithmic differentiation.

Following the notation of [Pock et al., 2006], we define $\mathbf{v} = [v_q]_{q=1}^{Q}$ to be the vector of all program variables, including input parameters, intermediate variables, and output variables. We split up the original program into L sequential blocks, each modifying every variable at most once and referencing only variables modified in preceding blocks. The global operator $\mathbf{S}$ ($\mathbf{S} : \mathbb{R}^Q \to \mathbb{R}^Q$) maps the initial variable vector $\mathbf{v}^{(0)}$ to the final variable vector $\mathbf{v}^{(L)}$. The local operators $\mathbf{S}^{(l)}$ ($\mathbf{S}^{(l)} : \mathbb{R}^Q \to \mathbb{R}^Q$, $l = 1, \ldots, L$) map the variable vector at the beginning of block l to the vector at the end of that block:

$$\mathbf{v}^{(L)} = \mathbf{S}(\mathbf{v}^{(0)}) \tag{5.7}$$

$$\mathbf{v}^{(l)} = \mathbf{S}^{(l)}(\mathbf{v}^{(l-1)}) \tag{5.8}$$

with $l = 1, \ldots, L$.

We can now define the input parameter vector $\mathbf{x} = [x_n]_{n=1}^N$ and the function vector $\mathbf{y} = [y_m]_{m=1}^M$ as follows:

$$\mathbf{x} = \left[v_p^{(0)}\right]_{p=1}^{N} \tag{5.9}$$

$$\mathbf{y} = \left[v_r^{(L)}\right]_{r=Q-M+1}^{Q} \tag{5.10}$$

In addition, we define $\mathbf{S}_q$ ($q = 1, \ldots, Q$) to be the set of statements out of $\mathbf{S}$ that accounts for $v_q^{(L)}$, the final value of variable v_q, and require it to be differentiable in $\mathbb{R}^Q$. Consequently, $\mathbf{S}_q^{(l)}$ constitutes the definition of variable v_q in block l. Note that we can add an imaginary self-assignment for every v_q that is not defined in l.

5.2.1 Forward Mode

In the *forward* or *tangent-linear mode* of automatic differentiation the derivative information is propagated along the flow of the original program. When regarding equation (5.5), this corresponds to an evaluation of the chain rule from right to left. During the automatic differentiation process the input program is augmented by derivative statements resulting from the differentiation of the program's statements with respect to the program's variables. A subsequent evaluation of the resulting program yields the function values as well as the derivative along a predefined direction.

Let us now introduce the Jacobian matrices related to the global operator $\mathbf{S}$ and the local operators $\mathbf{S}^{(l)}$:

$$\mathbf{J} = \left[\frac{\partial \mathbf{S}_i}{\partial v_j^{(0)}}\right]_{i,j=1}^{Q} = \left[\frac{\partial v_i^{(L)}}{\partial v_j^{(0)}}\right]_{i,j=1}^{Q} \tag{5.11}$$

$$\mathbf{J}^{(l)} = \left[\frac{\partial \mathbf{S}_i^{(l)}}{\partial v_j^{(l-1)}}\right]_{i,j=1}^{Q} = \left[\frac{\partial v_i^{(l)}}{\partial v_j^{(l-1)}}\right]_{i,j=1}^{Q} \tag{5.12}$$

which are populated by the derivatives of all program variables with respect to these variables respectively for the whole program and a certain block l. For a block l of self-assignments (including imaginary ones) the corresponding Jacobian $\mathbf{J}^{(l)}$ equals the identity matrix. It should be noted that in practise we don't need to compute the whole Jacobian matrix $\mathbf{J}^{(l)}$ since we are only interested in the program variables that are actually defined in block l.

Similarly to (5.7) and (5.8), the Jacobian matrices operate on tangent-linear variable vectors $\mathbf{w}_{\mathrm{tl}}^{(l)}$ ($l = 0, \ldots, L$):

$$\mathbf{w}_{\mathrm{tl}}^{(L)} = \mathbf{J}(\mathbf{v}^{(0)})\mathbf{w}_{\mathrm{tl}}^{(0)} \tag{5.13}$$

$$\mathbf{w}_{\mathrm{tl}}^{(l)} = \mathbf{J}^{(l)}(\mathbf{v}^{(l-1)})\mathbf{w}_{\mathrm{tl}}^{(l-1)} \tag{5.14}$$

According to [Pock et al., 2006], we can record the following relationship:

$$\mathbf{J} = \bigodot_{l=L}^{1} \mathbf{J}^{(l)} \tag{5.15}$$

In other words, recursively applying the block-wise derivatives $\mathbf{J}^{(l)}$, which is tantamount to applying the chain rule along the program flow, yields the global derivative $\mathbf{J}$. Now if we set the tangent-linear variable vector $\mathbf{w}_{\mathrm{tl}}^{(0)}$ to be a unit vector $\hat{\mathbf{w}}_{\mathrm{tl}}^{(0)}$ in $\mathbb{R}^Q$, we induce the algorithm to compute the directional derivative

$$\frac{\partial \mathbf{v}^{(L)}}{\partial w_{\mathrm{tl}}^{(0)}} = \mathbf{J}(\mathbf{v}^{(0)})\hat{\mathbf{w}}_{\mathrm{tl}}^{(0)} \tag{5.16}$$

along the direction defined by $\hat{\mathbf{w}}_{\mathrm{tl}}^{(0)}$.

The initial tangent-linear vector $\mathbf{w}_{\mathrm{tl}}{}^{(0)}$ is also referred to as *weight* or *seed vector* because it weights the influence of the input parameters on the directional derivative. Our primary interest lies in the directional derivative of $\mathbf{f}$:

$$\frac{\partial \mathbf{f}}{\partial s} = \left[\frac{\partial v_r^{(L)}}{\partial w_{\mathrm{tl}}^{(0)}}\right]_{r=Q-m+1}^{Q} \tag{5.17}$$

where $\mathbf{s}$ is a vector in the input parameter space of $\mathbf{f}$.

If, for example, the weight vector is chosen to be the unit vector co-directional with x_n, the differentiated program is induced to yield the n-th column of the Jacobian $\mathbf{J}$. These values represent the derivatives of all program variables (including the function values) with respect to the input parameter x_n. Consequently, N evaluations of the differentiated program are required to compute the whole gradient of the program $\mathbf{f}$. This means that there exists a linear relationship between the number of input parameters and the cost of computing the gradient, as it is the case for symbolic differentiation (Section 5.1.1) and numerical approximation of the gradient (Section 5.1.2). Yet, the forward mode of algorithmic differentiation is superior to the latter in terms of accuracy and to the former in terms of efficiency for most N [Griewank, 1989].

5.2.2 Reverse Mode

In the *reverse* or *adjoint mode* of automatic differentiation the derivative information is propagated against the flow of the original program. When regarding equation (5.5), this corresponds to an evaluation of the chain rule from left to right. Again the original program is augmented by derivative statements, but in contrast to the forward mode, the program flow is reversed. This fact brings up new challenges, especially when dealing with constraints posed by a GPU-based programming model. However, the reverse mode comes up with a reward that leaves no doubt about the fundamental usefulness of the algorithm: The program's derivatives with respect to all input parameters can be computed with only one evaluation of the differentiated program, where the derivative program's performance is not linearly dependent on the dimensionality of the gradient vector. Instead, the ratio between the cost of evaluating the gradient and evaluating the original function has a constant upper bound of five [Griewank, 1989].

Trivially speaking, we are interested in the last M rows of the Jacobian matrix $\mathbf{J}$, which contain the derivatives of the function values $v_r^{(L)}$ $(r = Q - M + 1, \ldots, Q)$ with respect to all program variables $\mathbf{v}^{(0)}$. Equation (5.16) revealed that the forward mode can only produce a weighted sum of the Jacobian's columns. Hence, we transpose the Jacobian and observe the implications on the algorithm. The transpose of the global Jacobian can be obtained as the result of multiplying the transposed block-wise Jacobian matrices in reversed order:

$$\mathbf{J}^T = \bigodot_{l=1}^{L} \mathbf{J}^{(l)^T} \tag{5.18}$$

Hence, the transposition entails a reversal of the original block-order. As a consequence, the derivative information flows from the end of the program up to the beginning:

$$\mathbf{w}_{\text{ad}}^{(0)} = \mathbf{J}^T(\mathbf{v}^{(0)})\mathbf{w}_{\text{ad}}^{(L)} \tag{5.19}$$

$$\mathbf{w}_{\text{ad}}{}^{(l-1)} = \mathbf{J}^{(l)^T}(\mathbf{v}^{(l-1)})\mathbf{w}_{\text{ad}}^{(l)} \tag{5.20}$$

where $\mathbf{w}_{\text{ad}}^{(l)}$ $(l = 0, \ldots, L)$ denotes the adjoint variable vector.

Since the derivatives of one final program variable with respect to the initial program variables are now arranged in the columns of $\mathbf{J}$, we can retrieve them by initializing the adjoint variable vector $\mathbf{w}_{\text{ad}}^{(L)}$ appropriately. Regarding (5.12) and bearing in mind the nature of matrix-vector multiplications, we can state the following:

$$\frac{\partial v_q^{(L)}}{\partial \mathbf{v}^{(0)}} = \mathbf{J}^T(\mathbf{v}^{(0)})\hat{\mathbf{w}}_{\text{ad}}^{(L)} \tag{5.21}$$

where $\hat{\mathbf{w}}_{\text{ad}}^{(L)}$ is defined as the unit vector along the v_q-axis. It can be considered as a seed vector controlling the program variable for which to compute the gradient, namely the variable $v_q^{(L)}$. As a matter of fact, we are not interested in all columns of $\mathbf{J}^T$, but only the M rightmost ones, which contain the gradients of the function values y_m $(m = 1, \ldots, M)$. In addition, we can limit our consideration to the N upper rows, which are populated by the derivatives with respect to the input parameters x_n $(n = 1, \ldots, N)$:

$$\frac{\partial y_m}{\partial \mathbf{x}} = \left[\frac{\partial v_{Q-M+m}^{(L)}}{\partial v_p^{(0)}}\right]_{p=1}^{N} \tag{5.22}$$

Until now we did not discuss the implications of propagating derivative information against the original program flow. In Section 5.2 we observed that we need the program variables' intermediate values $\mathbf{v}^{(l)}$ in order to compute the derivatives of statements implementing nonlinear functions. While in the forward mode the tangent-linear variables evolve along with the original program variables, we are facing the contrary situation in the reverse mode. That is to say, for the derivative of a nonlinear operator $\mathbf{S}^{(l)}$ we need the intermediate variable vector $\mathbf{v}^{(l-1)}$, which attributes to the evaluation of the blocks 1 to $l-1$, as can be seen in equation (5.20). By virtue of program flow reversal, these blocks have not been evaluated yet.

There are two complementary approaches to solving this problem. The first option is to recompute the intermediate values $v_q^{(l)}$ whenever they are needed by evaluating the blocks 1 to l, which is inherently computationally intense. The second approach computes all intermediate values during a so-called *forward sweep* and stores them before they are over-written or go out of scope. Naturally, this may require a vast amount of memory, which can pose a big drawback for GPU-based computations, where off-chip communication is costly and buffering is only possible to a very limited extent (see Section 4.3). The main challenge consists in finding an optimal tradeoff between storage and recomputation for a problem at hand and a given set of constraints. A lot of research has been done on how to recompute variables efficiently [Giering and Kaminski, 2002] and keep memory requirements low [Grimm et al., 1996], yet the optimal solution is often dependent on the respective field of application.

5.2.3 Algorithmic Differentiation Implementations and Tools

Implementations of algorithmic differentiation take the source code of a computer program as input and produce a new program which evaluates the original program as well as some kind of derivative at a certain point in the input parameter space. There are two classical types of implementations:

- **Operator overloading**

  This approach exploits the *overloading* concept of modern program languages, which enables the custom redefinition of functions and certain built-in operators like $+$, $*$, and $=$, such as *function* and *operator overloading* in C++. This capability is used to extend the implementation of mathematic functions and operators by their derivatives or by statements which keep a runtime log virtually without changing the code of the original program. The evaluation of the derivatives and the logging, which may be used to buffer intermediate values, subsequently takes place "in the background".

  The advantages of operator overloading are the terseness and flexibility of the code and the availability of a runtime log of the program. Yet, it lacks transparency, comes up with a certain runtime overhead and is not supported by all programming languages [Bischof and Bücker, 2000]. However, the major drawback is the lack of context, which is inherent in operator overloading. It prevents many source code optimizations as well as the exploitation of the chain rule's associativity [Bischof et al., 1997].

  Popular tools implementing this technique comprise: ADOL-C (C, C++) [Griewank et al., 1996], ADOL-F (Fortran), ADC03 (C, C++), and ADF03 (Fortran).

- **Source transformation**

  This approach transforms the input source code such that it explicitly computes the derivative information, thus augmenting the program by derivative statements. As opposed to operator overloading, it is not bound to any predefined code structure. Therefore, and since information on context is available, source transformation can make use of a wide range of optimization techniques including code restructuring and compile-time optimizations. On the other hand, it is considerably harder to implement tools based on source transformation than on operator overloading.

Popular tools implementing this technique comprise: ADIC (C, C++) [Bischof et al., 1997], ADIFOR (Fortran), TAMC (Fortran), OpernAD (Fortran), and Tapenade (Fortran).

5.3 Differentiation of the Registration Algorithm

In order to obtain the derivative of the cost function with respect to all six model transformation parameters, the reverse mode of algorithmic differentiation has to be applied to the whole DRR generation (Algorithm 1) and similarity measuring code (Algorithm 2). Since *Cg* does not support high level language concepts, an *operator overloading* approach is out of question. Instead, an implementation of *source transformation*, which additionally leaves room for optimization by means of code-restructuring, is used to obtain an initial version of the differentiated algorithm. A custom-built parser traverses the code's parse tree produced by the GNU C compiler[1] and generates the derivative statements for every node. To handle symbols and expressions the GiNaC library [Bauer et al., 2002] is used.

By virtue of reverse mode AD, the flow of the whole program is reversed. Hence, we first consider the similarity measuring algorithm. Algorithm 3 depicts the derivatives of the statements in Algorithm 2 as produced by the reverse mode algorithm.

The ad_-prefixes mark the adjoint counterparts of the original program variables. Note that statements which do not affect the result, that is, adjoint variables of constants, are not included. Furthermore, we did not consider the original program variables whose intermediate values are requested before they have been evaluated. For example, the value of σ_{wI} is already needed in the first line, where in the original algorithm it is computed only in the seventh line, which in turn accounts for the derivative statements in line 7 and 8. What is more, the values of Σ_{corr}, σ_{wI}, Σ_{wI^2}, Σ_{wI}, and μ_{wI} depend directly on the DRR intensities $I(u, v)$, which are the most expensive to obtain. It is out of question to recompute them for every use, which leads us to considering a pre-computation strategy.

The result of applying the reverse mode AD to Algorithm 1 is depicted in Algorithm 4. Note that this time we partly included the statements which evaluate the original program variables' values as required by the adjoint

[1]`http://gcc.gnu.org/`

Algorithm 3 The reverse mode of AD applied to the similarity measuring algorithm.

```
Require: $\text{ad\_}* = 0$; $\text{ad\_NCC} = 1$
 1: $\text{ad\_}\Sigma_{\text{corr}} = \text{ad\_}\Sigma_{\text{corr}} + \frac{1}{\Sigma_w \sigma_{wI} \sigma_{wJ}} \text{ad\_NCC}$
 2: $\text{ad\_}\sigma_{wI} = \text{ad\_}\sigma_{wI} + \frac{-\Sigma_{\text{corr}}}{\Sigma_w \sigma_{wI}^2 \sigma_{wJ}} \text{ad\_NCC}$
 3: for all $(u, v) \in \Omega$ do
 4:     $\text{ad\_}I(u,v) = \text{ad\_}I(u,v) + w(u,v)(J(u,v) - \mu_{wJ})\text{ad\_}\Sigma_{\text{corr}}$
 5:     $\text{ad\_}\mu_{wI} = \text{ad\_}\mu_{wI} - w(u,v)(J(u,v) - \mu_{wJ})\text{ad\_}\Sigma_{\text{corr}}$
 6: end for
 7: $\text{ad\_}\mu_{wI} = \text{ad\_}\mu_{wI} - \frac{\mu_{wI}}{\sqrt{\frac{\Sigma_{wI^2}}{\Sigma_w} - \mu_{wI}^2}} \text{ad\_}\sigma_{wI}$
 8: $\text{ad\_}\Sigma_{wI^2} = \text{ad\_}\Sigma_{wI^2} + \frac{\Sigma_{wI^2}}{2\Sigma_{wI}\sqrt{\frac{\Sigma_{wI^2}}{\Sigma_w} - \mu_{wI}^2}} \text{ad\_}\sigma_{wI}$
 9: $\text{ad\_}\Sigma_{wI} = \frac{1}{\Sigma_w} \text{ad\_}\mu_{wI}$
10: for all $(u, v) \in \Omega$ do
11:     $\text{ad\_}I(u,v) = \text{ad\_}I(u,v) + 2w(u,v)I(u,v)\text{ad\_}\Sigma_{wI^2}$
12:     $\text{ad\_}I(u,v) = \text{ad\_}I(u,v) + w(u,v)\text{ad\_}\Sigma_{wI}$
13: end for
```

statements. Due to the inversion of the program flow, the volume is traversed from back to front in the adjoint algorithm. The derivative of vol is numerically approximated by central differences in the following way:

$$\frac{\partial \text{vol}}{\partial \mathbf{p}_{\text{obj}}} = \begin{pmatrix} \frac{\text{vol}\left(\mathbf{p}_{\text{obj}} + \begin{pmatrix}1 & 0 & 0\end{pmatrix}^T\right) - \text{vol}\left(\mathbf{p}_{\text{obj}} - \begin{pmatrix}1 & 0 & 0\end{pmatrix}^T\right)}{2} \\ \frac{\text{vol}\left(\mathbf{p}_{\text{obj}} + \begin{pmatrix}0 & 1 & 0\end{pmatrix}^T\right) - \text{vol}\left(\mathbf{p}_{\text{obj}} - \begin{pmatrix}0 & 1 & 0\end{pmatrix}^T\right)}{2} \\ \frac{\text{vol}\left(\mathbf{p}_{\text{obj}} + \begin{pmatrix}0 & 0 & 1\end{pmatrix}^T\right) - \text{vol}\left(\mathbf{p}_{\text{obj}} - \begin{pmatrix}0 & 0 & 1\end{pmatrix}^T\right)}{2} \end{pmatrix} \tag{5.23}$$

The (partly omitted) statements computing ad_$\mathbf{r}$ in Algorithm 4 stem from the elements of the rotation matrix $\mathbf{R}$, whose calculation is dependent on the rotation parameters $\mathbf{r}$.

In order to calculate the gradient in (5.23), the volume has to be sampled again. In Section 4.3, however, we discussed that loading data from memory is strongly unfavorable compared to computing it. To exploit the efficiency of contiguous memory reads, it would be desirable to traverse the volume only once to sample the rays *and* compute the derivatives of vol. This, in turn, requires the differentiated DRR rendering code to be evaluated along with the calculation of the DRR intensities $I(u, v)$. Yet, line 11 of Algorithm 4 is dependent on ad_$I(u, v)$, which constitutes the outcome of Algorithm 3,

Algorithm 4 The reverse mode of AD applied to the DRR rendering algorithm.

```
Require: $\mathbf{C}, \mathbf{R}, \mathbf{T}, \mathbf{H} \in M_4(\mathbb{R})$; $n_{\text{smpl}} \in \mathbb{N}$; $\text{ad\_}\mathbf{t}, \text{ad\_}\mathbf{r} = \emptyset$
 1: for all $(u, v) \in \Omega$ do
 2:   $\tilde{\mathbf{f}}_{\text{cam}} = \mathbf{H}\tilde{\mathbf{f}}_{\text{win}}$
 3:   $\tilde{\mathbf{b}}_{\text{cam}} = \mathbf{H}\tilde{\mathbf{b}}_{\text{win}}$
 4:   $\dots$
 5:   $\mathbf{s}_{\text{obj}} = \mathbf{C}\mathbf{R}^{-1}\mathbf{T}^{-1}\mathbf{C}^{-1}\mathbf{s}_{\text{world}}$
 6:   $\mathbf{x}_{\text{obj}} = \mathbf{R}^{-1}\mathbf{x}_{\text{world}}$
 7:   $\mathbf{p}_{\text{obj}} = \mathbf{s}_{\text{obj}} + (n_{\text{smpl}} - 1)\mathbf{x}_{\text{obj}}$
 8:   $\text{ad\_}\mathbf{p}_{\text{obj}} = \text{ad\_}\mathbf{x}_{\text{obj}} = \text{ad\_}\mathbf{R} = \emptyset$
 9:   for $i = 1$ to $n_{\text{smpl}}$ do
10:     $\text{ad\_}\mathbf{x}_{\text{obj}} = \text{ad\_}\mathbf{x}_{\text{obj}} + \text{ad\_}\mathbf{p}_{\text{obj}}$
11:     $\text{ad\_}\mathbf{p}_{\text{obj}} = \text{ad\_}\mathbf{p}_{\text{obj}} + \frac{\partial \text{vol}}{\partial \mathbf{p}_{\text{obj}}}(\mathbf{p}_{\text{obj}})\, \text{ad\_}I(u, v)$
12:     $\mathbf{p}_{\text{obj}} = \mathbf{p}_{\text{obj}} - \mathbf{x}_{\text{obj}}$
13:   end for
14:   $\text{ad\_}\mathbf{s}_{\text{obj}} = \text{ad\_}\mathbf{p}_{\text{obj}}$
15:   $\text{ad\_}\mathbf{t} = \text{ad\_}\mathbf{t} - \mathbf{R}\,\text{ad\_}\mathbf{s}_{\text{obj}}$
16:   $\text{ad\_}\mathbf{R}_{[1,:]} = \text{ad\_}\mathbf{R}_{[1,:]} + \mathbf{s}_{\text{world}}.x\,\text{ad\_}\mathbf{x}_{\text{obj}}$
17:   $\text{ad\_}\mathbf{R}_{[2,:]} = \text{ad\_}\mathbf{R}_{[2,:]} + \mathbf{s}_{\text{world}}.y\,\text{ad\_}\mathbf{x}_{\text{obj}}$
18:   $\text{ad\_}\mathbf{R}_{[3,:]} = \text{ad\_}\mathbf{R}_{[3,:]} + \mathbf{s}_{\text{world}}.z\,\text{ad\_}\mathbf{x}_{\text{obj}}$
19:   $\text{ad\_}\mathbf{R}_{[1,:]} = \text{ad\_}\mathbf{R}_{[1,:]} + (\mathbf{T}^{-1}\mathbf{C}^{-1}\mathbf{x}_{\text{world}}).x\,\text{ad\_}\mathbf{s}_{\text{obj}}$
20:   $\text{ad\_}\mathbf{R}_{[2,:]} = \text{ad\_}\mathbf{R}_{[2,:]} + (\mathbf{T}^{-1}\mathbf{C}^{-1}\mathbf{x}_{\text{world}}).y\,\text{ad\_}\mathbf{s}_{\text{obj}}$
21:   $\text{ad\_}\mathbf{R}_{[3,:]} = \text{ad\_}\mathbf{R}_{[3,:]} + (\mathbf{T}^{-1}\mathbf{C}^{-1}\mathbf{x}_{\text{world}}).z\,\text{ad\_}\mathbf{s}_{\text{obj}}$
22:   $\text{ad\_}r_y = \text{ad\_}r_y + (-sin(r_y)cos(r_z))\text{ad\_}\mathbf{R}_{[1,1]}$
23:   $\text{ad\_}r_z = \text{ad\_}r_z + (-cos(r_y)sin(r_z))\text{ad\_}\mathbf{R}_{[1,1]}$
24:
25:   $\dots$
26:
27:   $\text{ad\_}r_x = \text{ad\_}r_x + (-sin(r_x)cos(r_y))\text{ad\_}\mathbf{R}_{[3,3]}$
28:   $\text{ad\_}r_y = \text{ad\_}r_y + (-cos(r_x)sin(r_y))\text{ad\_}\mathbf{R}_{[3,3]}$
29: end for
```

thus producing a circular dependency. When closely inspecting Algorithm 4, though, one can observe that its results ad_$\mathbf{t}$ and ad_$\mathbf{r}$, that is, the adjoints of the translation and rotation parameters we are trying to recover, are linearly dependent on ad_I since all intermediate adjoint variables are the result of either a scalar multiplication or a dot product involving their adjoint predecessors. We make use of this observation by omitting ad_I in the evaluation of Algorithm 4, which yields the modified Algorithm 5. Instead of accumulating ad_$\mathbf{t}$ and ad_$\mathbf{r}$ while iterating through Ω, we store their pixel-wise preliminary values in the two-dimensional arrays ad_$\mathbf{t}(u, v)$ and ad_$\mathbf{r}(u, v)$. Furthermore, we add the statements accumulating the DRR intensities (line 17 and 18 in Algorithm 1) to the derivative code in order to obtain the intensities $I(u, v)$. Therewith we can calculate Σ_{corr}, σ_{wI}, Σ_{wI^2}, Σ_{wI}, and μ_{wI} (see Section 4.4.2). This provides the basis for evaluating Algorithm 3, which yields the pixel-wise adjoints ad_$I(u, v)$. We incorporate these values by multiplying the preliminary gradients ad_$\mathbf{t}(u, v)$ and ad_$\mathbf{r}(u, v)$ with ad_$I(u, v)$ on a per-pixel basis. Summing up the arrays ad_t_x, ad_t_y, ad_t_z, ad_r_x, ad_r_y, and ad_r_z over Ω eventually yields the six-valued gradient sought-after (see Algorithm 6).

5.3.1 GPU-Based Evaluation

The calculation of the gradients as described above can be ported to the GPU as-is, where blocks within loops over Ω are evaluated by means of fragment shader programs. Pixel-wise values to be filed for future use, that is, array writes with index (u, v), are rendered to textures which are subsequently provided as shader inputs. Wherever arrays are summed up over Ω, texture reduction as described in Section 4.4.2 is used. However, there is still room for optimizations. An as-is implementation would require five rendering and four texture reduction passes:

- Computing $I(u, v)$, $I(u, v)^2$, and the preliminary values of ad_$\mathbf{t}(u, v)$ and ad_$\mathbf{r}(u, v)$ omitting ad_$I(u, v)$ (Algorithm 5).
- Reducing $w(u, v)I(u, v)$ and $w(u, v)I(u, v)^2$ to sum yielding Σ_{wI} and Σ_{wI^2}.
- Computing $\mathrm{corr}(u, v)$ (equation 4.8). Note that corr is not required for the gradient itself, but to compute the similarity measure required by the optimizer.
- Reducing $\mathrm{corr}(u, v)$ to sum yielding Σ_{corr}.

Algorithm 5 The modified version of Algorithm 4 omitting ad_$I(u,v)$.

```
Require: $\mathbf{C},\mathbf{R},\mathbf{T},\mathbf{H} \in M_4(\mathbb{R})$ ; $n_{\text{smpl}} \in \mathbb{N}$; ad_$\mathbf{t}$, ad_$\mathbf{r} = \emptyset$
 1: for all $(u,v) \in \Omega$ do
 2:   $\tilde{\mathbf{f}}_{\text{cam}} = \mathbf{H}\tilde{\mathbf{f}}_{\text{win}}$
 3:   $\tilde{\mathbf{b}}_{\text{cam}} = \mathbf{H}\tilde{\mathbf{b}}_{\text{win}}$
 4:   ...
 5:   $\mathbf{s}_{\text{obj}} = \mathbf{C}\mathbf{R}^{-1}\mathbf{T}^{-1}\mathbf{C}^{-1}\mathbf{s}_{\text{world}}$
 6:   $\mathbf{x}_{\text{obj}} = \mathbf{R}^{-1}\mathbf{x}_{\text{world}}$
 7:   $\mathbf{p}_{\text{obj}} = \mathbf{s}_{\text{obj}} + (n_{\text{smpl}} - 1)\mathbf{x}_{\text{obj}}$
 8:   ad_$\mathbf{p}_{\text{obj}}$ = ad_$\mathbf{x}_{\text{obj}}$ = ad_$\mathbf{R} = \emptyset$
 9:   for $i = 1$ to $n_{\text{smpl}}$ do
10:     $I(u,v) = I(u,v) + \text{vol}(\mathbf{p}_{\text{obj}})$
11:     ad_$\mathbf{x}_{\text{obj}}$ = ad_$\mathbf{x}_{\text{obj}}$ + ad_$\mathbf{p}_{\text{obj}}$
12:     ad_$\mathbf{p}_{\text{obj}}$ = ad_$\mathbf{p}_{\text{obj}}$ + $\frac{\partial \text{vol}}{\partial \mathbf{p}_{\text{obj}}}(\mathbf{p}_{\text{obj}})$
13:     $\mathbf{p}_{\text{obj}} = \mathbf{p}_{\text{obj}} - \mathbf{x}_{\text{obj}}$
14:   end for
15:   ad_$\mathbf{s}_{\text{obj}}$ = ad_$\mathbf{p}_{\text{obj}}$
16:   ad_$\mathbf{t}(u,v)$ = ad_$\mathbf{t}(u,v)$ − $\mathbf{R}$ad_$\mathbf{s}_{\text{obj}}$
17:
18:   ...
19:
20:   ad_$r_y(u,v)$ = ad_$r_y(u,v) + (-sin(r_y)cos(r_z))$ad_$\mathbf{R}_{[1,1]}$
21:   ad_$r_z(u,v)$ = ad_$r_z(u,v) + (-cos(r_y)sin(r_z))$ad_$\mathbf{R}_{[1,1]}$
22:
23:   ...
24:
25: end for
```

Algorithm 6 Computing the final pixel-wise gradients with respect to the transformation parameters

```
1: for all $(u,v) \in \Omega$ do
2:   ad_$t_x$ = ad_$t_x$ + ad_$t_x(u,v)$ad_$I(u,v)$
3:   ad_$t_y$ = ad_$t_y$ + ad_$t_y(u,v)$ad_$I(u,v)$
4:   ad_$t_z$ = ad_$t_z$ + ad_$t_z(u,v)$ad_$I(u,v)$
5:   ad_$r_x$ = ad_$r_x$ + ad_$r_x(u,v)$ad_$I(u,v)$
6:   ad_$r_y$ = ad_$r_y$ + ad_$r_y(u,v)$ad_$I(u,v)$
7:   ad_$r_z$ = ad_$r_z$ + ad_$r_z(u,v)$ad_$I(u,v)$
8: end for
```

- First loop in Algorithm 3.
- Reducing $-w(u,v)(J(u,v)-\mu_{wJ})\text{ad\_}\Sigma_{\text{corr}}$ to sum to obtain $\text{ad\_}\mu_{wI}$.
- Second loop in Algorithm 3.
- Computing the final values of $\text{ad\_}\mathbf{t}(u,v)$ and $\text{ad\_}\mathbf{r}(u,v)$ (Algorithm 6).
- Reducing $\text{ad\_}\mathbf{t}(u,v)$ and $\text{ad\_}\mathbf{r}(u,v)$ to sum.

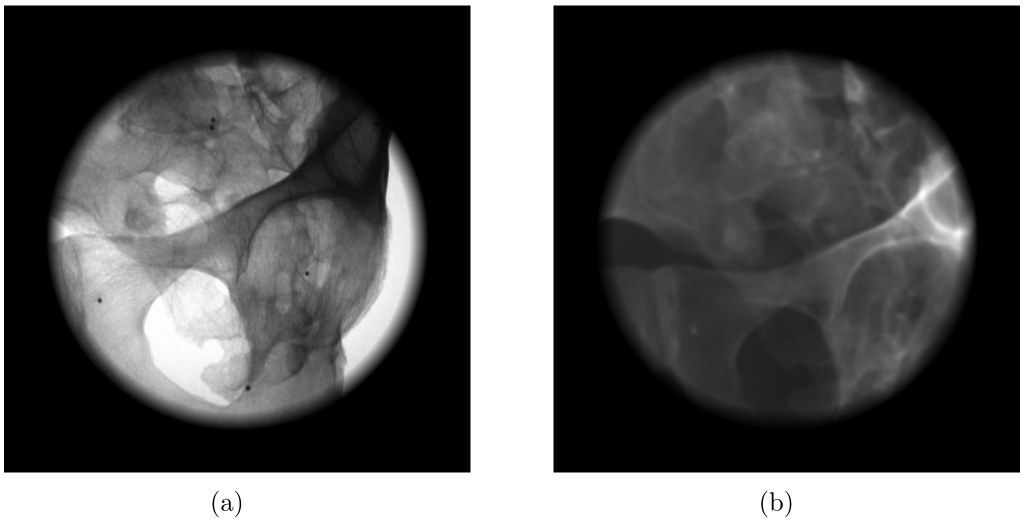


Figure 5.1: A sample X-ray image $w(u,v)J(u,v)$ and a masked DRR $w(u,v)I(u,v)$

By means of some code restructuring, we reduce the cost to three rendering and texture reduction passes each. Computing only the NCC measure, for example, to approximate the gradient numerically, on the other hand, requires two rendering passes (one to compute $w(u,v)I(u,v)$ and $w(u,v)I(u,v)^2$, and one to compute $\text{corr}(u,v)$) and two texture reduction passes. However, to obtain the whole six-valued gradient by means of central differences, the NCC computation routine has to be invoked twelve times. The restructured algorithm looks as follows:

1. **First rendering pass**: Compute $w(u,v)I(u,v)$ (see Figure 5.1(b)), $w(u,v)I(u,v)^2$, and the preliminary values of $\text{ad\_}\mathbf{t}(u,v)$ and $\text{ad\_}\mathbf{r}(u,v)$ (see Figure 5.2(a) and 5.2(b))
2. **First reduction pass**: Reduce $w(u,v)I(u,v)$ and $w(u,v)I(u,v)^2$ to sum.

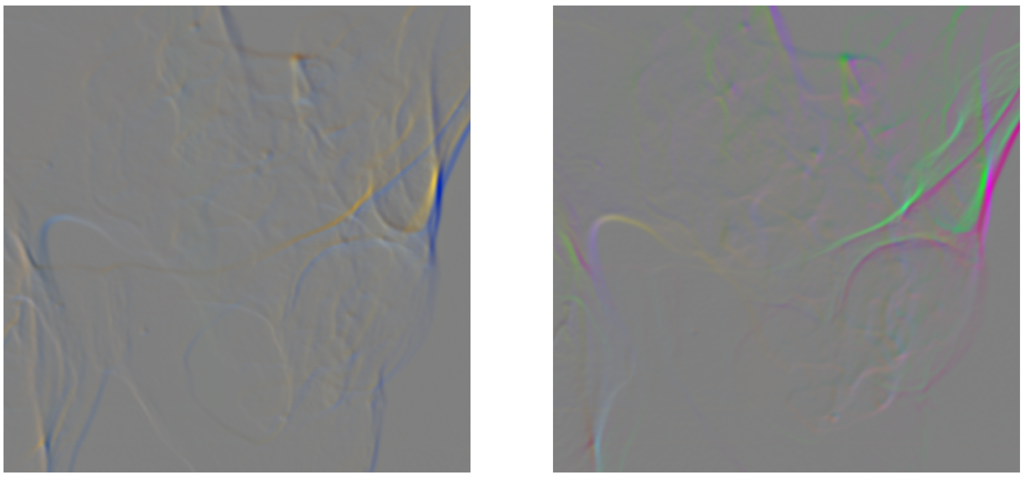

(a) (b)

Figure 5.2: The preliminary translation and rotation gradient images ad_$\mathbf{t}(u, v)$ and ad_$\mathbf{r}(u, v)$ as returned by the first rendering pass of the restructured algorithm. They encode the potential change of the DRR in Figure 5.1(b) when $\mathbf{t}$ or $\mathbf{r}$ vary.

3. **Host**: Read Σ_{wI} and Σ_{wI^2} back to CPU main memory and compute μ_{wI} as well as σ_{wI}

4. **Second rendering pass**: Compute $\text{corr}(u, v)$ and $w(u, v)(J(u, v) - \mu_{wJ})$ (see Figure 5.3(a) and 5.3(b)).

5. **Second texture reduction**: Reduce $\text{corr}(u, v)$ and $w(u, v)(J(u, v) - \mu_{wJ})$ to sum.

6. **Host**. Read Σ_{corr} and the sum of weighted X-ray differences from mean back to CPU main memory. Compute the NCC measure, ad_Σ_{corr}, ad_σ_{wI}, ad_μ_{wI} (using the sum of X-ray differences), ad_Σ_{wI^2}, and ad_Σ_{wI}.

7. **Third rendering pass**: Compute ad_$I(u, v)$ using ad_Σ_{corr}, ad_Σ_{wI^2}, and ad_Σ_{wI}. Multiply ad_$\mathbf{t}(u, v)$ and ad_$\mathbf{r}(u, v)$ with ad_$I(u, v)$ to get the pixel-wise gradients (see Figure 5.4(a) and 5.4(b)).

8. **Third reduction pass**: Reduce ad_$\mathbf{t}(u, v)$ and ad_$\mathbf{r}(u, v)$ to sum.

9. **Host**: Read ad_t_x, ad_t_y, ad_t_z, ad_r_x, ad_r_y, and ad_r_z back to CPU main memory.

These values constitute the elements of the cost function's gradient with respect to the six model transformation parameters $\mathbf{x}$ at the position $\mathbf{x}_i$ in parameter space:

$$\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i) = \begin{pmatrix} \text{ad\_}t_x & \text{ad\_}t_y & \text{ad\_}t_z & \text{ad\_}r_x & \text{ad\_}r_y & \text{ad\_}r_z \end{pmatrix}^T \tag{5.24}$$

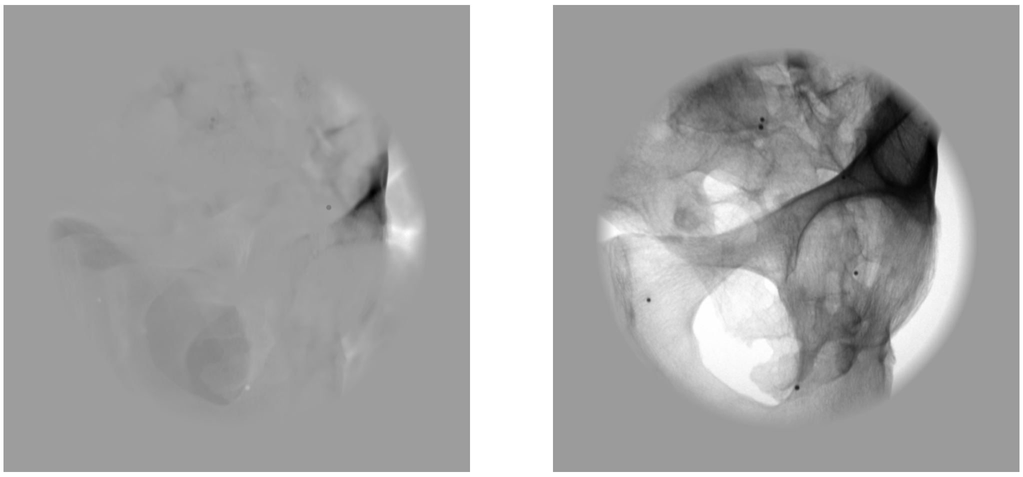

(a) (b)

Figure 5.3: The correlation image $\text{corr}(u, v)$, which provides the basis for computing the NCC measure, and an image depicting the X-ray differences from mean $w(u, v)(J(u, v) - \mu_{wJ})$.

The final gradient images in Figure 5.4(a) and 5.4(b) show the pixel-wise gradients with respect to $\mathbf{t}$ and $\mathbf{r}$ encoded by colors. They were calculated on the basis of the X-ray image in Figure 5.1(a) and the DRR in Figure 5.1(b). Note that the values of $\text{ad\_}\mathbf{t}(u, v)$ and $\text{ad\_}\mathbf{r}(u, v)$ are mapped to the intensity range such that the null vector is represented as the grey tone equidistant to white (255, 255, 255) and black (0, 0, 0). Hence, an intensity of 128 and above represents a positive gradient, an intensity below a negative one. The red channel contains the gradients in x, that is, either the translation along or the rotation around the x-axis, which corresponds to the axis through the patient's shoulders. The green channel contains the gradients with respect to the axis from the back to the front side of the patient, that is, the y-axis. The z-axis, which progresses from the patient's head to their feet, is represented by the blue channel. The blue and yellow regions in Figure 5.4(a), which stem from a respectively high and low intensity in the blue channel, indicate a rapid change of the cost function in the t_z dimension. Figure 5.4(b), on the other hand, suggests a strong dynamics in r_y since magenta and green are salient. Indeed, the DRR in Figure 5.1(b) was generated with a translation of -10 millimeters in z and a rotation of 5 degrees in y.

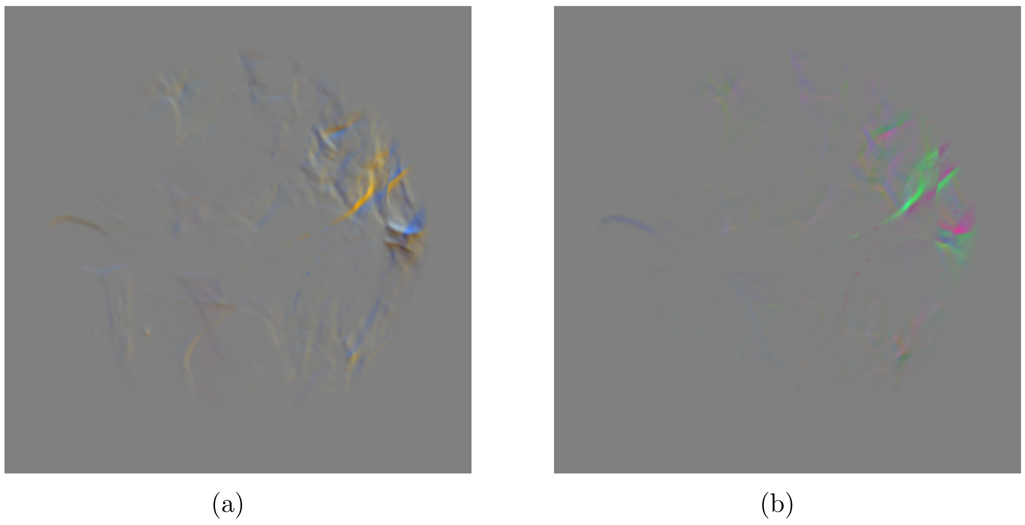


Figure 5.4: The final translation and rotation gradient images ad_$\mathbf{t}(u, v)$ and ad_$\mathbf{r}(u, v)$ computed from the preliminary gradient images in Figure 5.2(a) and 5.2(b), and ad_$I(u, v)$. The R, G, and B channel correspond to the x, y, and z-coordinate respectively.

When using more than one X-ray views to conduct the registration, the above procedure is carried out for every X-ray/weight mask pair and the resulting gradient vectors are summed up to form the actual step vector required by the optimization routine.

Performing the evaluation of derivative code on the GPU entails a tremendous registration speedup compared to approaches based on CPU-computation or numerical approximation of the gradient and code restructuring can additionally boost efficiency. However, both measures trade the flexibility of algorithmic differentiation for performance. One of the major advantages of AD is the *automatic* generation of derivative code, which greatly benefits the maintainability of the program. After applying changes to the original code, a rerun of the AD algorithm ensures the integrity of the program. When porting parts of the calculation to the GPU and performing more complex optimizations, on the other hand, it is currently indispensable for the programmer to engage with the derivative code. However, since efficient computation is one of our main goals and we consider the registration algorithm to be largely invariant, we accept that fact and focus on maximizing speed.

Chapter 6

Results

6.1 Implementation and Operation

The registration algorithm was implemented on a Linux machine using *C++* as the host programming language, the *OpenGL* graphics library [OpenGL, 2007] to communicate with the GPU, and the *Cg* language [NVidia, 2007a] to implement the fragment shader programs. A *Fortran* implementation of the *L-BFGS-B* algorithm (see Section 3.6.3 and [Zhu et al., 1997]) is used for optimization. The camera calibration procedure is currently not integrated in the main application. It is implemented by a set of *Matlab* programs which compute the *intrinsic* camera parameters from multiple views of the calibration target, undistort the X-ray images, and determine the pose of the camera for each image (see Section 3.4). The programs are similar to those presented in [Bouguet, 2007], yet refined by some methods from robust statistics.

The experiments were carried out on a Linux machine, which is equipped with an Intel® Core™2 Quad processor running at 2.66 GHz and a NVidia® GeForce® 8800 GTX graphics board with 768 MB of GPU memory.

6.2 Validation

In order to assess the quality of the alignment, a standardized method to determine the registration error has to be applied. The mean target registration error (mTRE) is a common measure when no salient features are available in the images. In [Van de Kraats et al., 2004], the evaluation of the mTRE by means of a set of evenly distributed points p_i inside a region of interest P within the volume is proposed. It is calculated as the mean distance between the points p_i mapped to the world by the estimated transformation $\mathbf{T}_{\text{reg}}$ and the ground truth transformation $\mathbf{T}_{\text{ground}}$ respectively:

$$\text{mTRE}(P, \mathbf{T}_{\text{reg}}, \mathbf{T}_{\text{ground}}) = \frac{1}{n} \sum_{i=0}^{n} ||\mathbf{T}_{\text{reg}} p_i - \mathbf{T}_{\text{ground}} p_i|| \tag{6.1}$$

We define our region of interest to be a cube of size $10 \times 10 \times 10$ centimeters, where we evenly distribute 1000 points on a regular grid.

To determine the *capture range* of the registration algorithm, the criteria for a successful registration and the percentage of allowed failures has to be defined. We consider registrations with a mTRE below 1.5 millimeters as successful and require 90 percent of all registrations to be successful in order for a certain initial displacement to be within the capture range. The evaluation method presented in [Van de Kraats et al., 2004] uses the mTRE as a measure for the initial displacement. Random starting positions are uniformly divided into a set of initial mTRE intervals in the following way:

1. Define a set of mTRE intervals, for example, $0-1, 1-2, \ldots, 69-70$ millimeters in our case.

2. Determine the value range that produces a maximum mTRE of 1 millimeter for each transformation parameter separately. For the translation parameters, this is always 1 millimeter. For the sake of simplicity, a linear relationship between the rotation parameters and the resulting mTRE is assumed. For our choice of P, however, this yields an angle of 1.5 degrees, which amounts to a maximum variation of 105 degrees for the highest initial mTRE interval. As opposed to a displacement of 70 millimeters, such an orientation of the patient is far from being feasible, though. Hence, we use the angle that produces a mTRE of 0.3 millimeter instead, which yields a maximum deviation of 30.6 degrees from ground truth.

3. For each interval generate random starting positions from the transformation parameter ranges that produce a maximum mTRE of the intervals upper bound. Discard those starting positions which yield a mTRE beyond the current interval's bounds. This is done until each interval contains the same predefined number of starting positions, in our case 5 or 10 respectively.

In order to evaluate our registration approach, we use two CT scans with corresponding X-ray images. The first dataset represents the image of a cadaveric pelvic bone with $512 \times 512 \times 346$ voxels and a voxel spacing of $0.602 \times 0.602 \times 0.600$ millimeters. Ground truth was calculated from metal markers, which were applied to the bone before being imaged. The corresponding X-ray images were acquired for three different camera poses and have 1024×1024 pixels. The second dataset is the image of a patient's abdomen with $512 \times 288 \times 512$ voxels and a voxel spacing of $0.713 \times 1 \times 0.713$ millimeters. X-ray images with 512×512 pixels are available for three different camera poses. For the latter dataset no exact ground truth information is available. Hence, we first estimate it as the mean of some final parameter vectors obtained for registrations with small initial displacements. Then we carry out the registration evaluation as described above and approximate ground truth by the mean of all final parameter vectors that have a lower mTRE from our estimate than 2.5 millimeters. Eventually, we calculate the mTRE of the final parameter vectors with the updated approximation of ground truth and determine the capture ranges.

To make the algorithmic differentiation results comparable, we carried out the same validation procedure with a custom-built registration application based on numerical approximation of the gradient. Like for the algorithmic differentiation approach, all computationally and memory intensive parts, namely DRR rendering, similarity measuring, and texture reduction, were computed on the GPU. The gradient was determined by means of central differences around the current position in transformation parameter space, where each parameter was varied by a fixed δ separately.

In the following two sections the results of our experimental evaluation of both gradient calculation approaches in terms of convergence and accuracy (Section 6.3) and performance (Section 6.4) are presented. The results are subsequently discussed in Section 7.2.

6.3 Convergence and Accuracy

For the pelvic bone registrations were carried out for 70 initial mTRE intervals containing 5 random starting positions. Our approach achieved a mean final mTRE of 0.24 millimeters for all successful registrations, that is, registrations with a final mTRE below 1.5. A total of 257 (73.4%) registrations were successful. Figures 6.1 and 6.2 show the relationship between the initial and the final mTRE of single registration passes by means of scatter plots. Each marker represents one experimental registration. Its x-coordinate specifies the initial mTRE, where the y-coordinate depicts the respective final mTRE. Figure 6.3 shows the progression of the percentage of successful registrations. Each discrete position on the x-axis represents an upper bound for the initial mTRE. The corresponding y-coordinate states the percentage of registrations within the initial mTRE range $[0, x]$ which converged successfully. We will refer to it as *capture range plot.* Using the 90 percent criteria, the algorithmic differentiation approach reaches a capture range of 49 millimeters, which means that 90 percent of all registrations within the initial mTRE interval [0,49] (490 in number) had a final mTRE below 1.5 millimeters.

The numerical approximation approach converged successfully 260 times (74.3%), yielding a mean final mTRE of 0.1 millimeters for the pelvic bone data. The capture range amounted to 48 millimeters. Figures 6.4 and 6.5 show mTRE scatter plots. The capture range plot is depicted in Figure 6.3.

The same evaluation procedure was performed with the CT scan of a patient's abdomen and three corresponding X-ray images, yet with 10 starting positions per initial mTRE interval. The algorithmic differentiation approach converged 338 times (48.3%), yielding a mean final mTRE of 0.6 millimeters and a capture range of 33 millimeters. Figures 6.7 and 6.8 show scatter plots of the mTRE, where Figure 6.9 depicts the capture range plot.

For the approach based on numerical approximation of the gradient, 381 (54.4%) registrations were successful. The mTRE of this registrations averaged at 0.2 millimeters and the capture range amounted to 35 millimeters. Figures 6.10, 6.11, and 6.12 show mTRE scatter and capture range plots for these experiments.

The results of the convergence and accuracy measurements are summarized in Table 6.1.

	Accuracy (mm)		**Success (%)**		**Capture range (mm)**	
Dataset	**NA**	**AD**	**NA**	**AD**	**NA**	**AD**
Pelvis	0.1	0.24	74.3	73.4	48	49
Abdomen	0.2	0.6	54.4	48.4	35	33

Table 6.1: Summary of the convergence and accuracy results for both datasets. The accuracy is given as the mean final mTRE of successful registrations in millimeters. The capture range represents the maximum initial mTRE for which 90 percent of all registrations are successful. All values are provided for numerical approximation of the gradient (**NA**) and algorithmic differentiation (**AD**).

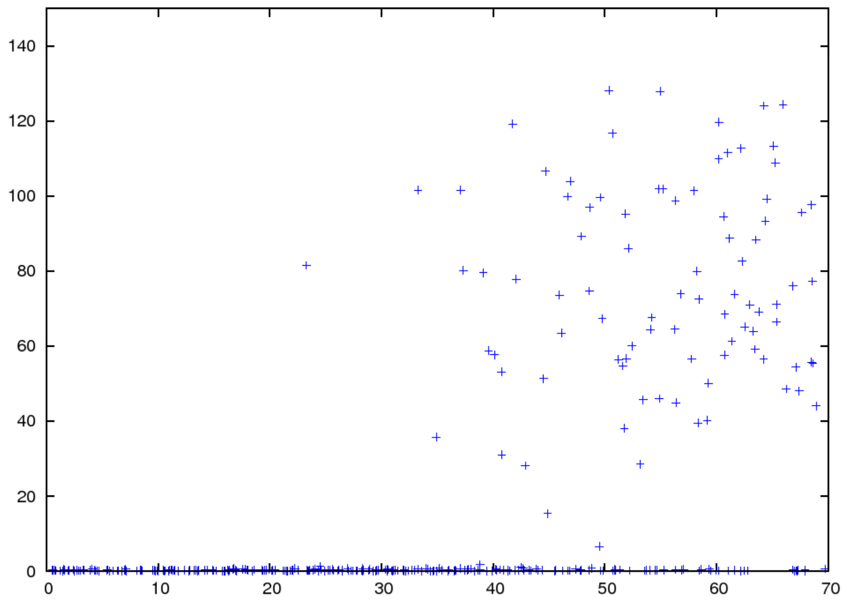


Figure 6.1: mTRE scatter plot of the registration experiments carried out with the pelvic bone data using algorithmic differentiation. (all values given in millimeters).

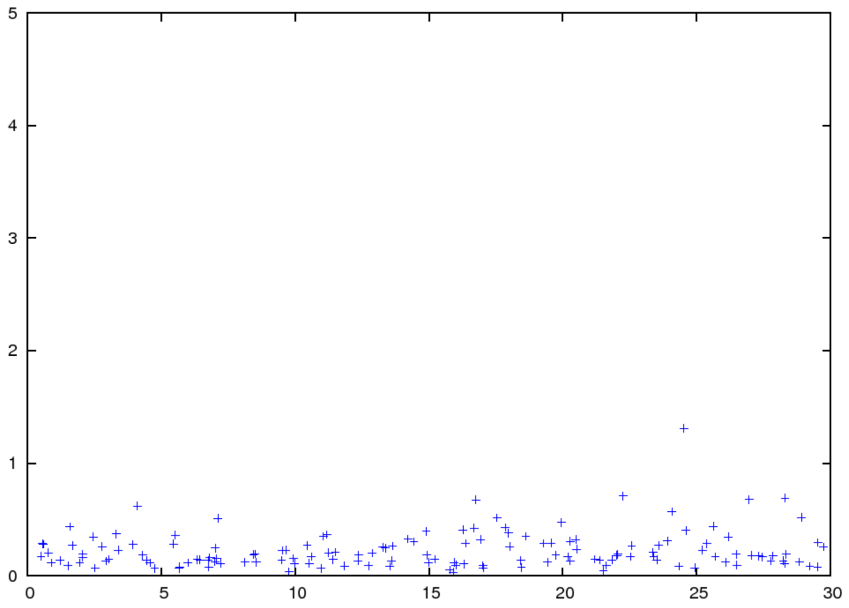


Figure 6.2: Detail of Figure 6.1 showing only a limited range of initial displacements (in millimeters).

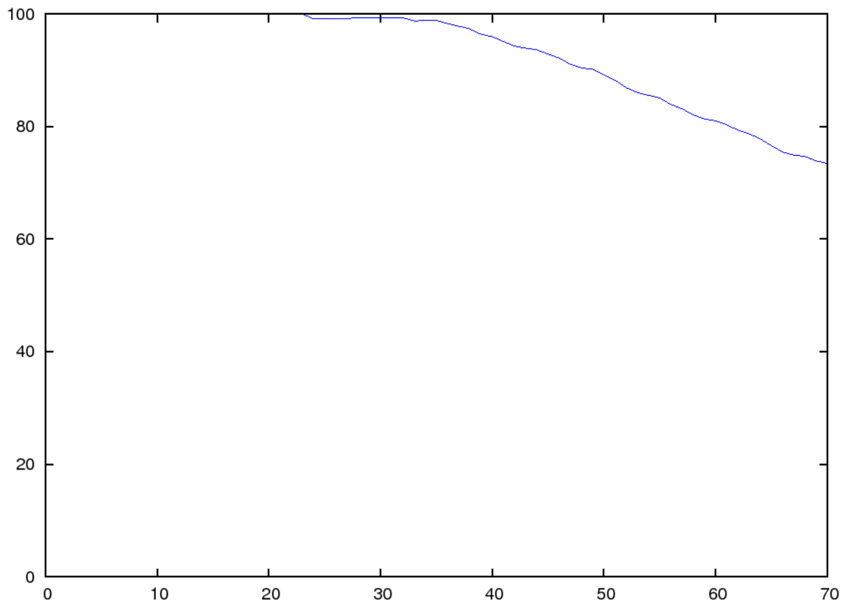


Figure 6.3: The capture range plot for the pelvic bone data and algorithmic differentiation depicting the percentage of registrations within the initial mTRE range $[0, x]$ which was successful

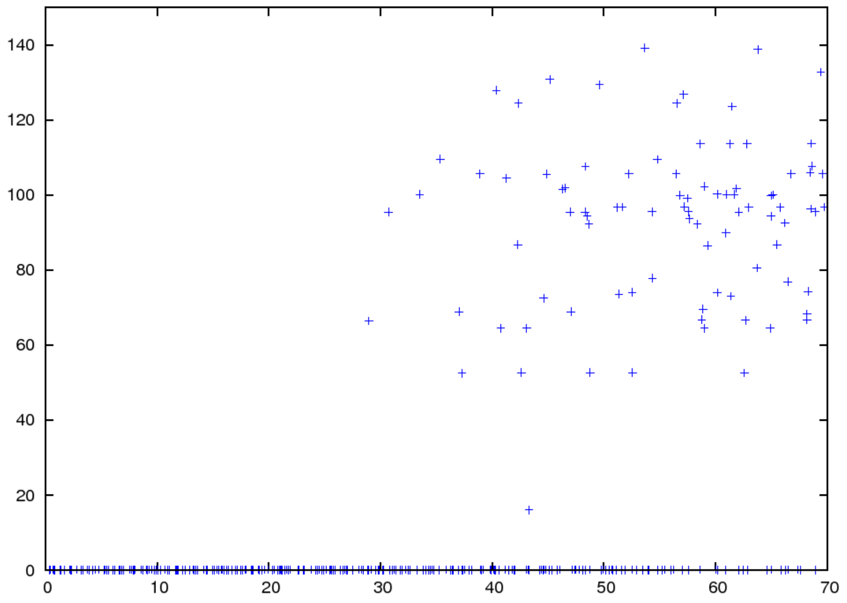


Figure 6.4: The mTRE scatter plot for the experiments conducted with the pelvic bone data using numerical approximation of the gradient.

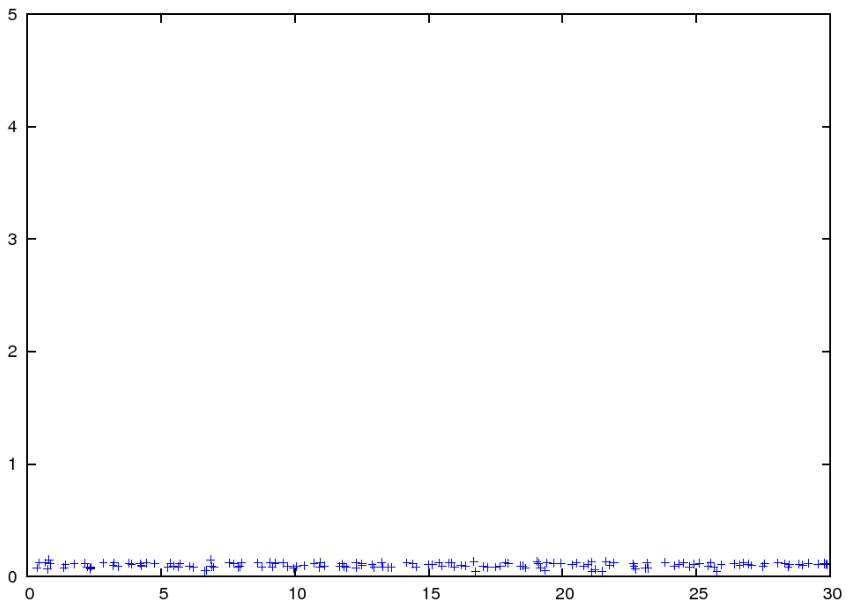


Figure 6.5: Detail of Figure 6.4 showing only a limited range of initial displacements (in millimeters).

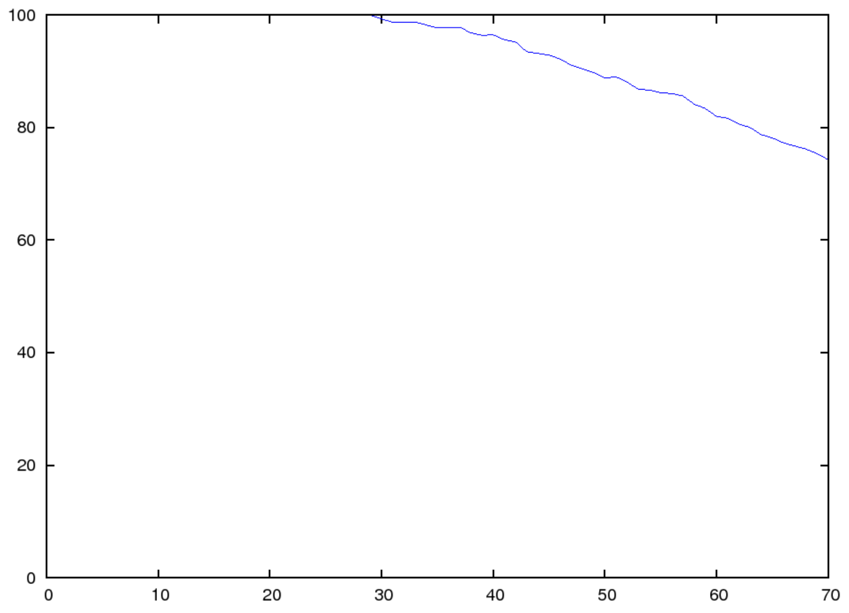


Figure 6.6: The capture range plot for the pelvic bone data and numerical approximation of the gradient.

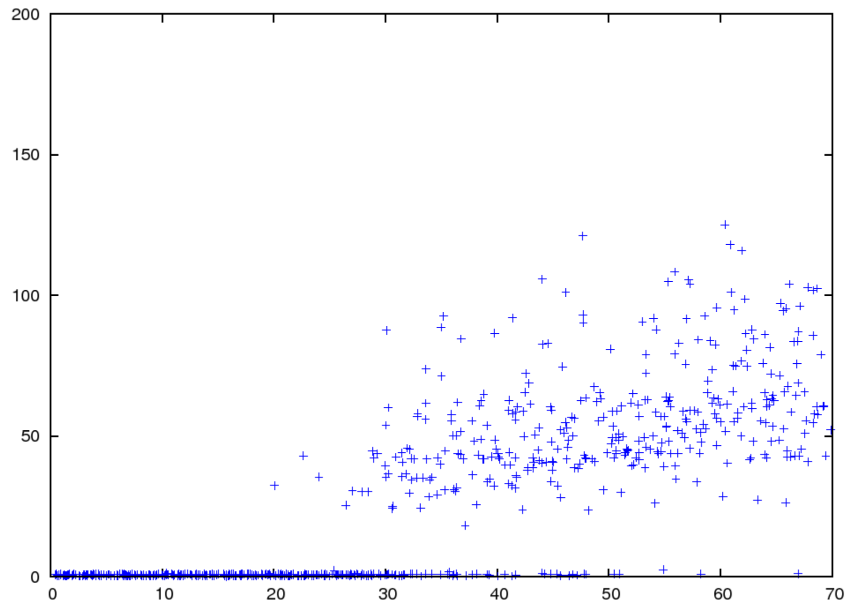


Figure 6.7: The mTRE scatter plot for the experiments conducted with the human abdomen data using algorithmic differentiation.

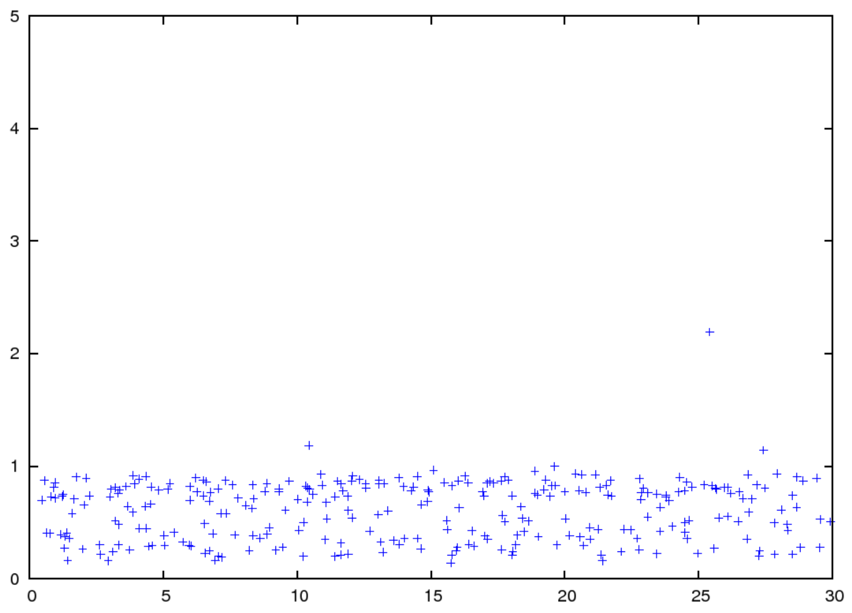


Figure 6.8: Detail of Figure 6.7 showing only a limited range of initial displacements (in millimeters).

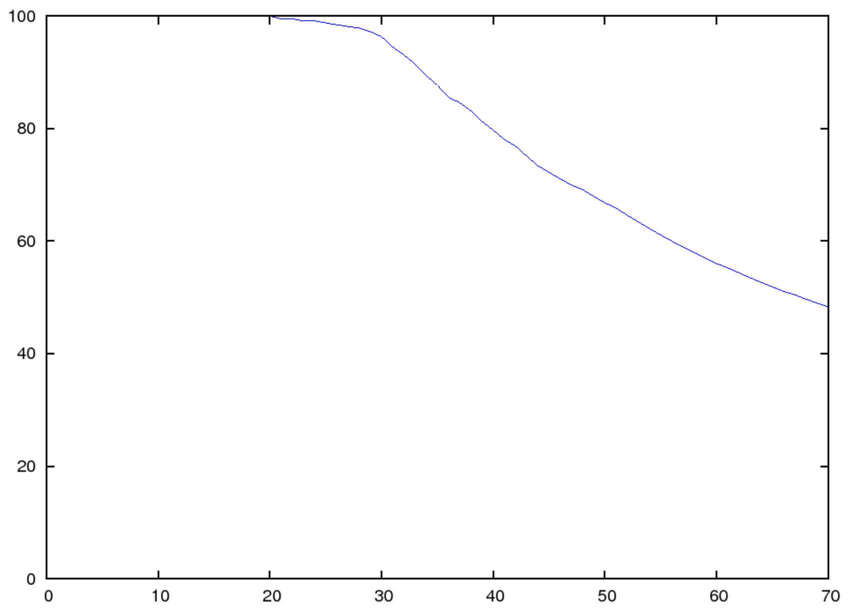


Figure 6.9: The capture range plot for the human abdomen data and algorithmic differentiation.

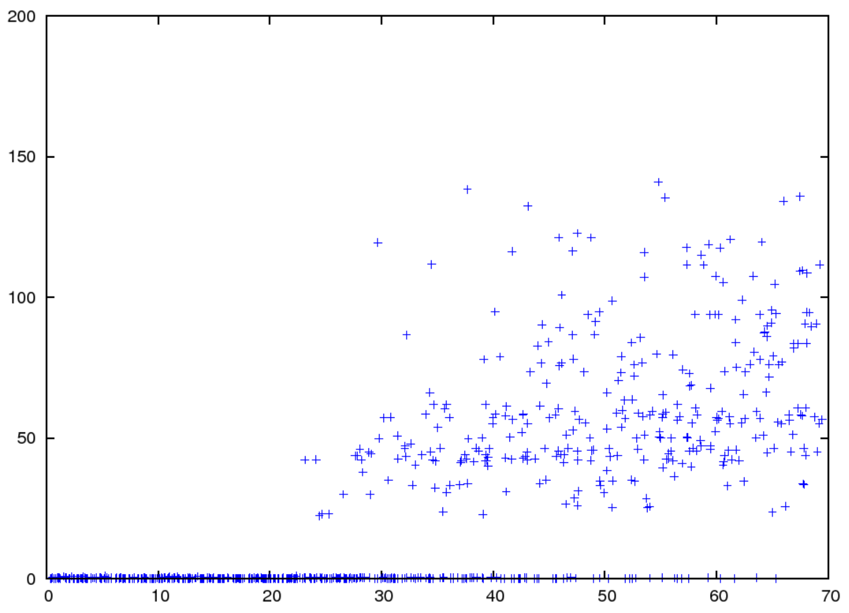


Figure 6.10: The mTRE scatter plot for the experiments conducted with the human abdomen data using numerical approximation of the gradient.

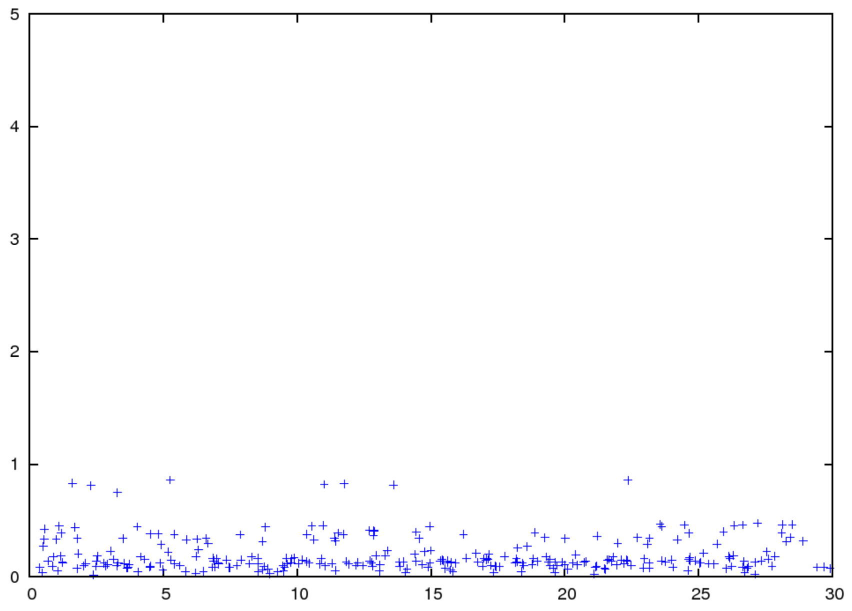


Figure 6.11: Detail of Figure 6.10 showing only a limited range of initial displacements (in millimeters).

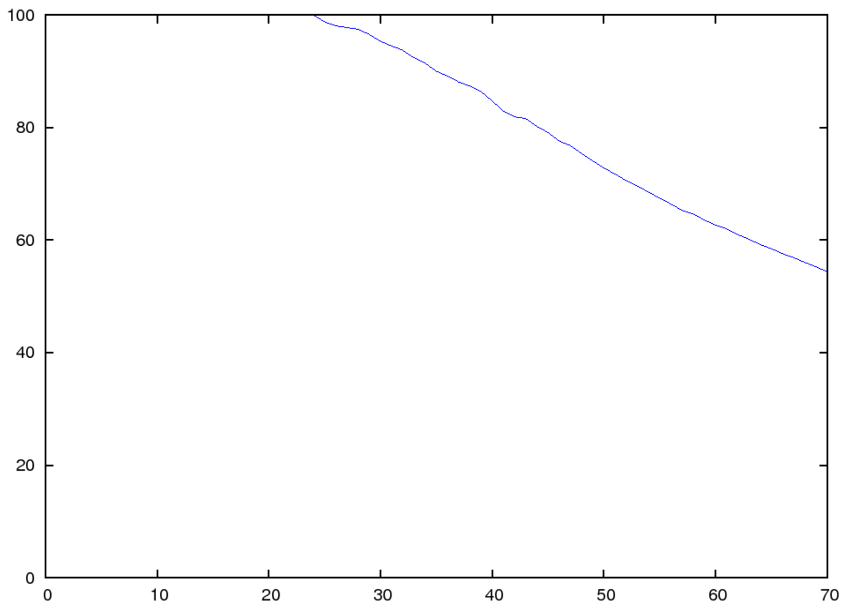


Figure 6.12: The capture range plot for the human abdomen data and numerical approximation of the gradient.

6.4 Performance

To determine the performance of our registration, we evaluated both the mean gradient calculation time and the mean registration time for the algorithmic differentiation and the numerical approximation approach. The latter uses central differences to approximate the gradient, thus requiring twelve evaluations of the cost function for the six model transformation parameters. Since both approaches include the local evaluation of the similarity measure, the number of cost function evaluations totals to thirteen. In order to get points in parameter space to calculate the gradient for, 700 random starting positions were generated as described in Section 6.2. For every position the gradient calculation procedure used in the above registration experiments was evoked. Eventually, the mean calculation time per gradient was calculated.

For the human abdomen data the numerical approximation approach yielded a mean gradient calculation time of 0.70 seconds, where the algorithmic differentiation took 0.29 seconds on average. For the registration series presented in Section 6.3 the numerical approximation approach required a mean time of 68.72 seconds to converge. Calculating the gradient by means of algorithmic differentiation, on the other hand, yielded a mean registration time of 28.52 seconds.

For the pelvic bone data computing a gradient required a mean of 1.87 seconds when approximating it numerically, and 0.88 seconds when using algorithmic differentiation. A mean registration took 199.35 seconds to converge for the former, and 96.35 seconds for the latter gradient computation approach.

Table 6.2 summarizes the time measurements carried out with the pelvic bone and the human abdomen data.

	Gradient (sec)				**Registration** (sec)		
Dataset	**NA**	**AD**	**NA/AD**	**AD/f**	**NA**	**AD**	**NA/AD**
Pelvis	1.87	0.88	2.13	6.12	199.35	96.35	2.07
Abdomen	0.70	0.29	2.41	5.39	68.72	28.52	2.41

Table 6.2: Mean time (in seconds) required to calculate a gradient and carry out a registration using both numerical approximation (**NA**) and algorithmic differentiation (**AD**). The **NA/AD** columns depict the ratio between the time required for numerical approximation and algorithmic differentiation. The column **AD/f** states the ratio between the runtime of evaluating the gradient using AD and evaluating the similarity measure f.

Chapter 7

Conclusion

7.1 Summary

This work addresses the problem of integrating high-quality preoperative patient scans into surgical interventions by means of intraoperative radiographs in an efficient, accurate, robust, and unobtrusive manner to facilitate minimally-invasive surgery. We presented a method to compute the gradient of an intensity-based similarity measure with respect to the rigid-body transformation parameters which is virtually insensitive to the size of the gradient in terms of performance. What is novel over prior art is the use of *algorithmic differentiation* to determine the exact gradient of the similarity measure efficiently. Moreover, the arithmetic power of modern graphics hardware is not only exploited to compute *digitally reconstructed radiographs* from the patient scans, but also to evaluate the objective function and to compute its gradient at the positions defined by the optimization procedure. Hence, all computationally and memory intensive parts of the registration are performed on the GPU, which represents a highly efficient computation engine for repetitive and parallelizable tasks. The optimization strategy was chosen to allow wide capture ranges and explicit bounds on the parameter space.

The set of registration experiments carried out with human abdomen and pelvic bone data showed a registration speed-up factor in the range between 2.1 and 2.4 for algorithmic differentiation compared to numerical approximation of the gradient. While both approaches performed about equally as far as capture ranges are concerned, the latter turned out to be superior in terms of accuracy. The registrations performed with a pelvic bone dataset, which does not contain any soft tissue, were generally more accurate and robust.

7.2 Discussion

The experimental evaluation of the presented 2D/3D registration method showed that the algorithmic differentiation approach is superior to numerical approximation in terms of speed. It could be shown that the ratio between the cost of evaluating the gradient and evaluating the cost function amounted to a maximum of about six, which approximately verifies the upper bound of five stated in [Griewank, 1989]. The evaluation of the derivatives within the inner volume sampling loop, such as the numerical approximation of the volume's gradient, turned out to be the main reason why this ratio could not be reduced further. We are confident, however, that a pre-computation of the volume's gradient will remove this bottleneck. Likewise, the benefit of reverse mode algorithmic differentiation might become more obvious for higher-dimensional gradients.

It turned out that 2D/3D registration based on algorithmic differentiation currently suffers a loss of accuracy compared to numerical approximation of the gradient. We ascribe that partly to the sensitivity of an exact gradient computation to noise, which may mislead the descent towards the minimum. Blurring the volume with a Gaussian filter might solve this problem, yet its effect has not been investigated yet. However, the final mean target registration error for successful registrations generally amounted to 0.6 millimeter and below on average, which we consider as satisfactory for targeting locations within the human body. As far as robustness, that is, the extent of the initial displacement for which registrations still converge reliably, is concerned, both approaches resemble. The capture ranges are superior to most of those stated in related literature (see Chapter 2), yet they are inherently hard to compare when stemming from different experimental settings. For example, we did not conduct experiments with the data provided by the *Image Science Institute*[1] since major modifications to our application in terms of masking would be required.

Additionally, the ratio between the mean registration times of both approaches roughly reflected the speed-up factor for single gradient computations, which we interpret as the gradient from algorithmic differentiation guiding the search towards an optimal match about as efficiently as a numerically approximated gradient.

The fact that the experiments carried out with the pelvic bone data exhibited higher accuracy and capture ranges suggests that the non-rigid structures present in the human abdomen data impair the NCC measure. We expect

[1]http://www.isi.uu.nl/Research/Databases/

similarity measures which are less strict about the nature of the underlying relationship between the images such as mutual information to perform better for patient data. However, the limited region of interest, which stems from certain parts of the images being masked out, is also likely to have an impact on the success of the registration, especially when salient features, such as bones, are covered by the mask.

It should be noted that the robustness and the accuracy of the registration are strongly dependent on the optimization strategy and its parameterization, which leaves a lot of room for further improvement. Likewise, there is a broad range of techniques that may considerably accelerate volume rendering and has not been investigated yet. However, what we regard as the focus of this work is the evidence that accurate gradients of the similarity measure can be computed in a much shorter period of time, which could be supplied. Hence, we consider our approach a solid basis for the implementation of a fast, accurate, and reliable application performing 2D/3D registration, yet also for related problems, such as 3D/3D or non-rigid registrations, to which the presented concept can be transferred.

7.3 Future Perspectives

The development of programmable graphics processors and technologies such as CUDA™ are very promising to facilitate the porting of general purpose applications to the GPU. As soon as native support for three-dimensional textures will be available, more elaborate similarity measures, that is, measures that do not rely on a pixel-wise evaluation such as mutual information can be implemented by means of GPU programs effortlessly. Additionally, it will be easier to evaluate the derivative code produced by reverse mode algorithmic differentiation on the GPU since the programming paradigm is becoming more flexible and features that are currently missing such as buffering intermediate values will be available. Therewith, the automatic creation and optimization of derivative GPU code, which would greatly improve the maintainability and extendibility of the application, will be more straightforward to implement.

As far as the registration itself is concerned, there is lot of room for improvement. As mentioned above, the accuracy and performance of the approach is strongly dependent on the optimization strategy itself. A thorough investigation of different gradient-based optimization strategies and the influence of their parameterization is to be carried out. For example, the performance and robustness benefit of supplying the data to the optimizer at several levels of detail and applying a low-pass filter before sub-sampling is to be closely evaluated. Future development will be also aimed at further speeding up the inner rendering loop. This may be realized by implementing acceleration techniques such as empty space skipping or by pre-computing the volume gradient.

Ongoing work in the context of automatic segmentation of medical instruments is to be completed, potentially extended by soft-tissue recognition and masking, and integrated into the application. Additionally, the effect of defining a threshold for the CT intensities in order to generally exclude non-rigid structures, whose influence on the similarity measure is to be evaluated beforehand, should be investigated.

List of Figures

3.1 A C-arm X-ray camera 14

3.2 A CT scanner. 15

3.3 The schematic setting of X-ray image acquisition that is emulated in order to generate the DRRs. 24

3.4 A comparison of the DRRs (shown in the upper row) and the X-ray images (shown in the middle row) produced by the setting in Figure 3.3. The last row depicts the addition of the DRRs (red channel) and the radiographs (green channel). Yellow hints at a good match between them. 25

3.5 The left image shows a radiograph with the spheric calibration markers clearly visible. The right image shows the result of the marker segmentation. 27

3.6 The left image shows a radiograph containing different kinds of artifacts such as an ultrasonic probe, a surgical needle, calibration targets, and the camera's aperture. The right image shows the radiograph after the application of w. 34

4.1 The NVidia® GeForce® 8800 GTX graphics board. 43

4.2 Memory performance benchmark of the GeForce® 8800 GTX. The left bar represents the cache bandwidth, the middle bar the sequential off-chip memory access bandwidth, and the right bar the random off-chip memory access bandwidth. . . . 45

4.3 Off-board communication benchmark of the GeForce® 8800 GTX. The left chart subsumes the results for data transfers from main to GPU memory (*download*), where the right chart shows the bandwidths for transferring data from GPU memory back to main memory (*readback*). 48

4.4 An example of *texture reduction* used to sum up the intensities of an image. The viewport written to is scaled down to a forth of its current size at every step. 52

5.1 A sample X-ray image $w(u, v)J(u, v)$ and a masked DRR $w(u, v)I(u, v)$ 68

5.2 The preliminary translation and rotation gradient images ad_$\mathbf{t}(u, v)$ and ad_$\mathbf{r}(u, v)$ as returned by the first rendering pass of the restructured algorithm. They encode the potential change of the DRR in Figure 5.1(b) when $\mathbf{t}$ or $\mathbf{r}$ vary. 69

5.3 The correlation image $\mathrm{corr}(u, v)$, which provides the basis for computing the NCC measure, and an image depicting the X-ray differences from mean $w(u, v)(J(u, v) - \mu_{wJ})$. 70

5.4 The final translation and rotation gradient images ad_$\mathbf{t}(u, v)$ and ad_$\mathbf{r}(u, v)$ computed from the preliminary gradient images in Figure 5.2(a) and 5.2(b), and ad_$I(u, v)$. The R, G, and B channel correspond to the x, y, and z-coordinate respectively. 71

6.1 mTRE scatter plot of the registration experiments carried out with the pelvic bone data using algorithmic differentiation. (all values given in millimeters). 77

6.2 Detail of Figure 6.1 showing only a limited range of initial displacements (in millimeters). 77

6.3 The capture range plot for the pelvic bone data and algorithmic differentiation depicting the percentage of registrations within the initial mTRE range $[0, x]$ which was successful . . . 78

6.4 The mTRE scatter plot for the experiments conducted with the pelvic bone data using numerical approximation of the gradient. 78

6.5 Detail of Figure 6.4 showing only a limited range of initial displacements (in millimeters). 79

6.6 The capture range plot for the pelvic bone data and numerical approximation of the gradient. 79

6.7 The mTRE scatter plot for the experiments conducted with the human abdomen data using algorithmic differentiation. . . 80

6.8 Detail of Figure 6.7 showing only a limited range of initial displacements (in millimeters). 80

6.9 The capture range plot for the human abdomen data and algorithmic differentiation. 81

6.10 The mTRE scatter plot for the experiments conducted with the human abdomen data using numerical approximation of the gradient. 81

6.11 Detail of Figure 6.10 showing only a limited range of initial displacements (in millimeters). 82

6.12 The capture range plot for the human abdomen data and numerical approximation of the gradient. 82

List of Tables

6.1 Convergence and accuracy results 76
6.2 Performance results 83

Bibliography

[AMD, 2007] AMD (2007). AMD^{TM} Firestream^{TM}. `http://ati.amd.com/products/streamprocessor/specs.html`.

[Bauer et al., 2002] Bauer, C., Frink, A., and Kreckel, R. (2002). Introduction to the GiNaC Framework for Symbolic Computation within the C++ Programming Language. *Journal of Symbolic Computation*, 33:1–12.

[Birkfellner et al., 2005] Birkfellner, W., Seemann, R., Figl, M., Hummel, J., Ede, C., Homolka, P., Yang, X., Niederer, P., and Bergmann, H. (2005). Fast DRR generation for 2D/3D registration. In *Prodeedings of the MICCAI*, pages 960–967.

[Bischof and Bücker, 2000] Bischof, C. H. and Bücker, H. M. (2000). Computing Derivatives of Computer Programs. In *Modern Methods and Algorithms of Quantum Chemistry: Proceedings, Second Edition*, volume 3 of *NIC Series*, pages 315–327. NIC-Directors, Jülich.

[Bischof et al., 1997] Bischof, C. H., Roh, L., and Mauer, A. (1997). ADIC: An Extensible Automatic Differentiation Tool for ANSI-C. *Software-Practice and Experience*, 27(12):1427–1456.

[Blinn, 1982] Blinn, J. F. (1982). Light reflection functions for simulation of clouds and dusty surfaces. In *SIGGRAPH '82: Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, pages 21–29.

[Bouguet, 2007] Bouguet, J.-Y. (2007). Camera Calibration Toolbox for Matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`.

[Brent, 1973] Brent, R. P. (1973). *Algorithms for Minimization without Derivatives*. Prentice-Hall.

[Chan and Shen, 2005] Chan, T. and Shen, J. (2005). *Image Processing and Analysis – Variational, PDE, Wavelet, and Stochastic Methods.* SIAM, Philadelphia.

[Chiappetta, 2007] Chiappetta, M. (2007). Intel Core 2 Extreme QX6800. `http://www.computerbase.de/artikel/hardware/grafikkarten/2006/test_nvidia_geforce_8800_gtx/2/`.

[Chisu, 2005] Chisu, R. (2005). Techniques for Accelerating Intensity-based Rigid Image Registration. Master's thesis, Technische Universität München.

[ComputerBase, 2007] ComputerBase (2007). Test: nVidia GeForce 8800 GTX. `http://www.computerbase.de/artikel/hardware/grafikkarten/2006/test_nvidia_geforce_8800_gtx/2/`.

[Freeman, 2007] Freeman, V. (2007). Intel Core 2 Extreme QX6800 Processor Review. `http://www.sharkyextreme.com/hardware/cpu/article.php/3261_3670586__4`.

[Freund et al., 2004] Freund, E., Heinze, F., and Rossmann, J. (2004). Direction dependent projection fields for the fast DRR generation for medical 2D/3D registration. In *Proceedings of the IEMBS*, pages 1751–1754.

[Giering and Kaminski, 2002] Giering, R. and Kaminski, T. (2002). Recomputations in reverse mode AD. In *Automatic differentiation of algorithms: from simulation to optimization*, pages 283–291. Springer-Verlag New York, Inc., New York, NY, USA.

[Göddeke et al., 2005] Göddeke, D., Strzodka, R., and Turek, S. (2005). Accelerating Double Precision FEM Simulations with GPUs. In *18th Symposium Simulationstechnique*, volume Frontiers in Simulation, pages 139–144.

[Gong et al., 2006] Gong, R. H., Abolmaesumi, P., and Stewart, J. (2006). A Robust Technique for 2D-3D Registration. In *Engineering in Medicine and Biology Society*, pages 1433–1436.

[Griewank, 1989] Griewank, A. (1989). On Automatic Differentiation. In *Mathematical Programming: Recent Developments and Applications*, pages 83–108. Kluwer Academic Publishers.

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

[Grimm et al., 1996] Grimm, J., Pottier, L., and Rostaing-Schmidt, N. (1996). Optimal Time and Minimum Space-Time Product for Reversing a Certain Class of Programs. In *Computational Differentiation: Techniques, Applications, and Tools*, pages 95–106. SIAM, Philadelphia, PA.

[Harris, 2005] Harris, M. (2005). Mapping computational concepts to GPUs. In *ACM SIGGRAPH 2005 Courses*, pages 50–72.

[Houston, 2007] Houston, M. (2007). GPUBench Report. `http://graphics.stanford.edu/projects/gpubench/results/8800GTX-0003/`.

[Ino et al., 2006] Ino, F., Gomita, J., Kawasaki, Y., and Hagihara, K. (2006). A GPGPU Approach for Accelerating 2-D/3-D Rigid Registration of Medical Images. In *Lecture Notes in Computer Science*, volume 4330, pages 939–950. Springer.

[Jackson and Thomas, 2004] Jackson, S. and Thomas, R. (2004). *Cross-Sectional Imaging Made Easy.* Churchill-Livingstone.

[Kajiya, 1986] Kajiya, J. T. (1986). The rendering equation. In *Proceedings of the SIGGRAPH*, pages 143–150.

[Kajiya and Herzen, 1984] Kajiya, J. T. and Herzen, B. P. V. (1984). Ray tracing volume densities. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, pages 165–174.

[Khamene et al., 2006a] Khamene, A., Bloch, P., Wein, W., Svatos, M., and Sauer, F. (2006a). Automatic Registration of Portal Images and Volumetric CT for Patient Positioning in Radiation Therapy. *Medical Image Analysis*, pages 96–112.

[Khamene et al., 2006b] Khamene, A., Chisu, R., Wein, W., Navab, N., and Sauer, F. (2006b). A Novel Projection Based Approach for Medical Image Registration. In *WBIR*, pages 247–256.

[Kilgariff and Fernando, 2005] Kilgariff, E. and Fernando, R. (2005). The GeForce 6 series GPU architecture. In *ACM SIGGRAPH 2005 Courses*, pages 29–49.

[Kim et al., 2007] Kim, J., Li, S., Pradhan, D., Hammoud, R., Chen, Q., Yin, F., Zhao, Y., Kim, J., and Movsas, B. (2007). Comparison of Similarity Measures for Rigid-body CT/Dual X-ray Image Registrations. *Technology in Cancer Research and Treatment*, 6(4):337–346.

[Krüger and Westermann, 2003] Krüger, J. and Westermann, R. (2003). Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings of IEEE Visualization 2003*, pages 287–292.

[Kubias et al., 2007a] Kubias, A., Deinzer, F., Feldmann, T., and Paulus, D. (2007a). Extended Global Optimization Strategy for Rigid 2D/3D Image Registration. In *Computer Analysis of Images and Patterns*, pages 759–767.

[Kubias et al., 2007b] Kubias, A., Deinzer, F., Feldmann, T., Paulus, S., Paulus, D., Schreiber, B., and Brunner, T. (2007b). 2D/3D Image Registration on the GPU. In *Proceedings of the OGRW*.

[Köhn et al., 2006] Köhn, A., Drexl, J., Ritter, F., König, M., and Peitgen, H. O. (2006). GPU Accelerated Image Registration in Two and Three Dimensions. In *Bildverarbeitung für die Medizin*, pages 261–265. Springer.

[LaRose, 2001] LaRose, D. (2001). *Iterative X-ray/CT Registration Using Accelerated Volume Rendering*. PhD thesis, Carnegie Mellon University.

[Levenberg, 1944] Levenberg, K. (1944). A Method for Solution of Certain Nonlinear Problems in Least Squares. *Quantitative Applied Mathematics*, 2.

[Levoy, 1988] Levoy, M. (1988). Display of Surfaces From Volume Data. *IEEE Computer Graphics and Applications*, 8(3):29–37.

[Livyatan et al., 2003] Livyatan, H., Yaniv, Z., and Joskowicz, L. (2003). Gradient-Based 2D/3D Rigid Registration of Fluoroscopic X-ray to CT. *IEEE Transactions on Medical Imaging*, 22(11):1395–1406.

[Maes et al., 1997] Maes, F., Collignon, A., Vandermeulen, D., Marchal, G., and Suetens, P. (1997). Multi-Modality Image Registration by Maximization of Mutual Information. *IEEE Transactions on Medical Imaging*, 16:187–198.

[Maes et al., 1999] Maes, F., Vandermeulen, D., and Suetens, P. (1999). Comparative Evaluation of Multiresolution Optimization Strategies for Multimodality Image Registration by Maximization of Mutual Information. *Medical Image Analysis*, 3(4):373–386.

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

[Marquardt, 1963] Marquardt, D. W. (1963). An algorithm for Least-Squares Estimation of non-linear parameters. *SIAM Journal on Applied Mathematics*, 11.

[Moré and Thuente, 1994] Moré, J. J. and Thuente, D. J. (1994). Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software*, 20(3):286–307.

[Nealder and Mead, 1965] Nealder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 7.

[NVidia, 2007a] NVidia (2007a). Cg. `http://developer.nvidia.com/page/cg_main.html`.

[NVidia, 2007b] NVidia (2007b). NVIDIA CUDA Programming Guide, Version 1.0. `http://developer.nvidia.com/object/cuda.html`.

[OpenGL, 2007] OpenGL (2007). OpenGL - The Industry Standard for High Performance Graphics. `http://www.opengl.org/`.

[Owens, 2005] Owens, J. (2005). Streaming architectures and technology trends. In *ACM SIGGRAPH 2005 Courses*, pages 9–28.

[Penney et al., 1998] Penney, G. P., Weese, J., Little, J. A., Desmedt, P., Hill, D. L. G., and Hawkes, D. J. (1998). A Comparison of Similarity Measures for Use in 2D-3D Medical Image Registration. *IEEE Transactions on Medical Imaging*, 17(4):586–595.

[Phong, 1973] Phong, B. T. (1973). *Illumination for computer-generated images*. PhD thesis, University of Utah.

[Pock et al., 2006] Pock, T., Pock, M., and Bischof, H. (2006). Algorithmic Differentiation: Application to Variational Problems in Computer Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Powell, 1964] Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162.

[Press et al., 1992] Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (1992). *Numerical Recipes in C*. Cambridge University Press, 2nd edition.

[Roche et al., 1999] Roche, A., Malandain, G., Ayache, N., and Prima, S. (1999). Towards a Better Comprehension of Similarity Measures Used in Medical Image Registration. In *Proceedings of the MICCAI*, pages 555–566.

[Roche et al., 1998] Roche, A., Malandain, G., Pennec, X., and Ayache, N. (1998). The Correlation Ratio as a New Similarity Measure for Multimodal Image Registration. In *Proceedings of the MICCAI*, pages 1115–1124.

[Rohlfing et al., 2002] Rohlfing, T., Russakoff, D. B., Murphy, M. J., and Calvin R. Maurer, j. (2002). Intensity-based registration algorithm for probabilistic images and its application for 2D to 3D image registration. In *Proceedings of SPIE: Medical Imaging 2002: Image Processing*, volume 4684, pages 581–591.

[Russakoff et al., 2003] Russakoff, D. A., Rohlfing, T., Maurer, C. R., and Jr. (2003). Fast Intensity-based 2D-3D Image Registration of Clinical Data Using Light Fields. In *IEEE International Conference on Computer Vision*, pages 416–422.

[Schmid and Töpelt, 2007] Schmid, P. and Töpelt, B. (2007). Tom's Hardware's 2007 CPU Charts. `http://www.tomshardware.com/2007/07/16/cpu_charts_2007/page36.html`.

[Seibert, 2004] Seibert, J. A. (2004). X-Ray Imaging Physics for Nuclear Medicine Technologists. Part 1: Basic Principles of X-Ray Production. *Journal of Nuclear Medicine Technology*, 32(3):139–147.

[Siemens, 2007] Siemens (2007). ARCADIS Orbic 3D. `http://www.medical.siemens.com/`.

[Tomaževič et al., 2006] Tomaževič, D., Likar, B., and Pernuš, F. (2006). 3-D/2-D Registration by Integrating 2-D Information in 3-D. *IEEE Transactions on Medical Imaging*, 25(1):17–27.

[Tomaževič et al., 2003] Tomaževič, D., Likar, B., Slivnik, T., and Pernuš, F. (2003). 3-D/2-D Registration of CT and MR to X-ray Images. *IEEE Transactions on Medical Imaging*, 22(11):1407–1416.

[Tsai, 1992] Tsai, R. Y. (1992). A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-shelf TV Cameras and Lenses. In *Radiometry*, pages 221–244. Jones and Bartlett Publishers, Inc.

[Van de Kraats et al., 2004] Van de Kraats, E., Penney, G., Tomazevic, D., Van Walsum, T., and Niessen, W. (2004). Standardized Evaluation of 2D-3D Registration. In *Proceedings of the MICCAI*, pages 574–581.

[Weese et al., 1997] Weese, J., Buzug, T. M., Lorenz, C., and Fassnacht, C. (1997). An approach to 2D/3D registration of a vertebra in 2D X-ray fluoroscopies with 3D CT images. In *Proceedings of the CVRMed-MRCAS*, pages 119–128.

[Weese et al., 1999] Weese, J., Göcke, R., Penney, G. P., Desmedt, P., Buzug, T., and Schumann, H. (1999). A fast voxel-based 2D/3D registration algorithm using a volume rendering method based on the shear warp factorization. In *Proceedings of the SPIE: Medical Imaging 1999: Image Processing*, volume 3661, pages 802–810.

[Wein, 2003] Wein, W. (2003). Intensity Based Rigid 2D-3D Registration Algorithms for Radiation Therapy. Master's thesis, Technische Universität München.

[Wein et al., 2005] Wein, W., Röper, B., and Navab, N. (2005). 2D/3D Registration Based on Volume Gradients. In *Proceedings of SPIE: Medical Imaging 2005: Image Processing*, volume 5747.

[Westover, 1991] Westover, L. A. (1991). *Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm*. PhD thesis, UNC Chapel Hill.

[Zheng, 2007] Zheng, G. (2007). Unifying Energy Minimization and Mutual Information Maximization for Robust 2D/3D Registration of X-Ray and CT Images. In *LNCS: Pattern Recognition*, volume 4713, pages 547–557. Springer.

[Zhu et al., 1997] Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. (1997). L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560.

[Zöllei, 2001] Zöllei, L. (2001). 2D-3D Rigid-Body Registration of X-Ray Fluoroscopy and CT Images. Master's thesis, MIT AI Lab.