Fast Registration of Medical Imaging Data Using Optimised Radial Basis Functions

Roger Steven Rowland

June 2007

A Thesis submitted to the School of Computing Sciences

University of East Anglia

in Partial Fulfillment of the Requirement for

the Degree of Doctor of Philosophy in Computer Science

Abstract

The ability to register medical images to align anatomical features accurately has great benefits for clinical diagnosis, patient monitoring, education and surgical planning. Since the 1990s, much research has gone into the development of accurate methods for the non-rigid registration of medical image data in two and three dimensions. It is generally true that more accurate registration requires more computation and therefore more time.

With the growing development of computer assisted surgery applications such as image guided surgery and surgical navigation, there is an increasing need for fast and accurate non-rigid image registration, suitable for use intra-operatively.

Within this thesis, we describe the techniques that underlie non-rigid registration, identify the time consuming tasks and develop methods to accelerate the process while retaining acceptable registration accuracy. Specifically, we provide both algorithmic and hardware-based acceleration of radial basis function evaluation – a common feature of many non-rigid registration techniques.

Using algorithmic acceleration with 256^3 voxel synthetic data sets and 729 landmark pairs, we have achieved $> 90\%$ warping accuracy in 13 seconds (maximum error 2.8mm) and $> 98.8\%$ accuracy in 46 seconds (maximum error 1.6mm). Using a hardware implementation of the same technique gives $> 80\%$ accuracy in 0.81 seconds (maximum error 1.4mm). Using real CT and MRI data sets, each with 256^3 voxels, we manually placed 53 landmarks and achieved a visually acceptable non-rigid registration in 0.6 seconds.

Acknowledgements

First, I would like to acknowledge the support, advice, encouragement and knowledge supplied by my supervisors – Dr. Rudy Lapeer and Dr. Mark Fisher. In particular, Rudy's knowledge and experience of augmented reality, medical imaging and computer assisted surgery has been essential. Rudy has also been responsible for nagging, cajoling, proof-reading, criticising and general idea-bouncing – an enormously rewarding experience and an extremely likeable character, a good friend. Thanks also go to all the PhD students in the lab – Min Si Chen, Paul Gasson and Gerardo Gonzalez-Garcia with whom I have exchanged advice and jokes in equal measure. Last but never least are my friends and family, who have toiled along with me during the course of this research, at times suffering more than myself. To my ex-wife and good friend, Mary, our children, Georgia and William – who have always loved me and supported me and given me the determination not to give up. To my partner, confidante, soul-mate and inspiration, Su Yu, I can only say with all honesty that, without you, none of this would have been completed – you *are* my strength. To my loving, wise, long-suffering and disturbingly healthy and happy parents, Jack and Nita – who have always supported me, always encouraged me to succeed, but have never pushed me – I owe you a lot, I really do.

This thesis is dedicated to Georgia and William.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

List of abbreviations used in this thesis :

2D	–	Two dimensional
3D	–	Three dimensional
AR	–	Augmented reality
ASM	–	Active shape model
CAS	–	Computer aided surgery
CC	–	Correlation coefficient
CM	–	Cardinality match
CPU	–	Central processing unit
CT	–	Computed tomography
DM	–	Digital mammography
DR	–	Digital radiography
EBS	–	Elastic body spline
FAIR	–	Fast automatic image registration
FAIR-II	–	Fast automatic image registration 2
FEM	–	Finite element method
FFD	–	Free form deformation
FGT	–	Fast Gauss transform
FLE	–	Fiducial localization error
FMM	–	Fast multipole method
fMRI	–	Functional magnetic resonance imaging

FPGA – Field-programmable gate array

FRE – Fiducial registration error

GEBS – Gaussian elastic body spline

GPU – Graphics processing unit

ICP – Iterative closest point

IGS – Image guided surgery

LAN – Local area network

MFC – Microsoft foundation classes

MI – Mutual information

MMI – Mattes mutual information

MQ – Multi-quadric

MRI – Magnetic resonance imaging

MRSD – Mean reciprocal square differences

MS – Mean squares

NMI – Normalised mutual information

NNI – Nearest neighbour interpolation

PC – Personal computer

PDE – Partial differential equation

PDF – Probability density function

PET – Positron emission tomography

PVI – Partial volume interpolation

RAM – Random access memory

RBF – Radial basis function

RMS – Root mean square

SAM – Statistical appearance model

SGI – Silicon Graphics Incorporated

SM3 – Shader model 3.0

SN – Surgical navigation

SPECT – Single photon emission computed tomography

TPS – Thin-plate spline

TRE – Target registration error

TRI – Tri-linear interpolation

US – Ultrasound

VHP – Visible human project

Chapter 1

Introduction

1.1 Medical imaging

Since the initial development of techniques to produce digital images of the human body in the latter half of the last century, there has been an enormous amount of scientific research and development in this area.

The ability to visualise structures and tissues inside the body is now an essential and commonplace diagnostic tool. A variety of techniques are employed to produce images in two or three dimensions and in some cases these are viewed over time to show, for example, a 3D image of a beating heart.

To produce an image it is necessary to expose the patient to some form of energy, which interacts in some way with the tissues of interest. Different types of energy interact with different types of tissue in different ways and with some techniques there is a risk of damage to the patient as an effect of the imaging process. The different techniques are known as *image modalities* and are characterised by their ability to visualise particular tissues, their sensitivity, image resolution and effect on the patient.

All of these factors are taken into account when deciding the most appropriate method to use for a particular patient or condition. According to Bushberg

et al. [20], examples of the more commonly used modalities for digital imaging are as follows :

Digital radiography (DR) uses an x-ray source on one side of the patient and a (typically flat) electronic x-ray detector on the other side. A short pulse of x-rays, less than 0.5 seconds, passes through the patient towards the detector. The x-rays that enter the patient's body are modified by the extent to which they are scattered and absorbed (attenuated) by individual tissues. As the attenuation properties of skin, bone, air, etc. within the patient are different, the distribution of x-rays that hits the detector forms a radiographic image which can be used for diagnostic purposes. Radiographic images are used for a wide range of pathologies, including broken bones, lung cancer and cardiovascular disorders.

Digital mammography (DM) is digital radiography of the breast. In principle this is the same as digital radiography except that much lower x-ray energies are used than in any other radiographic technique. For this reason, digital mammography uses x-ray machines and detectors that have been specifically designed for the purpose. Mammography is used for breast cancer screening as well as for diagnostic purposes.

Computed tomography (CT) was the first computerised medical imaging modality and became clinically available in the 1970s. CT passes x-rays through the patient at many different angles by rotating the x-ray source around the body. One or more linear detector arrays collect the x-rays after they have passed through the body and a computer synthesises the results into a *tomographic* image – a slice through the body. One advantage over digital radiography is that each slice can be produced without the presence of any overlying or underlying structures. Multiple CT slices can be reconstructed by computer to form a 3D image of the patient, which

allows it to be used for a wide range of diagnostic purposes, including tumour identification, ruptured discs, subdural haematomas, aneurysms and other pathologies.

Single photon emission computed tomography (SPECT) is a tomographic technique which is based on nuclear medicine imaging. In this technique a substance containing a radioactive isotope is administered to the patient either orally, by injection or by inhalation. This radioactive substance will distribute itself in the patient according to some physiological status – for example it may be designed to concentrate in cancerous tissues. To produce the tomographic images, a nuclear camera records x-ray or gamma-ray emissions from the patient from a series of different angles and, in a similar way to CT, this information is synthesised into tomographic slices. SPECT images provide information on *functional* aspects of the patient's physiology rather than purely structural.

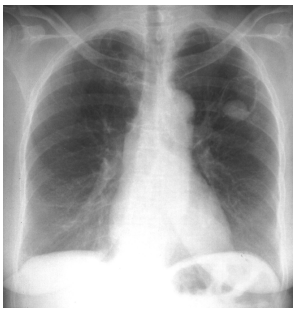
Positron emission tomography (PET) is similar to SPECT in that the images are produced by measuring emissions from the patient's body. In PET, it is positrons (positively charged electrons) which are used and these are emitted by some radioactive isotopes such as fluorine 18 and oxygen 15. These radioisotopes are incorporated into a compound which will localise in the body in a similar way to SPECT. The decay of the isotope produces a positron which immediately combines with an electron, producing energy known as *annihilation radiation*. This is similar to gamma-ray emission except that *two* photons are emitted in almost exactly opposite directions. A ring of detectors around the patient detects the photon pairs and records the line through the patient along which the decay event must have occurred. By combining many such detections, it is possible to compute the 3D distribution of the radioisotope and produce a series of tomographic emission images. Like SPECT, PET is used

for functional imaging of the brain and for imaging primary tumours and their metastases.

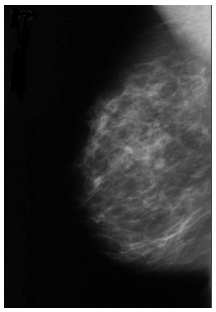
Magnetic resonance imaging (MRI) is a tomographic technique which uses a strong magnetic field and takes advantage of the nuclear magnetic resonance properties of the proton [115]. The proton is the nucleus of a hydrogen atom, which is prevalent in biological tissues. The patient is placed in the magnetic field and a pulse of radio waves is generated by coils around the patient. The protons in the patient's tissues absorb the radio wave energy and re-emit it after a time that depends on the magnetic properties of the surrounding tissue. The emitted radio waves are detected by the coils surrounding the patient. By changing the magnetic field strength as a function of spatial location – using a magnetic field gradient – the proton resonant frequency will vary as a function of position. The MRI scanner uses the frequency and phase of the returning radio waves to determine the position of each signal from the patient and so build up a set of tomographic images through the body. MRI can provide exceptional quality of images for tissues containing different amounts of fat or water. Brain and spine imaging are common applications of MRI.

Ultrasound (US) uses mechanical energy, in the form of high frequency sound waves, to produce images of internal structures in the body. A short pulse of sound is generated by a transducer in contact with the patient's body and the sound waves travel into the tissues and are reflected by the internal structures, causing echoes. The reflected echoes are detected by the transducer, which uses them to build a profile of the structures along that path. The sound beam is swept over a range of angles and the reflection from each scan line is recorded and used to generate the ultrasound image. US is used widely during pregnancy to monitor the growing foetus and is also used to detect structural abnormalities of the

kidneys, liver, pancreas and spleen.

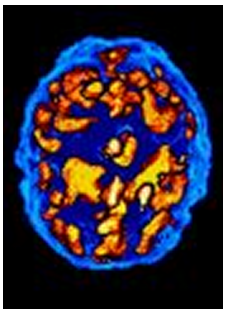


(a) DR.

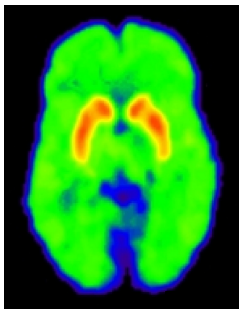


(b) DM.

(c) CT.



(d) SPECT.



(e) PET.

(f) MRI.

(g) US.

Figure 1.1: Example images from different medical imaging modalities.

Figure 1.1 shows example images from each of the above modalities[1]. Figure 1.1a shows a digital chest x-ray, Figure 1.1b shows a digital mammogram of a normal breast, Figure 1.1c shows an abdominal CT slice, Figures 1.1d, 1.1e and 1.1f show SPECT, PET and MRI images of the brain respectively, and Figure 1.1g shows an ultrasound scan of a human foetus.

[1]Unless otherwise stated, images are courtesy of Wikipedia Commons.

In addition to the differences in tissue type that these imaging modalities target, they also have different ranges of spatial resolution, which limit the size of feature which can be resolved clearly. Table 1.1, from Bushberg et al. [20], shows typical values for spatial resolution in millimetres for each of the above modalities.

Modality	**Abbreviation**	**Spatial resolution (mm)**
Digital radiography	DR	0.17
Digital mammography	DM	0.05 – 0.10
Computed tomography	CT	0.4
Single photon emission tomography	SPECT	7.0
Positron emission tomography	PET	5.0
Magnetic resonance imaging	MRI	1.0
Ultrasound (5MHz)	US	0.3

Table 1.1: Typical spatial resolution of medical imaging modalities.

1.2 Medical image registration

Because different imaging modalities can provide different but complementary information about the condition of an individual patient, it has become increasingly desirable to be able to 'fuse' the images from two or more modalities in order to provide a better visualisation for diagnostic use.

It is common to combine CT and MRI data to provide a composite view of bony structures and soft tissues, each of which is obtained from only one modality [63, 69, 83, 108]. Also, combining functional data from PET or SPECT with structural data from CT or MRI can provide diagnostic benefits [75, 117, 157, 163].

However, combining data from different imaging modalities is not straight-

forward. Problems with patient position, organ deformation or motion, differing spatial resolutions and even non-linear distortions introduced by the imaging modality itself (e.g. MRI) make this a complex task. The process of fusing image data from more than one modality so that there is a sufficiently good spatial alignment of anatomical structures is called *image registration* and this is described in more detail in Chapter 2.

1.3 Applications of medical image registration

Apart from diagnostic use, medical image registration is also used in computer aided surgery (CAS) applications. These include surgical navigation (SN), image guided surgery (IGS), augmented reality (AR) [160] and surgical simulation.

For many surgical procedures – for example endoscopic sinus surgery [24, 25], brachytherapy of cranial tumours [8, 9], pelvic surgery [70], cryotherapy of the liver and neurosurgey [182] – the task of the surgeon can be greatly assisted by providing an integrated view of the patient's body during the operation enhanced with medical image data from pre-operative scans of the same patient.

In this way, it can be possible to provide the surgeon with visual information of the underlying and surrounding structures at the actual operation site. Where it is important to avoid nerve damage – for example in maxillofacial surgery – or to navigate efficiently to the area of interest – for example in endoscopic sinus surgery – it is obviously of great benefit to have the ability to overlay the patient data from e.g. an MRI scan onto the surgeon's view through a stereo microscope or endoscope.

For this type of application, speed of image registration is as important as accuracy.

1.4 Thesis objectives

The main focus of the research presented in this thesis is to develop methods to accelerate the process of non-rigid medical image registration in three dimensions so that the run time performance and registration accuracy are suitable for use in intraoperative CAS applications.

Section 2.6.1 in Chapter 2 gives a concise statement of intent.

1.5 List of contributions

This thesis makes the following contributions :

1. Extends the "Fast RBF" evaluation formulation introduced by Livne and Wright [101] to three dimensions and applies it to mono-modal synthetic data and multi-modal medical image data.

2. Develops a software implementation of the "Fast RBF" algorithm and demonstrates its speed and accuracy using synthetic data and real medical image data.

3. Develops a hardware implementation of the "Fast RBF" algorithm and demonstrates its speed and accuracy using synthetic data and real medical image data.

4. Demonstrates the ability to balance speed and accuracy in the "Fast RBF" technique by adjusting a single run time parameter.

5. Demonstrates the execution time behaviour of the "Fast RBF" technique when supplied with an increasing number of inputs.

6. Demonstrates the importance of appropriate design of synthetic test data when assessing the accuracy of optimised warp evaluation techniques.

1.6 Organisation of the thesis

This chapter has presented an overview of the work and some background information about the application areas, which set the context for forming specific objectives. The remainder of this thesis addresses these objectives in the following manner:

Chapter 2 presents a review of previous work and background information, which covers registration of multi-modal medical images and, in particular, non-rigid registration techniques. The chapter concludes with a concise statement of the objectives of the thesis.

Chapter 3 gives more specific details of the application of radial basis functions and particularly the thin plate spline, in the context of non-rigid registration, and describes the methods developed to accelerate the application of these functions using both hardware and software optimisations.

Chapter 4 presents the experimental results of the non-rigid registration techniques developed in Chapter 3, using both synthetic test data and real medical imaging data sets.

Chapter 5 discusses a number of issues raised by the experimental results, offering explanations for some and identifying others as subjects for potential future work.

Chapter 6 provides a concluding summary of the work, lists the contributions of the thesis and presents ideas and suggestions for future research in this area.

Chapter 2

Image registration – Background

2.1 General

Image registration is a requirement that has existed for almost as long as we have had the technology necessary to produce digital images. The ability to align two or more images, from different viewpoints, taken at different times, or with different image sensors has various applications [196]. For example :

- Environmental monitoring;
- Satellite analysis of land usage;
- Weather forecasting;
- Security monitoring;
- Target recognition;
- Medical applications.

The last item in this list, medical applications, is the primary area of interest for this thesis. As we suggested in the introductory chapter, there are a number

of uses for image registration in a medical context. We might wish to register data from the same patient (i.e. *intrasubject*) but obtained from different imaging modalities (e.g. CT, MRI and ultrasound for structural analysis, or PET, SPECT and fMRI for functional analysis). This is known as *multi-modal registration*. Alternatively, we may wish to register data from the same patient using the same imaging modality but at different times (e.g. to assess tumour growth or treatment progress). This is called *multi-temporal registration*. Finally, we may wish to register data from a single imaging modality but for a number of different patients (i.e. *intersubject*), for example to produce an *atlas* – a statistical map of a certain part of the body – which can subsequently be employed for image registration, image guided surgery, or to assist segmentation where image contrast is poor [38, 55, 145]. This is particularly applicable to brain imaging, where it is important to identify individual structures in the brain which may not be easily identified from the image data alone [180].

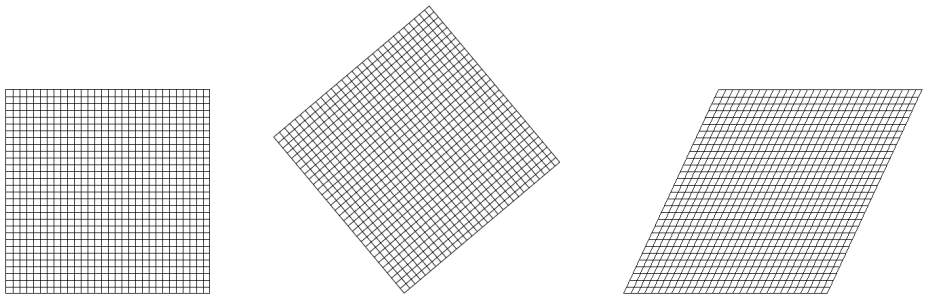
2.2 Rigid image registration

Early image registration techniques were focused on 2D images and *rigid*[1] transformations, later extending to *affine*[2] transformations – Figure 2.1 shows some simple 2D examples. Brown's 1992 survey [18] gives a comprehensive picture of the state of the art at that time.

However, in the medical context, rigid and affine transformations are not sufficient. Human bodies, although similar, cannot be modeled by combinations of linear transformations. Organs within the body will deform locally due to differences between patients, patient position in the imaging device, tumour growth or other physiological aspects. In fact, even breathing and heartbeat may have a significant effect. In addition, some imaging modalities, particularly

[1] Rigid implies just translation and/or rotation transformations.

[2] Affine means rigid plus scaling and/or shearing.

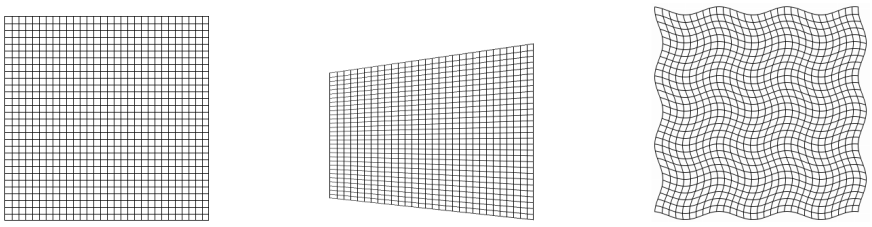


(a) Square grid. (b) Rigid. (c) Affine.

Figure 2.1: A square grid undergoing rigid and affine transformations.

MRI, introduce their own non-linear distortions [20], which have to be taken into account when performing multi-modal registration or even mono-modal registration between different makes or models of scanner. In 2004, Wang et al. [179] employed a 3D phantom to accurately measure the geometric distortion inherent in a number of clinical MRI systems. In this study, geometric errors ranged from 10 to 25 millimetres within a $240 \times 240 \times 240\text{mm}^3$ volume and could not be reduced along the axis normal to the image plane, making this a significant problem in 3D registration.

The limitations of rigid image registration were already apparent by 1997 when the survey by Maintz and Viergever [106] listed not only rigid and affine, but also *projective* and *curved* transformations among the registration techniques for medical imaging. While an affine transform will map parallel lines onto parallel lines, a projective transformation will map parallel lines onto lines that may *not* be parallel. A curved transformation will map lines onto curves and is sometimes called an *elastic* transformation. Figure 2.2 shows examples of these additional transformations.



(a) Square grid. (b) Projective. (c) Curved.

Figure 2.2: A square grid undergoing projective and curved transformations.

2.3 Non-rigid image registration

For the majority of medical imaging applications and particularly for multi-modal registration, a non-linear transformation – one that provides for some localised stretching of the image – is the accepted technique. In most published work, such techniques are termed *non-rigid* rather than non-linear, referring to the nature of the object(s) being registered rather than the techniques being used.

In 1996, Little et al. [99] approached non-rigid registration by first decomposing the problem into a number of rigid transformations. The technique was applied to spinal images, where rigid matching on individual vertebrae was followed by an approximation of the overall deformation by smooth interpolation. In 1998, Maintz et al. [105] adopted a similar approach by partitioning the image data into a number of 'windows' on which local, rigid registration is performed before forming a global transformation by combining the individual results. More recent work continues with similar piecewise or hierarchical registration methods, for example Pitiot et al. [125] and Likar and Pernus [98].

A number of publications show that non-rigid registration gives better results than rigid registration where deformable soft tissue is involved. For example, in 1998, Rueckert et al. [34, 146, 148] demonstrated the superiority of free-form deformations based on B-splines when compared to rigid and affine transformations applied to MRI breast images, and, in 2003, Fei et al. [41]

showed that non-rigid warping consistently outclassed rigid body registration for prostate and pelvic MRI data.

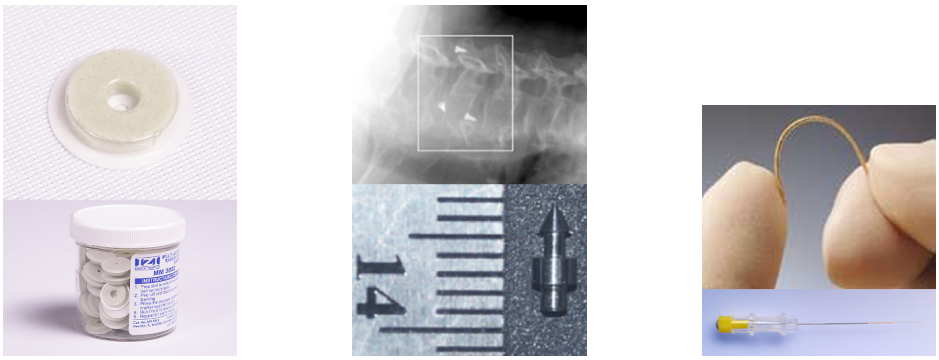
In addition to multi-modal image registration for visualisation purposes, non-rigid registration is also used for motion tracking [22, 23, 130, 143, 147] and, more recently, quantification of growth and motion [2, 144].

Typically, non-rigid registration of two medical images initially involves the identification (either manually or partially automated) of correspondences between the images, followed by a transformation which maps one image onto the other. The goal is to bring together the anatomically homologous locations of the two images. The methods used to register two medical images can be grouped into four main categories, which we will discuss in more detail individually. The categories are :

- Point based,
- Surface based,
- Intensity based, and
- Model based.

2.3.1 Point based registration methods

Point based methods involve the identification of a set of corresponding points in the images being registered. This process may be manual [95, 96] or semi-automated [12, 97, 151, 181], although it can be very difficult to automate landmark identification with accuracy from the medical image data alone, except in particular cases where the anatomy lends itself to conventional image processing or computer vision techniques. This problem can be mitigated by the use of specific markers, attached externally or internally to the patient's anatomy and which can be easily identified by the imaging modality [112, 150]. Maintz and



(a) Adhesive. (b) Percutaneous. (c) Internal.

Figure 2.3: A selection of fiducial markers used for medical image registration or image guided treatment.

Viergever [106] categorised these as *intrinsic* and *extrinsic* registration methods respectively.

As intrinsic registration methods rely only on data available from the images themselves, the selection of landmarks is sometimes a time-consuming manual process [63, 64, 66], although some researchers have attempted to automate this by performing some analysis of the image data to identify features that may be useful for matching, such as corners, edges or extreme curvature [12, 97, 107, 188]. Such techniques typically include an additional phase whereby landmark positions are optimised by maximising a similarity measure between the registered images in an iterative procedure [46, 47, 74, 139, 194]. Section 3.5 in Chapter 3 provides more detail about similarity measures used in this context.

Markers used for extrinsic registration may be attached to the patient internally or externally and are known as *fiducial markers*. Internal fiducial markers include small wires or coils, inserted either surgically or percutaneously prior to imaging. External markers include objects which may be stuck to the patient's skin, screwed into bone, or otherwise attached to a rigid part of the anatomy, for example a *stereotactic frame* or a dental moulding [9, 37]. Figure 2.3 shows a

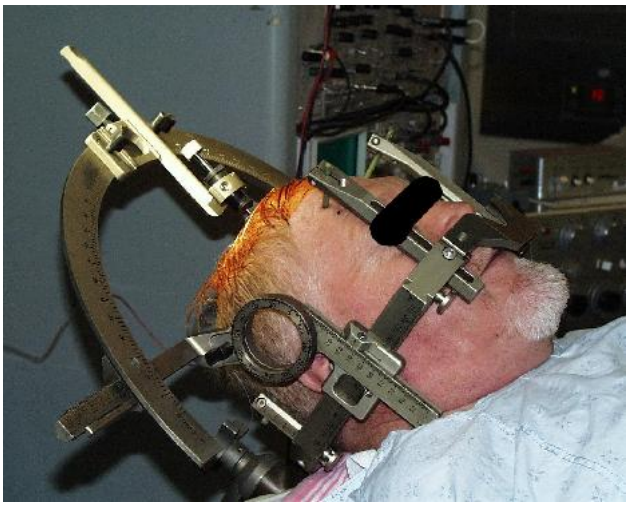


Figure 2.4: A stereotactic frame used for image guided neurosurgery.

number of examples of fiducial markers; Figure 2.3a shows an adhesive marker designed to be attached to a patient's skin, Figure 2.3b is a percutaneous marker used – in this instance – for image guided spinal surgery, and Figure 2.3c shows a VISICOIL™ linear fiducial soft tissue marker, which is designed to be used for image guided radiation therapy. Figure 2.4 shows a stereotactic frame being used for a neurosurgical procedure.

It is arguably much easier to automate the identification of fiducial markers in medical image data as they are specifically designed for that purpose. However, there are practical limits on their application for non-rigid registration because it is not feasible to insert them anywhere and everywhere. The use of fiducial markers places additional steps into the imaging procedure, which may be time consuming and uncomfortable for the patient and, with both internal and external fiducial markers, there is still potential for movement.

In general, point based registration methods have the advantage that it is usually faster to compute the mapping transformation than it is with surface based or intensity based registration because the number of landmark pairs is normally much smaller than the number of pixels or voxels in the image data.

However, this also means that the transformation is calculated without any reference to the intensity values in the underlying data with the result that poor placement of landmarks may leave areas with real local deformation unaffected. To overcome this problem, point based methods have been combined with surface or intensity based methods [175, 188], for example by dynamically creating or adjusting landmark positions based on local or global similarity measurements [12, 43, 122, 139], or by using automatically placed landmarks as a convenient starting point for surface based or intensity based registration [89, 151, 156].

2.3.2 Surface based registration methods

As the name implies, surface based registration requires the extraction of equivalent surfaces from two images in order for a mapping transformation to be derived. As Maintz and Viergever note [106], the surfaces to be registered must first be segmented from the raw image data. The implication of this is that the registration accuracy is dependent on the accuracy of the surface segmentation, which can be particularly difficult in some imaging modalities – for example, ultrasound [60, 158, 185].

Combining surface based registration with other techniques is not uncommon. Maurer et al. [113] combine surface and point based registration techniques and Betke et al. [12] supplement automated landmark detection in chest and lung images by using a surface based registration technique based on the *iterative closest point* (ICP) algorithm [11].

Generally, surface based registration techniques have been confined to neuroimaging and orthopaedic applications [106] and, while intermodal registration is certainly possible using this technique [5, 39, 73, 75, 108, 166], computer-aided surgery (CAS) and image-guided surgery (IGS) applications [52, 60, 62, 67, 185], as well as atlas-based brain mapping and segmentation [42, 50] form a large part

of the more recently published work. Audette et al. [5] provide a useful survey of some of the algorithmic techniques used for surface registration, particularly for medical applications.

In the field of computer-aided surgery, Hardy et al. [58] report surface registration for endoscopic sinus surgery as a superior technique in terms of time and accuracy when compared to landmark and fiducial registration methods. However, other studies have shown [75, 91, 184] that surface based registration techniques are still inferior to intensity based methods.

2.3.3 Intensity based registration methods

Intensity based – or voxel based – registration methods require less pre-processing than point based and surface based methods. As intensity based methods operate directly on the grey values in the images, they do not need prior identification of matching landmarks nor segmentation to extract surfaces.

In recent years, intensity based methods have become one of the most heavily researched non-rigid image registration techniques [196], the most prevalent of which use *mutual information* (MI) – first introduced by Viola and Wells [177, 183] – as a measurement of image similarity [3, 32, 79, 126, 128, 146, 158]. An overview of MI based registration methods is given by Maes et al. [103, 104], who originally worked independently in this area at around the same time as Viola and Wells.

While many intensity based registration methods use the whole image – ignoring any geometrical features of the objects of interest – there have been a number of approaches which combine the image similarity measure with a regularisation term to constrain the deformation. Examples are Maintz et al. [105], Rueckert et al. [149] and Rohlfing et al. [137]. In any case, all of these methods involve an iterative optimisation process in which the parameter space of the spatial transformation is 'explored' by the optimisation algorithm until the

similarity measure reaches a global maximum. Therefore, one important characteristic of the optimisation process is the ability of the algorithm to converge on the true global maximum rather than being trapped by one or more local maxima.

So, although not hindered by a manual or semi-automatic pre-processing step, in common with all iterative procedures, the speed of an intensity based registration technique is dictated by three attributes :

- The time necessary to calculate the similarity measure.
- The time required to apply a transformation to the source image.
- The number of iterations required before convergence.

2.3.4 Model based registration methods

Model based registration methods are those that attempt to produce a model of the deformation field based on the physical characteristics of the organ or tissues of interest. As it is usually an expensive computational process to perform the modeling exactly, there is often a need to compromise registration accuracy so that a simplified approximation of the physical system can be used, which is able to be calculated more quickly.

Bajcsy and Kovačič [7] first used an elastic model for matching CT data with an atlas of the brain. Subsequent work by Gee et al. [49] built further on this approach. Various other extensions of elastic modeling have been proposed [61, 80, 82, 111, 123, 189].

Another model based approach involves the finite element method (FEM). This is an engineering technique whereby the area of interest is subdivided into an interconnected mesh of elements, each of which has attributes consistent with the properties of the underlying physical feature or tissue. Deformations are then modeled by displacing the nodes in the mesh according to some outside

influence while constraining their motion based on an energy function which involves the individual properties of each node. Bro-Nielsen [14] used FEM for both image registration and surgical simulation [15–17]. While FEM applications are often concerned with surgical simulation (for example [86, 87]), they have been used, with varying degrees of success, for performing image registration [28, 42, 51] or validating other image registration techniques [155].

Other model based techniques include those based on fluid dynamics [26, 33], which use the Navier-Stokes equations, and optical flow techniques [30, 53, 76, 92, 164, 169]. Registration using fluid dynamics is slow when compared to other non-rigid registration techniques [187], which makes it unsuitable for intra-operative AR and other applications which require fast registration. Optical flow techniques build an array of displacement vectors to model the deformation field which requires identification of similar intensity values in the two images and is therefore more suitable for mono-modal registration.

In 2006, Petrovic et al. [124] demonstrated that it is possible to integrate segmentation, modeling and registration into a single framework by using a set of 32 MRI brain images to train a model to analyse the structure of an unknown brain image. This work is built on techniques based around Active Shape Models (ASM) and Statistical Appearance Models (SAM) and aims to provide a robust method for deriving a set of deformation fields defining the correspondences between the images, a classification of voxels into different tissue types and a statistical representation of the shape variability across a set of similar images.

2.3.5 Summary of non-rigid image registration methods

Point based methods require identification of homologous points on the images being registered. This is often a manual process and therefore slow and prone to error. Semi-automatic landmark placement is possible but also prone to er-

ror. However, once the landmarks are placed, calculation of the transformation function is typically faster than with the other methods because the number of landmarks is small in comparison to the number of voxels in the image.

Surface based methods require a segmentation step, which adds time and introduces another potential source of error. However, for certain well-defined structures – particularly in MRI brain images – this technique is more suitable than point based registration [55, 73, 108].

Intensity based techniques have become increasingly popular, partly because the need for time-consuming manual pre-processing steps is reduced or avoided completely. However, these techniques involve iteration, which consists of repeated transformations and similarity measurement, both of which can be lengthy.

Model based methods have the potential to achieve more accurate registrations for certain applications – for example, when modeling tissue deformation during surgery. However, the non-parametric transformations involved in many of the model based techniques are more computationally intensive than any of the other methods and, in general, more knowledge is required about the physical characteristics of the objects in the image data.

2.3.6 Transformation functions for non-rigid registration

Various transformation functions have been proposed and used for non-rigid medical image registration. In contrast to rigid or affine registration functions, which operate globally, non-rigid transformations allow local deformation and the deformation field can be calculated in two ways – using either parametric or non-parametric methods.

Among the parametric methods are those based on B-splines, which have been employed generally for curve and surface representation since the 1970s. B-splines have local support, which means that they are zero outside a sub-region.

Non-rigid registration techniques may either utilise B-splines directly [74, 82, 83, 142, 156, 168, 191] or may employ the Free-Form Deformation approach (FFD) in which the object to be registered is embedded within a B-spline object [68, 102, 109, 137, 146, 149, 192]. Lee et al. [90] proposed a multi-level approach for fitting a 2D cubic B-spline surface to scattered data, which overcame the normally problematic issues associated with conventional least-squares fitting by using a coarse-to-fine hierarchy of control lattices to generate a sequence of B-splines whose sum approaches the required interpolation. Later, Tustison et al. [172–174] generalised this technique to N-dimensions and allowed an arbitrary degree of B-spline.

Parametric methods with global support – registering the whole image – include techniques which use the thin-plate spline (TPS) [6, 27, 40, 69, 73, 81, 117, 139, 161, 163, 175, 181]. The TPS is an interpolating function, which maps corresponding control points exactly. It is a special type of radial basis function (RBF) which minimises the curvature of the displacement field resulting in a smooth deformation. Other RBFs have also been proposed which, unlike the TPS, have local support – for examples see Fornefett [45] and Rohde [133].

In 2006, Zagorchev and Goshtasby [193] produced a comparative study of transformation functions for non-rigid image registration, which included two RBFs in its comparison – the TPS and the multi-quadric (MQ) [56, 57]. The TPS and MQ transformations were favoured over piecewise linear and weighted mean transformations when there were relatively few point correspondences (less than one thousand) and the variation in spacing between the points was not large.

Non-parametric transformation methods calculate the deformation of each voxel directly and although more flexible than those using basis functions, are usually more computationally intensive. Examples include many of the model based registration techniques described at the end of the previous section. In

particular, methods using optical flow [30, 53, 76, 92, 164, 169] and fluid dynamics [26, 33].

In recent years, techniques using elastic body splines (EBS) have been proposed, particularly those based on Gaussian forces (GEBS) [80, 189]. These splines are the result of a physical model in the form of analytical solutions of the Navier equation and describe elastic deformations of physical objects. Kohlrausch et al. [80] report that the use of GEBS is particularly useful in medical image registration where the geometric differences between the images are caused by tissue deformation due to surgical intervention or pathological processes. This makes them suitable for intra-subject mono-modal registration applications.

2.4 Image registration in computer aided surgery

The term *computer aided surgery*, or CAS, covers a wide range of applications. These include :

Surgical planning This covers pre-operative image processing and visualisation, image registration and surgical simulation. Multi-modal images may be registered [67, 70] and visualised [24] to assist the surgeon in planning the operation [65, 195] and it may be possible to use the pre-operative image data to simulate the surgical procedure itself.

Surgical navigation This refers to the precise guidance of surgical tools and instruments [131] as well as the intraoperative imaging processes which support this – for example, endoscopy [25, 58], ultrasonography [185], interventional CT/MRI [167, 186] and surgical stereoscopic displays. Positional tracking of surgical tools is essential and image registration between preoperative and intraoperative images is complementary but important. Augmented reality (AR) [160] may be employed to provide an overlay

of CT or MRI data on the intraoperative endoscope or microscope view, or to give visual feedback of a 'virtual' surgical tool in an image of the patient's CT or MRI image data.

Treatment Although there is some overlap with surgical navigation, this area also includes applications such as surgical robotics [52], surgical laser treatments, radiotherapy [77, 162] and image guided surgery (IGS) [60, 150, 182]. Also in this category are patient monitoring systems – e.g. for anaesthesia delivery – and the design and production of medical prosthetic devices.

Training This area has techniques in common with surgical planning. Visualisation of medical image data for education purposes, surgical simulations for training [14, 16, 17, 85], and other technologies that support the process of decision making in surgery. The use of force feedback devices (haptics) can be used to familiarise surgeons with a realistic 'feel' of a surgical procedure as part of their training [48, 84].

From the above descriptions, it is evident that image registration is an important component of CAS applications. No single medical imaging modality covers all requirements. Additionally, in surgical applications, imaging devices for intraoperative use are typically surgical microscopes, endoscopes, ultrasound, and eyesight. For effective application of CAS, these images must be combined with preoperative image data, and this requires fast and accurate image registration.

Some CAS applications already use *rigid* registration. In 2002, Muratore et al. [118] were able to register CT and 3D ultrasound (US) images using a phantom to assess the feasibility of performing surface based registration on a limited part of the vertebral surface. Recently, Leroy et al. [93] report results of intensity based rigid registration of 3D ultrasound with CT images of the

kidney, concluding that the performance depends on a number of manual pre-processing steps although accuracy and computation time are encouraging.

Non-rigid image registration is computationally more expensive than rigid registration [106] and this makes it time consuming to obtain accurate results. CAS applications often require intraoperative registration of multi-modal medical image data and in these situations it is desirable to achieve fast and accurate non-rigid registration to avoid unnecessary delays to the procedure and associated discomfort and potential risk to the patient.

2.5 Fast non-rigid image registration

2.5.1 Motivation

In Section 2.3 we classified non-rigid registration techniques as point based, surface based, intensity based and model based. Each of these has advantages and disadvantages.

Previous work directed at improving the speed of non-rigid medical image registration has primarily targeted intensity based methods. The pre-processing steps in point based and surfaced based methods, and the inherent complexity of model based methods make this a logical choice.

In 2002, Ourselin et al. [120] described a parallel architecture consisting of five dual-processor PCs connected by an Ethernet LAN and running a version of their own non-rigid registration algorithm reported in a previous work [119]. Each PC utilised two Pentium III 933MHz processors and this configuration achieved a non-rigid registration time of 95 seconds using $256 \times 256 \times 26$ voxel MRI data with $512 \times 512 \times 28$ voxel CT data.

Ino et al. [72] presented a parallel algorithm using the hierarchical FFD method described in [149, 154]. The implementation ran on a cluster of 64 off-the-shelf PCs (with 128 processors) and used three techniques: data distribu-

tion, data-parallel processing, and dynamic load balancing to perform non-rigid registration of CT images of the liver. The images were $512 \times 512 \times 159$ voxels and the registration time of 12 hours on a single processor was reduced to eight minutes on the 128 processor cluster.

In 2003, Shekhar et al. [159] introduced a hardware based architecture, which they named *fast automatic image registration* or FAIR. The system is based on intensity based registration using MI as the optimisation criterion. The calculation of the MI metric is accelerated by custom built hardware, which uses a combination of pipelining and parallel memory access and offers the potential for further acceleration with the use of distributed computing. In 2005, the same group reported a proof of concept implementation, named FAIR-II [21], which achieved 100-fold speed improvement for MI calculation when compared with a single processor implementation using a 3.2GHz Pentium III Xeon workstation. No results were reported for overall warp evaluation performance. The FAIR-II hardware was implemented in a PCI prototype board using an Altera Stratix EP1S40 FPGA running at an internal frequency of 200MHz.

Rohlfing and Maurer [136] also presented a parallel implementation of non-rigid registration, this time using 64 processors on a 128-CPU shared-memory supercomputer (SGI Origin 3800). The algorithm in this implementation was again an intensity based method, using normalised mutual information (NMI) as a similarity metric and a modified version of the B-spline FFD approach described in [149]. The study showed that execution times for some applications could be reduced from a number of hours to less than one minute although the authors rightly commented that the availability of high performance shared-memory parallel supercomputers in the operating theatre is not a foreseeable scenario at present.

In 2004 and 2005, Levin et al. [95, 96] proposed a method for accelerating point based non-rigid registration by using the fast tri-linear interpolation ca-

pability of modern graphics cards. Their implementation evaluates a TPS warp at discrete points on a configurable sized grid which overlays each image data slice. Interpolation by the graphics card is used to estimate the intensity values of the voxels within each grid cell. Using data sets of $512 \times 512 \times 173$ voxels and 92 user-defined landmarks, registration time was reduced from 148.2 seconds using a brute force implementation to 1.63 seconds using the grid based warp. This technique is discussed in greater detail in Section 3.13.

Klein et al. [78] present a comparison of acceleration techniques for non-rigid medical image registration. In this study, the authors examine techniques to accelerate intensity based registration by producing a faster convergence in the iteration phase of the process.

2.6 Summary

Much of the previous work on the acceleration of non-rigid image registration has been focused on parallel hardware implementations of well-known registration techniques. Other groups have attempted to accelerate the calculation of MI based image similarity metrics or to obtain faster convergence for the optimisation step in intensity based registration.

However, an important component of all non-rigid registration methods is the evaluation of the transformation function and this is also an area that can be a candidate for acceleration. For RBF based transformation functions there are a number of techniques which can in theory be used for medical image registration, although at the time of writing, this has not been reported.

In 1992, Beatson and Newsam [10] described mathematical techniques for rapid evaluation of RBFs based on hierarchical and multipole expansions. In 2005, Roussos and Baxter [140] presented another technique for rapid RBF evaluation, this time using the Fast Gauss Transform (FGT). More recently, Livne and Wright [100, 101] have presented initial numerical results for fast evaluation

of smooth RBFs in two and three dimensions using tensor product interpolation, which claims lower algorithmic complexity than the other methods.

2.6.1 Problem statement

This thesis is directed at the general problem of accelerating the performance of non-rigid medical image registration. In particular, we will develop methods that accelerate the evaluation of the transformation function, specifically we will target parametric transformation functions based on radial basis functions. We will answer the following questions :

1. How much can we optimise the run time of a RBF based warp evaluation?
2. What effect do such optimisations have on warping accuracy?
3. How do these optimised techniques behave under different inputs?
4. Are the optimised methods accurate and/or fast enough for AR applications?

To answer these questions, we will do the following :

1. Design and implement an experimental software application to control all testing and to record results.
2. Design and build synthetic test data sets, which will allow us to evaluate the accuracy of a warp and the suitability of different image similarity metrics.
3. Implement brute force warp evaluation techniques in both software and hardware versions.
4. Implement the grid-based warping technique of Levin et al. [95, 96] to compare with new, optimised implementations.

5. Extend the formulation of Livne and Wright [101] to three dimensions, adapt it for medical image registration and implement it in both software and hardware versions.

6. Perform a structured test to evaluate all algorithms under a range of inputs and parameters.

2.6.2 Discussion

The reasons behind the choices in the previous section are as follows :

- Accelerating the evaluation of the transformation function will potentially benefit all non-rigid warping methods.

- RBF based transformations are a common choice for non-rigid registration.

- CAS applications require fast and accurate non-rigid registration and often use calibration objects which can assist point based registration techniques.

- Many existing non-rigid acceleration techniques require additional hardware, sometimes costly. We will use consumer level hardware only.

Chapter 3

Radial basis functions for non-rigid registration

3.1 Introduction

This chapter provides details of the general theory of radial basis functions (RBF) as applied to non-rigid registration and describes the methods we have developed to accelerate the application of these functions using both hardware and software.

The chapter begins with a description of the mathematical techniques we will use to solve and evaluate RBF based warping functions, then continues with the methods we will employ to compare results, before describing the creation of synthetic test data and the algorithmic and implementation techniques we have used to accelerate performance.

To complete the chapter, we devise a formal test plan, the results of which will answer the questions addressed by this thesis as summarised in Section 3.17. The results themselves are presented in Chapter 4 and discussed in Chapter 5.

3.2 Radial basis functions

The RBF method is one of the most widely used techniques for approximating or interpolating scattered data in multiple dimensions [19]. Practical applications include 2D and 3D image mappings, interpolation of meteorological data from scattered monitoring stations, measurement of sea temperature distribution from discrete samples, and neural network learning situations. There is also a long list of mathematical applications, which includes the numerical solution of partial differential equations (PDE).

When considering interpolation, the aim is to approximate a real-valued function $f(x)$ for which we have a finite set of values $f = (f_1, \ldots, f_N)$ at the distinct points $X = \{x_1, \ldots, x_N\} \subset \mathbb{R}^d$. In this situation, we can choose a RBF, $s(x)$, to represent such an approximation, which has the general form :

$$s(x) = p(x) + \sum_{i=1}^{N} \lambda_i \phi \left(|x - x_i| \right), \qquad x \in \mathbb{R}^d \tag{3.1}$$

where p is a polynomial, λ_i is a real-valued weight, ϕ is a *basis function*[1], and $|x - x_i|$ is the Euclidean distance between x and x_i. So, a RBF could be described as a weighted sum of a radially-symmetric basis function, augmented by a polynomial term.

The basis function, ϕ, can take different forms, each of which is suitable for particular types of applications. Some common choices for ϕ are :

- The Gaussian : $\phi(r) = \mathrm{e}^{-\alpha r^2}$
- The multi-quadric : $\phi(r) = \sqrt{r^2 + c^2}$
- The thin-plate spline : $\phi(r) = r^2 \log r$
- The biharmonic spline : $\phi(r) = r$
- The triharmonic spline : $\phi(r) = r^3$

[1]The late William Light advocated the term *basic* function [94].

where r is Euclidean distance and c and α are constants. The Gaussian basis function is used for neural networks, the multi-quadric *inter alia* for fitting topographical data, the thin-plate spline (TPS) for fitting smooth functions of two variables and the polyharmonic splines for fitting to functions of three variables.

The last three spline functions in the above list share the property that they minimise specific energy quantities [138, 178], which makes them the smoothest interpolators. As such, these functions are suitable candidates for the non-rigid registration of multi-modal medical image data in two and three dimensions. We will implement a general method for solving and evaluating these functions, based on the thin-plate spline.

3.3 The thin-plate spline

The thin-plate spline is the 2D generalisation of a cubic spline. It was first generalised in mathematical terms by Duchon [35, 36] and Meinguet [116], and later by Wahba [178] and is widely used for encapsulating coordinate mappings in two-dimensional space. Bookstein [13] used thin-plate splines as a technique for modelling biological shape change and described an algebraic approach using matched pairs of landmarks. This produces a smooth deformation that can be used as an interpolant between the source and target spaces. As with all RBFs, the landmarks do not need to be regularly spaced and the TPS formulation guarantees to fit all landmark points exactly while minimising an approximate curvature over the whole deformed surface. Bookstein defines the 2D TPS function as :

$$f(x, y) = a_1 + a_x x + a_y y + \sum_{i=1}^{n} \lambda_i \phi \left(|P_i - (x, y)| \right) \tag{3.2}$$

where a_1, a_x and a_y are the coefficients of an affine transformation, which represents the behaviour of f at infinity. The remaining part of the function is

a weighted sum of the basis functions $\phi(r)$ where r is the Euclidean distance between the point (x, y) and each landmark point P_i. The structure of the TPS function reflects the general RBF formulation in Equation 3.1. As noted in Section 3.2, the basis function for a thin-plate spline in two dimensions takes the following form :

$$\phi(r) = r^2 \log r \tag{3.3}$$

The TPS transformation is invariant under translation and/or rotation of either set of landmarks, the only restriction being that landmark points may not be co-linear. The function $f(x, y)$ minimises the following quantity, which represents the bending energy of a thin metal plate on point constraints [138, 178] :

$$\iint_{\mathbb{R}^2} \left(\left(\frac{\partial^2 f}{\partial x^2} \right)^2 + 2 \left(\frac{\partial^2 f}{\partial x \partial y} \right)^2 + \left(\frac{\partial^2 f}{\partial y^2} \right)^2 \right) dxdy \tag{3.4}$$

As a result of TPS analysis, it is possible to produce a function that maps any point in the two-dimensional source space to its corresponding point in the target space in a manner which is least 'bent' and which interpolates smoothly between the known matching landmarks.

However, for deforming three-dimensional medical imaging data, it is necessary to extend the TPS formulation.

3.3.1 Solving the TPS transformation in 3D

To solve the TPS transformation in three dimensions, we follow the procedure formulated by Lapeer in 1999 [86, 88].

Extending Bookstein's two dimensional TPS function (Equation 3.2) to three dimensions gives us the following :

$$f(x, y, z) = a_1 + a_x x + a_y y + a_z z + \sum_{i=1}^{n} \lambda_i \phi \left(|P_i - (x, y, z)| \right) \tag{3.5}$$

	$d=1$	$d=2$	$d=3$	$d=4$
$m=1$	$-\frac{1}{2}r$	–	–	–
$m=2$	$\frac{1}{12}r^3$	$\frac{1}{8\pi}r^2\log r$	$-\frac{1}{8\pi}r$	–
$m=3$	$-\frac{1}{240}r^5$	$-\frac{1}{128\pi}r^4\log r$	$\frac{1}{96\pi}r^3$	$\frac{1}{64\pi^2}r^2\log r$
$m=4$	$\frac{1}{10080}r^7$	$\frac{1}{4608\pi}r^6\log r$	$-\frac{1}{2880\pi}r^5$	$-\frac{1}{1536\pi^2}r^4\log r$

Table 3.1: TPS basis functions for different orders m of derivatives in the functional and different image dimensions d.

Table 3.1 from Rohr [138, p. 195] lists the TPS basis functions that minimise the bending energy in systems of different dimensions, which – ignoring constants – shows that the RBF we need for 3D applications is $\phi(r) = r$.

The energy quantity minimised by Equation 3.5 using $\phi(r) = r$ is the three dimensional extension of Equation 3.4 [138, 178] :

$$\iiint_{\mathbb{R}^3} \left(\left(\frac{\partial^2 f}{\partial x^2}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x \partial y}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x \partial z}\right)^2 + \left(\frac{\partial^2 f}{\partial y^2}\right)^2 + 2\left(\frac{\partial^2 f}{\partial y \partial z}\right)^2 + \left(\frac{\partial^2 f}{\partial z^2}\right)^2 \right) dxdydz \quad (3.6)$$

Numerical implementation

The input to the TPS transformation is an arbitrary set of paired landmarks, each pair representing the 3D coordinates of the same feature in the source and target spaces. We wish to derive the mapping function f, which maps the set of n source points to the corresponding n target points :

$$(x'_n, y'_n, z'_n) = f(x_n, y_n, z_n) \quad (3.7)$$

By redefining Equation 3.5 in matrix form, we can describe the set of linear equations constituting this transformation by five matrices. Disregarding the

affine element for the moment, the weighted sum in Equation 3.5 can be written :

$$CR = Y \tag{3.8}$$

Where C is an $n \times n$ matrix defined in terms of the distances between the source landmark points :

$$C = \begin{bmatrix} \phi(r_{1,1}) & \phi(r_{1,2}) & \phi(r_{1,3}) & \cdots & \phi(r_{1,n}) \\ \phi(r_{2,1}) & \cdots & \cdots & \cdots & \phi(r_{2,n}) \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ \phi(r_{n,1}) & \phi(r_{n,2}) & \phi(r_{n,3}) & \cdots & \phi(r_{n,n}) \end{bmatrix} \tag{3.9}$$

where

$$r_{i,j} = |(x_i, y_i, z_i) - (x_j, y_j, z_j)| \tag{3.10}$$

which represents the Euclidean distance between two source landmark points and $\phi(r)$ is the RBF.

The matrix Y is a $n \times 3$ matrix containing the coordinates of the target landmarks :

$$Y = \begin{bmatrix} x'_1 & y'_1 & z'_1 \\ x'_2 & y'_2 & z'_2 \\ x'_3 & y'_3 & z'_3 \\ \vdots & \vdots & \vdots \\ x'_n & y'_n & z'_n \end{bmatrix} \tag{3.11}$$

The $n \times 3$ matrix, R, is the matrix of TPS parameters which we wish to compute :

$$R = \begin{bmatrix} \lambda_{x1} & \lambda_{y1} & \lambda_{z1} \\ \lambda_{x2} & \lambda_{y2} & \lambda_{z2} \\ \lambda_{x3} & \lambda_{y3} & \lambda_{z3} \\ \vdots & \vdots & \vdots \\ \lambda_{xn} & \lambda_{yn} & \lambda_{zn} \end{bmatrix} \tag{3.12}$$

Note that, from Equation 3.8, we could compute the TPS parameters by evaluating $R = C^{-1}Y$.

We construct a fourth matrix, A, which is an $n \times 4$ matrix formed from the source landmark positions representing the affine part of the transformation :

$$A = \begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & y_n & z_n \end{bmatrix} \tag{3.13}$$

Using these four matrices, together with a 4×4 zero matrix, O, we create a single, large matrix L as follows :

$$L = \begin{bmatrix} C & A \\ A^T & O \end{bmatrix} \tag{3.14}$$

The matrix L represents a system of linear equations which we need to solve :

$$V = L^{-1}Y \tag{3.15}$$

This system of equations may be solved by a number of techniques, such as Gaussian elimination or LU Decomposition [129]. The resulting matrix V has the following form :

$$V = \begin{bmatrix} \lambda_{x1} & \lambda_{y1} & \lambda_{z1} \\ \lambda_{x2} & \lambda_{y2} & \lambda_{z2} \\ \lambda_{x3} & \lambda_{y3} & \lambda_{z3} \\ \vdots & \vdots & \vdots \\ \lambda_{xn} & \lambda_{yn} & \lambda_{zn} \\ t_x & t_y & t_z \\ s_{xx} & s_{xy} & s_{xz} \\ s_{yx} & s_{yy} & s_{yz} \\ s_{zx} & s_{zy} & s_{zz} \end{bmatrix} \tag{3.16}$$

The first n rows of V are the TPS parameters (i.e. the matrix R), while the bottom 4×3 part of the matrix contains the coefficients of the affine transformation that can be rewritten as a 4×4 homogenous transformation matrix T as follows :

$$T = \begin{bmatrix} s_{xx} & s_{yx} & s_{zx} & t_x \\ s_{xy} & s_{yy} & s_{zy} & t_y \\ s_{xz} & s_{yz} & s_{zz} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

So, using the weights from matrix R and the linear transformation T, we can transform any point (x, y, z) in the 3D source space to its corresponding position (x', y', z') in the target space, by evaluating Equation 3.7, where :

$$f(x', y', z') = T \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + \sum_{i=0}^{n} ((\lambda_{xi}, \lambda_{yi}, \lambda_{zi}) \phi(|(x_i, y_i, z_i) - (x, y, z)|)) \tag{3.18}$$

The experimental software we develop will use the above approach to solve the system of linear equations in the matrix L (Equation 3.15) while allowing us to plug in a choice of RBF. Subsequently, by evaluating (for all points in the source data) Equation 3.18 using the calculated coefficients, we can apply a non-linear warp to synthetic and real-world three-dimensional data sets to allow us to explore the performance and accuracy of different RBFs and alternative software implementations.

3.4 Choice of radial basis function

As noted in Section 3.3, thin-plate splines are a special class of RBFs, which minimise the bending energy of a thin plate [19, 35, 138]. The basis functions

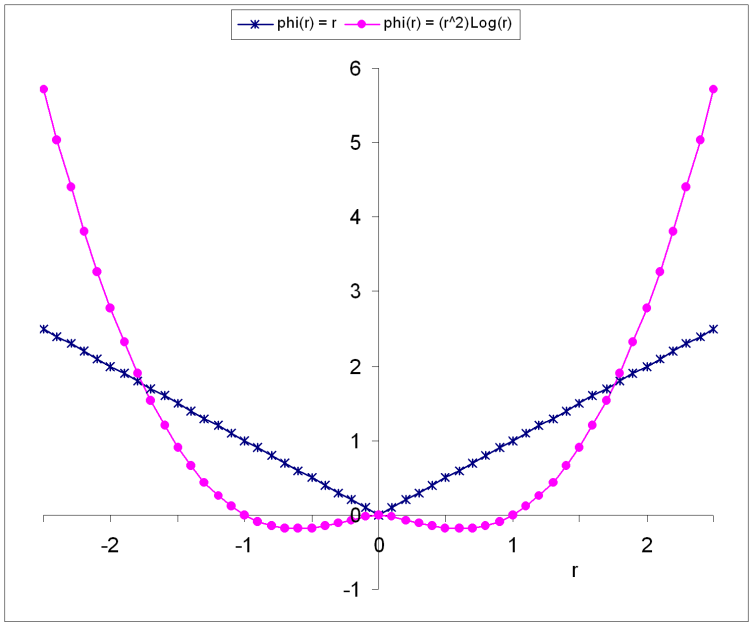


Figure 3.1: The TPS basis functions $\phi(r) = r^2 \log r$ and $\phi(r) = r$.

possessing this property depend only on the order of derivatives in the functional, m, and the image dimension, d [178, pp. 31–32]. Nevertheless, for these experiments, we are primarily concerned with optimising performance rather than evaluating the accuracy of different basis functions.

As we noted in Section 3.3.1, Table 3.1 suggests that the basis function (disregarding constants) for minimising the bending energy in a system with three dimensions is $\phi(r) = r$ which has the obvious advantage that it is faster to compute than the 2D TPS basis function shown in Equation 3.3. However, as $\phi(r) = r$ is only C^0 continuous in its centre (see Figure 3.1), we will also consider $\phi(r) = r^2 \log r$, which is C^1 continuous in its centre. This will be important for better justification of the "Fast RBF" warp evaluation technique as described in Section 3.14.

Therefore, to properly examine the behaviour and performance for the particular area of interest of this thesis, we will compare results for both of these basis functions.

3.5 Measuring the accuracy of a warp

Measuring the performance in terms of speed is straightforward, but for assessing the *accuracy* of a warp, we need not only ground truth data but also an objective method to compare the result of a warp to the ground truth.

For rigid, point-based registration, Maurer et al. [112, 114] suggested three useful error measurements for analysing registration accuracy :

- *Fiducial Localization Error* (FLE) – the error in locating fiducial points
- *Fiducial Registration Error* (FRE) – the root-mean-square (RMS) distance between corresponding fiducial points after registration
- *Target Registration Error* (TRE) – the displacement between homologous points other than fiducial points, after registration.

The authors found that, when FRE and FLE are measured in the RMS sense, then FRE approaches FLE (from below) with increasing N (the number of fiducial points), and that, for a given N, TRE is proportional to FLE. In 1998, Fitzpatrick et al. [44] derived an equation that allowed an approximation of the RMS value of TRE to be calculated where the ground truth data is not known.

For non-rigid registration, however, the situation is different. Over the years, there have been many metrics and methods proposed for producing some objective measurement of image similarity for assessing or guiding image registration [31, 32, 54, 106, 152, 153, 177]. There are obvious differences between techniques to measure similarity of mono-modal images compared with those for multi-modal images. In the former, one can assume similar image content and similar intensity values – perhaps after rescaling – which is very different to the multi-modal case. When comparing CT with MRI, for example, each modality targets different types of tissue – CT being ideal for bony structures and contrast enhanced tissues, while MRI is better for soft tissue. In the multi-

modal scenario, probabilistic metrics such as Mutual Information [177] are one of the most popular used today.

As we have control over the experimental set-up, we can design synthetic data that takes into consideration the characteristics of one or more image similarity metrics we might use, in order to produce a realistic reflection of registration accuracy. The test data we will use for the experiments is detailed in Section 3.6, but first we will review some of the available metrics [71] that we might employ for measuring image similarity. These are as follows :

- Mean squares (MS)
- Mean reciprocal square differences (MRSD)
- Mutual information (MI)
- Mattes mutual information (MMI)
- Normalised mutual information (NMI)
- Correlation coefficient (CC)
- Cardinality match (CM)

3.5.1 Mean squares

The mean squares metric is the simple mean squared voxel-wise intensity difference between two images. The ideal value of this metric – i.e. for identical images – is zero. This is suitable only for images obtained from the same modality and with the assumption that the intensity values representing the same point must be identical.

This metric is calculated as

$$MS(A, B) = \frac{1}{N} \sum_{i=1}^{N} (A_i - B_i)^2 \tag{3.19}$$

where A_i is the i-th voxel of image A, B_i is the i-th voxel of image B and N is the total number of voxels being compared.

3.5.2 Mean reciprocal square differences

This metric is similar to the mean squares metric, but but adds intensity differences after passing them through a bell-shaped function $\frac{1}{1+x^2}$. This is also suitable only for images obtained from the same modality and with the assumption that the intensity values representing the same point must be identical.

This metric is calculated as

$$MRSD(A,B) = \sum_{i=1}^{N} \frac{1}{1 + \frac{(A_i - B_i)^2}{\lambda^2}} \tag{3.20}$$

where A_i is the i-th voxel of image A, B_i is the i-th voxel of image B, N is the total number of voxels being compared and λ defines the capture radius (in intensity units). The optimal value of the metric is N.

3.5.3 Mutual information

As we mentioned in Chapter 2, the mutual information metric was first introduced as a tool for image registration by Viola and Wells [177, 183]. If we interpret the intensity values in each image as random variables, the mutual information metric measures how well one random variable describes the other.

The major difference between this metric and the previous two is that the images do not need to be acquired from the same modality. The mutual information metric makes no assumptions about the relationship between the two images – it assumes no functional correlation, only a predictable one.

For a detailed discussion of this metric, see [176]. Briefly, we can define the joint probability density function (PDF) between two images A and B, for example by normalising their 2D histograms – call this $P(i,j)$.

If we let $P_a(i)$ and $P_b(j)$ denote the corresponding marginal PDFs, we can use these to calculate the marginal and joint entropies

$$H(A) = -\sum_i P_a(i) \log P_a(i) \tag{3.21}$$

$$H(B) = -\sum_j P_b(j) \log P_b(j) \tag{3.22}$$

$$H(A,B) = -\sum_{i,j} P(i,j) \log P(i,j) \tag{3.23}$$

then the mutual information between the two images A and B is given by

$$MI(A,B) = H(A) + H(B) - H(A,B) \tag{3.24}$$

A number of specialisations have been made to the mutual information metric since the original version above. Two of these are described in the next sections.

3.5.4 Mattes mutual information

Mattes et al. [109, 110] use a variation of the mutual information metric, in which intensity values in each image are grouped into discrete *bins* evenly spread over the dynamic range of the images. The marginal and joint PDFs are calculated from these bins and the entropy values computed as in the previous section.

In the Viola and Wells implementation [177, 183], the PDFs are smoothed with a Gaussian kernel. Mattes et al. developed their technique for CT-PET registration, where inherent differences in the imaging modalities produce a significant degree of non-linear motion between the images. To recover this motion, the authors use a zero order B-spline kernel to compute the PDF of the fixed image A, and a third order B-spline kernel to compute the PDF of the moving image B.

3.5.5 Normalised mutual information

The normalised mutual information metric was proposed in 1999 by Studholme et al. [165], who give the following equation for its calculation

$$NMI(A,B) = \frac{H(A) + H(B)}{H(A,B)} \tag{3.25}$$

where the entropies $H(A)$, $H(B)$ and $H(A,B)$ are calculated as described in Section 3.5.3.

This metric is *overlap invariant*, which means that the metric does not depend on the degree of overlap of the two images. It has an optimal value of 2.0 and a minimum value of 1.0.

3.5.6 Correlation coefficient

The correlation coefficient is another metric which is best suited to mono-modal images. It assumes a linear relationship between the intensity values in the two images, so it can cope with changes in brightness or contrast.

It is calculated as follows

$$CC(A,B) = \frac{\sum_{i=1}^{N}(A_i - \bar{A})(B_i - \bar{B})}{(\sum_{i=1}^{N}(A_i - \bar{A})^2 \sum_{i=1}^{N}(B_i - \bar{B})^2)^{\frac{1}{2}}} \tag{3.26}$$

where A_i is the i-th voxel of image A, B_i is the i-th voxel of image B, N is the total number of voxels being compared and $\bar{A}$ and $\bar{B}$ are the mean intensity values of images A and B respectively.

3.5.7 Cardinality match

This is perhaps the most basic of metrics, suitable only for mono-modal images with identical intensity values representing the same tissue type.

The cardinality match is simply an indication of how many voxels are an exact match between the two images. If the values do not match, no account is

made of the amount by which they differ - all matches are good, all mismatches are bad.

This metric has an ideal value of 1.0 indicating that the images are identical. At the other end of the scale, a score of 0.0 indicates that the intensity values of every pair of corresponding voxels are different.

For practical image registration purposes the cardinality match metric is not suitable. However, by designing synthetic test data appropriately we can still make some use of this measurement in the evaluation of registration accuracy.

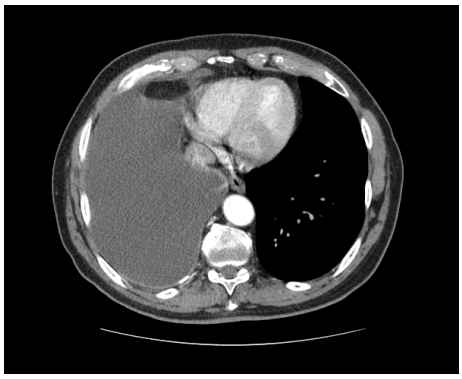
3.6 Design of synthetic test data

Synthetic data is necessary to confidently evaluate the accuracy of a warp. Unless we have ground truth data, we can only use statistical measures such as mutual information to assess the result of image registration.

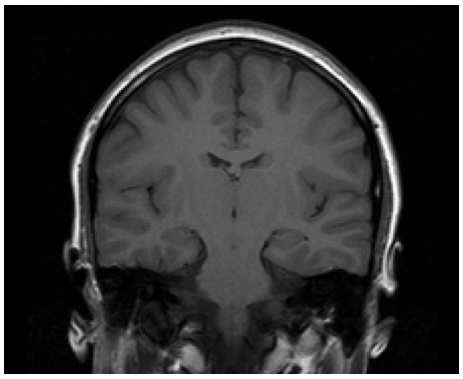
However, designing *useful* test data is not straightforward. We wish to be able to easily identify and quantify mis-registration but we also need to measure the effect that the optimised evaluations have on registration accuracy. Can we be sure that the test data is sensitive enough – and, conversely, not too sensitive – to show such differences, which may be small?

The answer of course is no - there is an infinite variety of image content that we might wish to input to the registration algorithms and we cannot test all permutations. However, particularly for medical image registration, we do know *something* about the content, which we can use to guide us in preparing test data.

Many organs and structures within the body produce relatively large areas of homogenous intensity response. For example, a CT scan of the liver or an MRI image of grey matter from a brain scan – see Figure 3.2. In these images, any registration error will tend to show mostly (or only) where there is a boundary with an area of a different intensity, that is, some contrast, or an



(a) CT section showing the liver.



(b) MRI coronal brain scan.

Figure 3.2: CT and MRI images showing areas of homogenous intensity.

"edge".

So, the first issue to consider for the synthetic test data is that we need to have areas of sharp contrast – large changes in intensity between adjacent voxels, to provide us with edges. If we construct a *difference image* by subtracting the result of a warp evaluation from the ground truth data, any registration inaccuracies will produce a pattern in this image that echoes the shape of the edge.

The second issue is to ensure that registration errors are not only made visible in areas of high contrast but also in areas of similar intensity. If we introduce a linear intensity gradient into these areas of the test data, so that adjacent voxels *always* have different intensities, we can make sure that a mis-alignment of just one voxel in a particular direction will also produce a detectable pattern in the difference image. To clarify, we can anticipate that mis-registration of more than a certain number of voxels in any direction would be readily identified visually. However, for smaller misalignments, we can magnify the visual effect in the difference image by incorporating a smooth gradient in the intensity values such that a misalignment of say two voxels in any direction produces a *stronger* pattern in the difference image than a misalignment of one voxel.

In proposing this, we should be aware that, by using a simple linear gradient

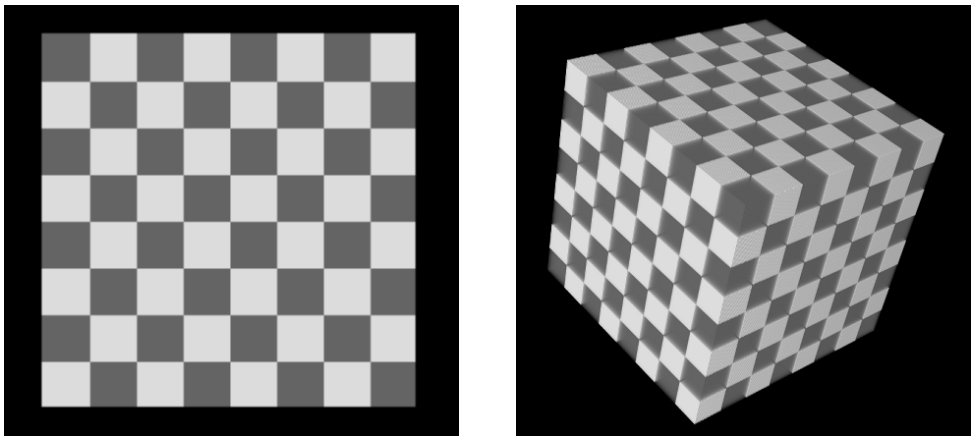
within the test data, we may inadvertently bias the tests in favour of techniques that estimate voxel intensity by linear interpolation. In fact, some of the optimised techniques will do exactly that (see Section 3.10.1). The test data we design will attempt to mitigate the effect of this bias as we shall see shortly.

Lastly, it would be useful to have a *visual* impression of the quality of a warp. This is more easily achieved if the test data is not just random, but has some familiar geometric structure in which we will be able to identify any inconsistencies in warping accuracy more naturally.

So, taking all the above points into consideration, we will describe the basic sets of data that will form the ground truth.

3.6.1 The ground truth

The basic component of the test data is a checkered cube. This is a symmetrical regular cube, centered within a region of black space, containing a number of alternating dark and light blocks (themselves cubes). For these tests, the overall data set is $256 \times 256 \times 256$ voxels and the cube is $200 \times 200 \times 200$ voxels and patterned with an $8 \times 8 \times 8$ checker effect which gives us 512 *cells* within the cube, each of $25 \times 25 \times 25$ voxels.



(a) Sample 2D section. (b) 3D view.

Figure 3.3: Basic component of synthetic test data.

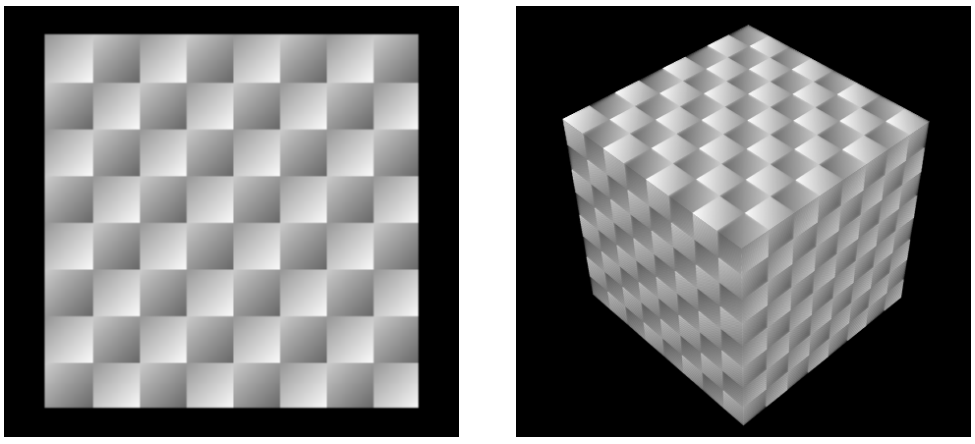
This gives us a familiar geometrical figure, which we can easily assess visually to see if there is any corruption of the expected regular pattern. Figure 3.3 shows a 2D section and a 3D representation of this basic data.

The cube also provides two levels of high contrast – the first level is the whole cube itself in relation to the surrounding black background, and the second level comprises the intersections of the light and dark cells *within* the cube. So we have some image contrast at two levels of resolution, albeit crudely.

This cube is not itself used for any evaluations, but is the starting point for generating four sets of ground truth data, each of which is described in the following sections.

3.6.2 Regular linear gradient

The first variation of the basic data involves the creation of a regular linear gradient across the intensity values in each cell of the cube. The gradient runs diagonally within the cell in three dimensions, so that each voxel is different from its immediate neighbours.



(a) Sample 2D section. (b) 3D view.

Figure 3.4: Synthetic data - regular linear gradient.

Figure 3.4 shows a 2D section and a 3D view of this data. The formula used to determine the actual intensity value within each cell is as follows :

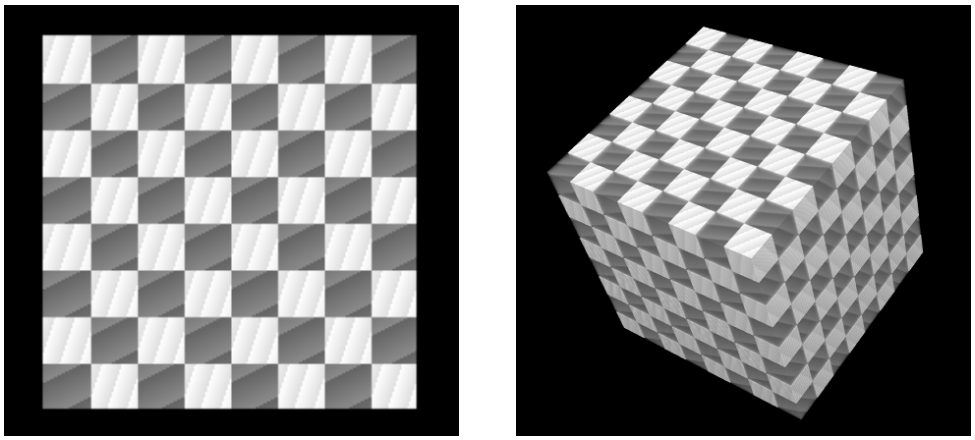
$$\text{Voxel intensity:} \quad v(x,y,z) = \begin{cases} 255 - 2(x+y+z) & \text{for a light cell} \\ 100 + 2(x+y+z) & \text{for a dark cell} \end{cases} \tag{3.27}$$

where x, y and z are the positional offsets in voxels from the lower left front corner of the cell being generated.

3.6.3 Irregular linear gradient

The second variation of the basic data is similar to the first, in that a linear gradient runs across the intensity values in each cell of the cube. This gradient also runs "diagonally" within the cell in three dimensions, so that each voxel is different from its immediate neighbours.

However, the gradient is shorter than in Section 3.6.2 so it repeats itself one or more times within each cell. Also, the gradient is different between dark and light cells and neither is a "regular" 45 degree gradient.



(a) Sample 2D section. (b) 3D view.

Figure 3.5: Synthetic data - irregular linear gradient.

Figure 3.5 shows a 2D section and a 3D view of this irregular gradient test data. The formula used to determine the intensity values within each cell is as follows :

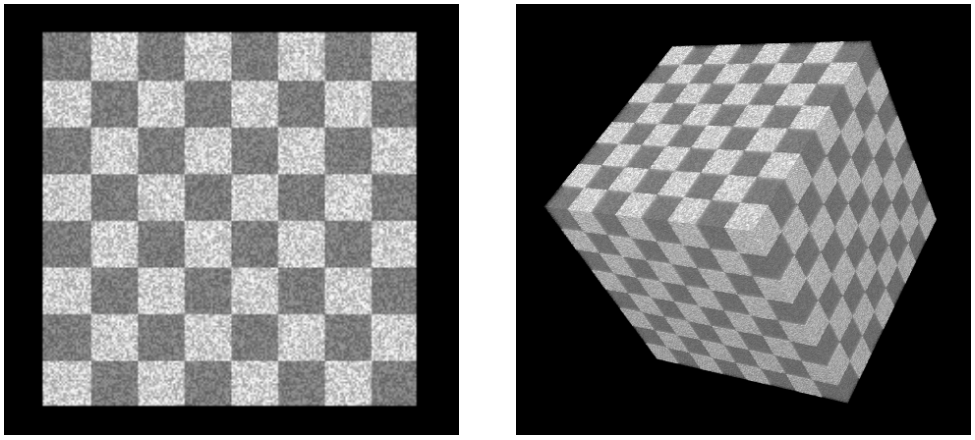
$$\text{Voxel intensity:} \quad v(x,y,z) = \begin{cases} 255 - ((3x + y + z) \mod 32) & \text{for a light cell} \\ 100 + ((x + 2y + 3z) \mod 40) & \text{for a dark cell} \end{cases} \tag{3.28}$$

where x, y and z are the positional offsets in voxels from the lower left front corner of the cell being generated.

Because of the repeating gradients, this test data gives us an additional level of granularity for potential mis-alignment at edges. It also partially overcomes the tendency for linear interpolators in an optimised evaluation algorithm to correctly "guess" a voxel value based on its neighbours.

3.6.4 Checkered cube plus noise

The third test data set is simply the basic two-tone cube from Section 3.6.1 with some random noise added. There are no gradient fills.



(a) Sample 2D section. (b) 3D view.

Figure 3.6: Synthetic data - checkered cube plus noise.

Figure 3.6 shows a 2D section and a 3D view of this noisy test data. The formula used to determine the intensity values within each cell is as follows

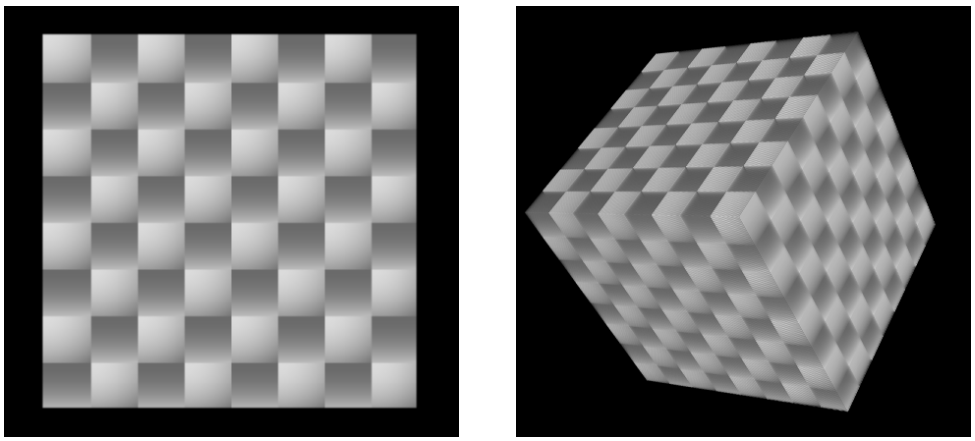
$$\text{Voxel intensity:} \quad v(x,y,z) = \begin{cases} 255 - R_l & \text{for a light cell} \\ 95 + R_d & \text{for a dark cell} \end{cases} \tag{3.29}$$

where x, y and z are the positional offsets in voxels from the lower left front corner of the cell being generated, R_l is a random number between zero and 100, and R_d is a random number between zero and 60.

This test data should be sensitive not only to mis-alignment but will also detect inappropriate interpolation. Effectively, the noise adds more fine resolution "edges" to the data. This test data may be able to give us some information about the suitability of different similarity metrics for noisy data. We will discuss this further in Chapter 5 Section 5.1.

3.6.5 Non-linear gradient

The fourth and final variation of the basic data involves the creation of a non-linear gradient across the intensity values in each cell of the cube. Again, this gradient runs "diagonally" within the cell in three dimensions, so that each voxel is different from its immediate neighbours.



(a) Sample 2D section. (b) 3D view.

Figure 3.7: Synthetic data - non-linear gradient.

Figure 3.7 shows a 2D section and a 3D view of this test data set. The formula used to determine the intensity values within each cell is as follows

$$\text{Voxel intensity:} \quad v(x,y,z) = \begin{cases} 220 - 100\frac{2x^2+3y^2}{5n^2} & \text{for a light cell} \\ 100 + 100\frac{4y^2+z^2}{5n^2} & \text{for a dark cell} \end{cases} \tag{3.30}$$

where x, y and z are the positional offsets in voxels from the lower left front corner of the cell being generated and n is the dimension of each cell (in this case, 25). The coefficients in Equation 3.30 were determined empirically.

This test data is perhaps the most visually realistic when compared to real medical image data – a mixture of soft and hard edges, areas of similar but varying intensity and some curved boundaries within the cells.

3.6.6 Deforming the ground truth

The previous sections have provided us with four sets of ground truth data

- Regular linear gradient
- Irregular linear gradient
- Checkered cube with random noise
- Non-linear gradient

We now need to deform this data so that we can exercise the warping algorithms using the deformed data and then compare the results to the ground truth.

To produce the warped test data sets we first generate a set of landmarks in the ground truth images. In order to verify the placement of the landmarks visually and to ensure that they are reasonably distributed within the data, we programmatically place them at the intersections of the dark/light cells in the basic cube data.

Remember that – for this study – we are not concerned with the *absolute* accuracy of the warp, we purely wish to compare performance of various warping algorithms and implementations. As long as we can produce a benchmark warp for the test data, we can achieve the desired result. Specifically, we do not wish to have poor coverage of landmarks nor inaccurate landmark placement to affect the results. For this reason, programmatic landmark generation is ideal.

Having obtained one set of landmarks in the ground truth data, a random TPS transformation is applied to these landmarks to produce a set of corresponding points in a warped data set. We now have a complete set of landmark pairs, which we use to produce the warped data sets as follows

1. Each landmark pair is swapped so that the source point becomes target point and vice versa.

2. The resulting set of landmark pairs is used to solve two TPS transformations as described in Section 3.3.1 – one using $\phi(r) = r^2 \log r$ and one using $\phi(r) = r$.

3. Each of the above TPS transformations is applied to all voxels in all four data sets, producing the synthetic test data.

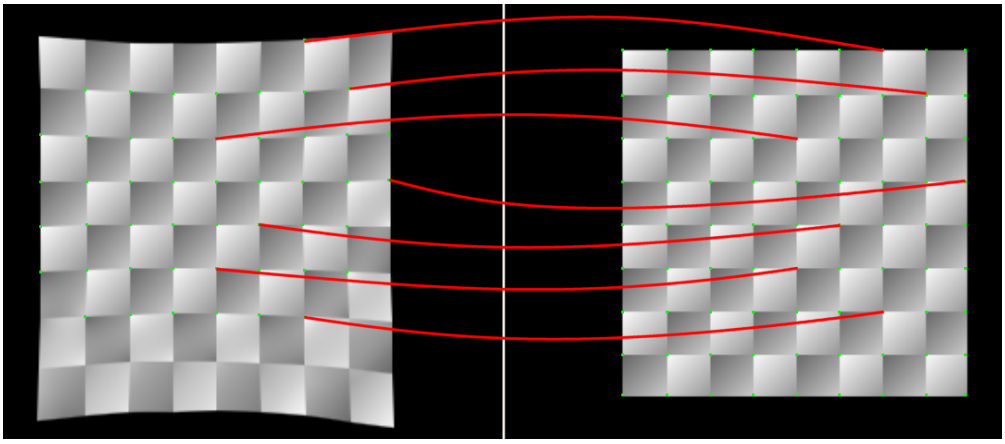


Figure 3.8: Selected landmark pairs in synthetic test data.

Now, we have eight sets of test data – each of the four ground truth images has been warped using both RBFs. In addition, we have a set of landmark pairs which we can use to perform a warp *back* to the ground truth and so compare the warping evaluation implementations. Table 3.2 shows the spatial distortions in the test data sets, in millimetres.

RBF	Minimum	Maximum	Range	Average	Std. Dev.
$\phi(r) = r$	0.000000	35.978528	35.978528	6.001375	6.119655
$\phi(r) = r^2 \log r$	0.000000	47.304805	47.304805	6.253584	6.679255

Table 3.2: Spatial distortions in millimetres in the synthetic test data.

Figure 3.8 shows a selection of matching landmark points in the regular linear gradient test data set, which has been warped using a RBF of $\phi(r) = r$ and Figure 3.9 shows the four $\phi(r) = r$ data sets as 3D projections. The four $\phi(r) = r^2 \log r$ data sets are visually similar so are not reproduced here.

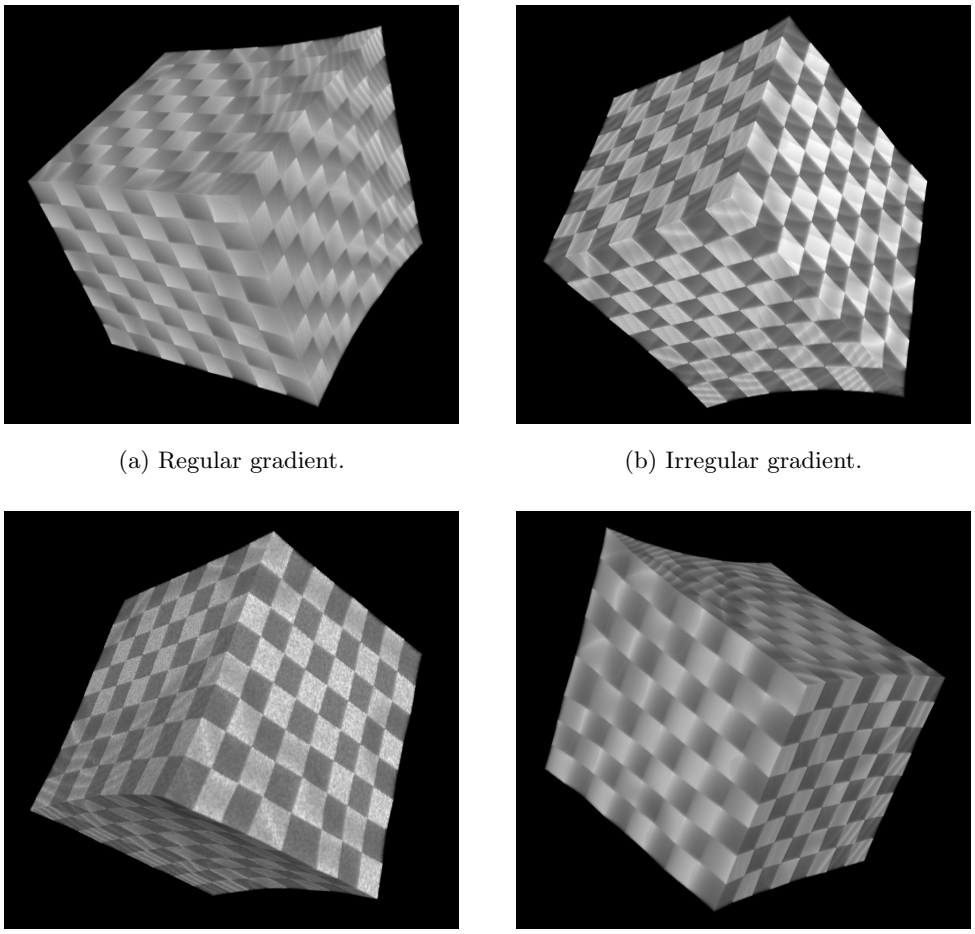
Note that all four data sets in Figure 3.9 have undergone exactly the same transformation[2] – only the voxel intensities differ. The landmarks are identical and so the mapping function between each data set and its corresponding ground truth is also identical. This is important for testing because it will help us to make a more useful judgement of the warp accuracy as explained in the introduction to Section 3.6.

3.7 Choice of image similarity metric

Table 3.3 shows the metrics previously detailed in Section 3.5, together with the corresponding benchmarks[3] for the $256 \times 256 \times 256$ synthetic test data.

[2]The 3D rendering method used to generate Figure 3.9 produces some visible 'banding' artifacts in these images, which are not present in the data – the 2D samples in Sections 3.6.2 to 3.6.5 give an accurate representation of the test data.

[3]Benchmarks produced using identical ground truth data sets for comparison.



(a) Regular gradient.

(b) Irregular gradient.

(c) Checkered cube plus noise.

(d) Non-linear gradient.

Figure 3.9: Various 3D views of synthetic data sets for $\phi(r) = r$.

Of these metrics, only those based on mutual information are suitable for multi-modal images. Although the correlation coefficient can accommodate changes in brightness and contrast, this is not sufficient for the differences in image content between say CT and MRI.

According to [165] the NMI metric is stable in more circumstances than the other two mutual information metrics (MI and MMI) we described. In addition, as Table 3.3 shows, it has an optimum value which is independent of the content of the test data.

So, although we should not blindly accept the NMI metric as an absolute

Acronym	Description	Benchmark
MS	Mean Squares	0.000000
MRSD	Mean Reciprocal Squared Difference	16777216
MI	Mutual Information	
	Regular linear gradient	3.250990
	Irregular linear gradient	2.306447
	Checkered cube + noise	3.933237
	Non-linear gradient	3.528989
MMI	Mattes Mutual Information	
	Regular linear gradient	-1.559989
	Irregular linear gradient	-1.508737
	Checkered cube + noise	-1.734901
	Non-linear gradient	-1.710639
NMI	Normalised Mutual Information	2.000000
CC	Correlation Coefficient	1.000000
CM	Cardinality Match	1.000000

Table 3.3: Similarity metrics - descriptions and benchmarks.

measure of registration accuracy (see Section 5.1 for one reason why), with the current state of research in this field, it is arguably one of the best metrics we have available.

As NMI is suited to both mono-modal and multi-modal scenarios, we will use this metric as the primary image similarity measurement for all experimental work.

3.8 Experimental methods

A full description of the software developed for performing these experiments is given in Appendix A. The main features provided by the software are as follows :

- The ability to load two three-dimensional images and to view them side by side as 2D slices along any axis.
- Independent scrolling through the two sets of image slices to identify and mark corresponding features to form a set of landmarks.
- The ability to save and load a set of landmarks to/from disk.
- Selection of RBF, interpolation option and any technique-specific warp evaluation parameters.
- Use landmarks to solve the TPS function and calculate the TPS weights and affine transformation coefficients.
- Perform the warp evaluation using a selected technique.

These steps are described in more detail in the following sections. We assume that we have identified a number of matching features in the two images and have manually marked these landmark pairs, or that we have a set of landmarks generated programmatically during the construction of the synthetic test data. Such a set of landmark pairs is the main input to the procedure that solves the warping function using the method described in Section 3.3.1.

3.9 Solving the warping function

The landmark positions are converted to floating point 3D coordinates according to the voxel dimensions of the input data sets[4]. The algorithm then proceeds as follows

1. Memory is allocated for the matrices L (Eq. 3.14), Y (Eq. 3.11) and V (Eq. 3.16), all containing double precision floating point elements.

[4]Note that the voxels need not be the same size in all three dimensions, as is common in medical image data.

2. Target landmark coordinates input by the user are loaded into matrix Y.

3. The top right – matrix A (Eq. 3.13) – and bottom left – A^T – of matrix L are populated using the source landmark coordinates input by the user.

4. The upper left block of L – the matrix C – is computed using Equation 3.9 on page 35.

5. We then solve $V = L^{-1}Y$ by first performing LU decomposition[129, p. 43] on L, followed by forward/backward substitution[129, p. 47] using Y.

6. The spline coefficients – matrix R (Eq. 3.12) – and the affine transformation – matrix T (Eq. 3.17) – are extracted from matrix V as the final result.

Note that, although the basis functions we are using in these experiments are both thin-plate splines (i.e. they minimise the bending energy functions in Equations 3.4 and 3.6), the technique we are using could in practice be applied to *any* suitable RBF with little modification.

To summarise, as a result of solving the warping function in Equation 3.5, the inputs we need to evaluate the warp on a 3D image are as follows :

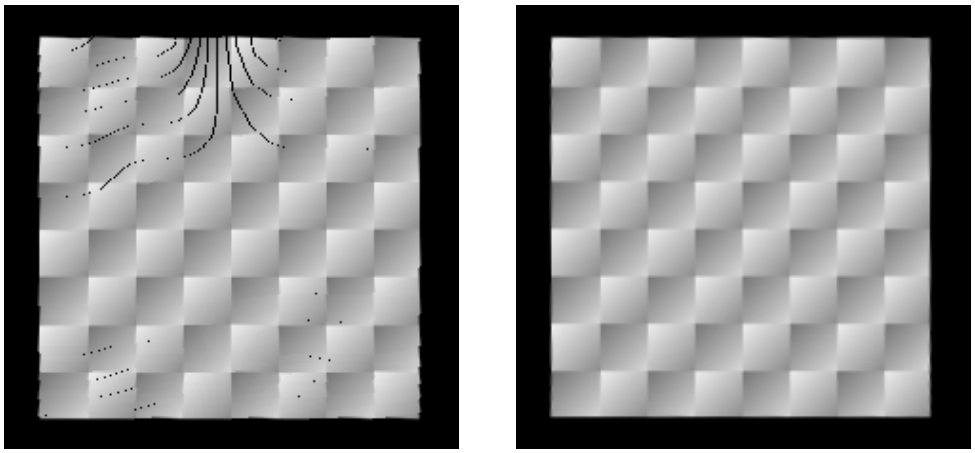
- The set of n landmark coordinates $(x_1, y_1, z_1) \cdots (x_n, y_n, z_n)$ in the source image[5].

- The set of n 3D weights – one for each landmark – representing the spline coefficients, $(\lambda_{x1}, \lambda_{y1}, \lambda_{z1}) \cdots (\lambda_{xn}, \lambda_{yn}, \lambda_{zn})$

- The 4×4 affine transformation matrix, T.

Before detailing the methods we will use to evaluate the warping function using the above information, there is one more consideration discussed in the next section.

[5]The corresponding points in the target image are *not* required.



(a) Forward mapping. (b) Backward mapping.

Figure 3.10: Cross section of synthetic cube data set showing tearing artifacts caused by forward mapping.

3.9.1 Forward mapping versus backward mapping

By mapping voxels from the source data to the target data (forward mapping) we may not always fully populate the target voxel set. This is because, due to numerical precision and rounding, there is a high probability that more than one source voxel will be mapped to the same target voxel, leaving "holes" or "tearing" in the target data set because the 1:1 mapping will have been lost. Figure 3.10a shows an example of this effect.

To overcome this problem, we need to calculate the *inverse* of the mapping from source to target so that we can consider every voxel in the target data set and map this *back* to a voxel in the source data set (backward mapping). The simplest way to do this is to reverse the roles of the source and target landmarks before we solve Equation 3.5. Then the TPS coefficients and affine transformation will represent the mapping from target space to source space and we can use backward mapping to populate the target data. Figure 3.10b shows the same slice of the synthetic data set as in Figure 3.10a, but this time warped using backward mapping.

3.9.2 Interpolation during backward mapping

When using backward mapping, it is possible that we may not map exactly to the coordinates of an existing voxel in the source data set. In this case we have to use an interpolator, a variety of which have been proposed for this purpose [54]. The three most popular methods are: nearest neighbour interpolation (NNI), tri-linear interpolation (TRI) and partial volume interpolation (PVI).

Collignon et al. [29, 103] introduced PVI as an interpolator specifically designed to be used with MI techniques. PVI is not an interpolator in the sense that it can be used to create an interpolated image [127] but is used to update the joint histogram of two images in a way which does not introduce new grey values into the data and facilitates the calculation of an MI based metric.

As the applications we are interested in require the creation of a warped image and we are not using iterative MI-based registration techniques, the experimental software will simply provide a choice between NNI and TRI during backward mapping so that we can explore any differences in performance and accuracy between these two techniques, which might arise as a result of the warp evaluation optimisations.

3.10 Evaluating the warping function

Once we have identified matching anatomical landmarks in the source and target data sets and calculated the set of 3D weights and the affine transformation associated with these landmarks (Section 3.9), we are ready to evaluate the warping function in Equation 3.18 for each voxel in a data set to map between the source and target image spaces.

As described above in Section 3.9.1, to avoid the creation of "tearing" artifacts in the resulting warped data, we will only use backward mapping – i.e. for each voxel in the target space we will map *back* to the source image to retrieve

the required voxel value.

Algorithm 1 shows the pseudocode for a routine which implements Equation 3.18 to transform an arbitrary point (x, y, z) to its corresponding warped position (x', y', z').

Algorithm 1 warp(x, y, z) : Evaluate warping function for a given point.

1: $(x', y', z') \leftarrow (0, 0, 0)$ {initialise result}
2: **for** $i = 1$ to n **do** {for each landmark}
3: $\quad (x_d, y_d, z_d) \leftarrow (x, y, z) - (x_i, y_i, z_i)$
4: $\quad r \leftarrow \sqrt{{x_d}^2 + {y_d}^2 + {z_d}^2}$ {distance to current landmark}
5: $\quad u \leftarrow \phi(r)$ {evaluate basis function}
6: $\quad x' \leftarrow x' + u\lambda_{x_i}$ {apply 3D weights}
7: $\quad y' \leftarrow y' + u\lambda_{y_i}$
8: $\quad z' \leftarrow z' + u\lambda_{z_i}$
9: **end for**
10: $(x', y', z') \leftarrow (x', y', z')+ \mathrm{T}(x, y, z)$
11: **return** (x', y', z')

After initialising the warped position in step 1, the weighted sum of RBFs is applied in steps 2 through 9 as follows.

The Euclidean distance, r, between the original point and each landmark in turn is calculated and then used to determine the value, u, of the RBF for that landmark (step 5). The 3D weights (spline coefficients) previously calculated for that landmark are then scaled by u and accumulated into the warped position, (x', y', z').

After the weights for *all* landmarks have been accumulated, the resulting warped position is augmented by an affine transformation of the original point, (x, y, z), using the transformation matrix T (Equation 3.17). The warped coordinates are returned as the result of the function.

3.10.1 Warping function evaluation techniques

In Section 3.4 we identified two RBFs for comparative testing. Section 3.9.1 identified two further options - the choice of nearest neighbour or tri-linear interpolation during backward mapping. Finally, we consider five different techniques to implement the warping function evaluation – three of which have been previously studied for non-rigid medical image registration and two of which have not.

Brute force evaluation in software This is a common and straightforward software implementation of Equation 3.18, performed for every voxel in the data set. This implementation is written in C++ and runs on the CPU using system RAM. It will form the basis for the *gold standard* as we shall see in Chapter 4.

Brute force evaluation - hardware accelerated This is another implementation of the brute force evaluation, but this time written as a pixel shader running primarily on the GPU and using texture memory for all inputs and outputs. Landmark positions and 3D weights are encoded in 32-bit floating point textures and the affine transformation matrix supplied in shader registers. The result of the warp is read back from texture memory.

Grid based evaluation - hardware assisted This implementation follows the method described by Levin et al. [95, 96] that evaluates the warp at regularly spaced grid points on a mesh which is superimposed on each 2D image slice. The resolution of the grid is configurable and it is intended that each grid cell will enclose multiple voxels. Effectively, the mesh is warped and drags the image data (in the form of a texture) along with it. This technique uses the fast tri-linear interpolation capability of the GPU to approximate individual voxel values *within* each grid cell.

Software accelerated "Fast RBF" evaluation This C++ software implementation uses the method proposed by Livne and Wright [101] for fast evaluation of smooth[6] RBFs and is described in detail in Section 3.14. It has not yet been studied in a medical image registration context, nor has it been applied to any practical problem [190].

Hardware accelerated "Fast RBF" evaluation The final technique combines the hardware and software accelerations by using pixel shaders, floating point textures and the GPU to re-implement the software accelerated RBF evaluation technique above. Again, this has not previously been applied to non-rigid medical image registration.

The following sections describe each of the above implementations in more detail.

3.11 Brute force evaluation in software

3.11.1 Overview

The brute force method, implemented in software, is the benchmark. This algorithm is a direct implementation of Equation 3.18 on page 37, which operates on every voxel in the target image space, mapping *back* to a value from the source space. All calculations are performed using double precision[7] floating point variables.

3.11.2 Algorithm implementation

Algorithm 2 shows the pseudocode for this implementation. Each target voxel is first converted to a floating point 3D position by multiplying the xyz voxel

[6] A subsequent paper by the same authors [100] details improvements for piecewise smooth RBF's, such as the thin-plate spline although these refinements are not considered significant for the intended application areas described in this thesis. Chapter 6 discusses this further.

[7] 64 bit floating point value, range $1.7E \pm 308$ (15 digits).

indices by their corresponding voxel dimension (step 2). This step is necessary because voxels from different data sets – particularly multi-modal – may not be the same size.

Algorithm 2 Brute force evaluation in software.

1: **for** each target voxel $V_t(x, y, z)$ **do**
2: $(x_t, y_t, z_t) \leftarrow V_t \times (voxelsize)$ {calculate target voxel position}
3: $(x_s, y_s, z_s) \leftarrow$**warp**$(x_t, y_t, z_t)$
4: $V_t(x, y, z) \leftarrow V_s(x_s, y_s, z_s)$ {set new voxel value}
5: **end for**

In step 3, the target voxel position is warped to the corresponding position (x_s, y_s, z_s) in the source image using the "warp" function from Algorithm 1 in Section 3.10 on page 60. Finally, in step 4, the source image is sampled at the warped position and the resulting intensity value stored in the original target voxel $V_t(x, y, z)$.

This algorithm is the simplest, containing as it does no hardware or software optimisations[8]. As a direct implementation of the warping function on a voxel by voxel basis, the output of this algorithm, and the time it takes to execute, will be the standard against which the optimised evaluations are measured. Chapter 4 presents the results of these comparisons.

3.12 Brute force evaluation - hardware accelerated

3.12.1 Overview

This evaluation technique is purely a hardware optimised version of the brute force method in the previous section. By "hardware", we mean that we use the programmable abilities of the graphics card and take advantage of the parallel processing capabilities of the card and GPU in order to execute the brute force

[8]Other than those applied by the optimising C++ compiler.

algorithm much faster than we could using conventional memory and the CPU.

One other difference is the numerical precision available during all computations. The graphics processor and texture memory currently only natively support 32-bit floating point precision, so we expect the results of this technique to show some differences from the 64-bit precision software implementation in the previous section.

3.12.2 Algorithm implementation

The algorithm that implements Equation 3.18 is coded in assembly language as a DirectX *pixel shader*, which is executed by the GPU for every pixel[9] we render.

The primary inputs to the algorithm – the image to be warped, the TPS weights and the landmark positions – are supplied to the pixel shader by encoding them into 32-bit floating point textures in the graphics card memory. The remaining parameters – affine transformation matrix, voxel sizes, data dimensions – are supplied to constant registers in the pixel shader before rendering.

To effect the warp, the source image is first loaded to graphics card memory as a 3D texture. Then the *target* image is rendered as a textured quad, slice by slice, with a 1:1 mapping of voxels to screen pixels.

As each slice is rendered, the pixel shader is invoked for every pixel (voxel) in that slice. This shader uses the TPS weights and landmarks from the 32-bit floating point textures to compute the warped position of the target voxel using Equation 3.18. Finally, the shader samples the source data texture at that position and returns the intensity value obtained to the frame buffer at the target voxel (pixel).

After each slice has been rendered, the contents of the frame buffer are read back into system memory and used to populate the corresponding voxels in the

[9]Strictly we should say 'fragment' but here we use a 1:1 mapping to pixels.

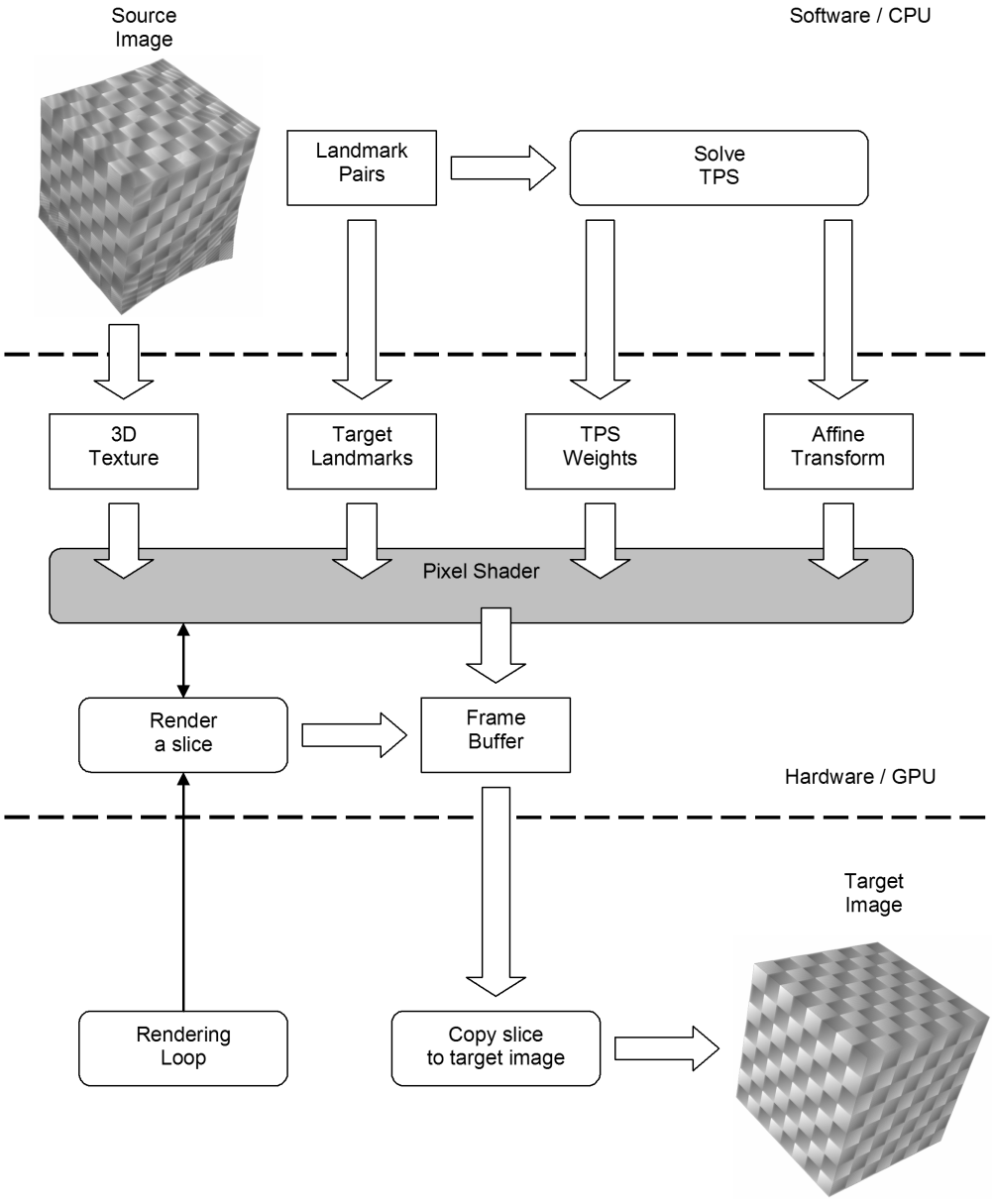


Figure 3.11: Brute force evaluation - hardware accelerated.

target data set. Once all slices have been rendered, the warp is complete.

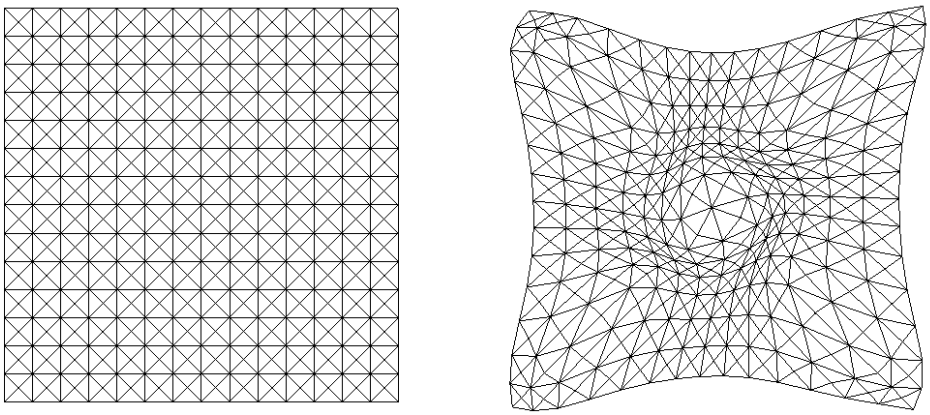
Figure 3.11 on page 65 shows a schematic diagram of this process. The area between the dotted lines represents the processes taking place on the graphics hardware. The source code of the shaders is listed in Appendix C.

3.13 Grid based evaluation - hardware assisted

3.13.1 Overview

This implementation is based on the method described by Levin et al. [95, 96]. There are two elements to the optimisation in this approach, one algorithmic and one based on hardware accelerated graphics.

The algorithmic optimisation is achieved by reducing the number of points that need to be evaluated through the warping function. Whereas the brute force methods evaluate the function at *every* target voxel position, this method only evaluates the warp at regularly spaced grid points on a mesh which is superimposed on each 2D image slice in the 3D data. The granularity of the grid is configurable. Figure 3.12 shows the grid mesh – Figure 3.12a before warping, and Figure 3.12b after applying an arbitrary warping function. Note that the grid is built using explicit triangulation, by adding a central vertex in each square grid cell. This is the same technique employed by Levin et al. [95, 96] and avoids incorrect interpolation due to the graphics driver internally rendering a quad as two triangles.



(a) Unwarped grid vertices.

(b) Warped grid vertices.

Figure 3.12: Regular grid, triangulated for better interpolation, demonstrating warping of vertices.

The second element of optimisation in this technique is the use of hardware accelerated tri-linear interpolation by the graphics card. The data to be warped is loaded into a 3D texture in memory on the graphics card and 3D texture coordinates are used to "tie" the texture to the vertices of the grid mesh. As each vertex is processed through the warping function, it is attached to a new position in the texture based on the position to which its original coordinate would warp. In this way, the grid stays still but the texture warps across it. The voxels covered by the grid cells are interpolated from the source data in the 3D texture and the resulting values used to populate the target image.

Algorithm 3 Grid based evaluation - hardware assisted.

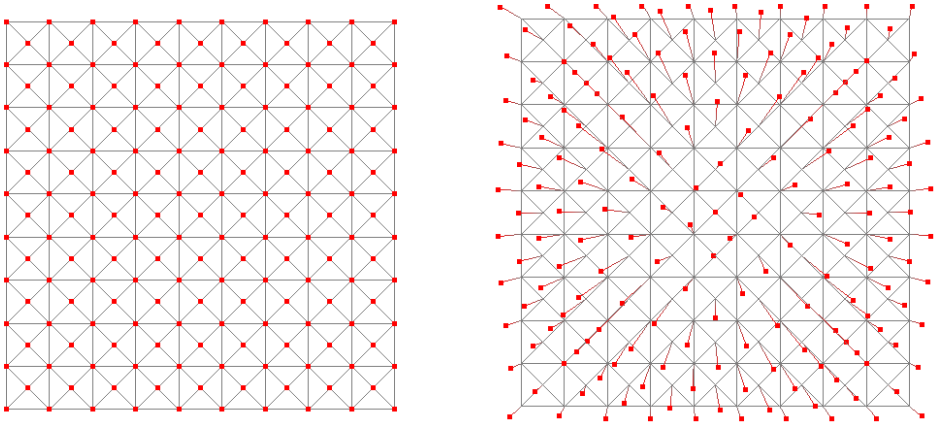
1: **for** each 2D data slice **do**
2: calculate slice z position
3: set grid vertices (x, y, z)
4: **for** each grid vertex **do**
5: $(x', y', z') \leftarrow$**warp**(x, y, z) {find position in source}
6: $(s', t', u') \leftarrow$ texture coordinates for (x', y', z')
7: store grid vertex data (x, y, z, s', t', u')
8: **end for**
9: render slice using (x, y, z, s', t', u') textured grid
10: read rendered data from graphics memory
11: replace 2D slice voxel values with rendered data
12: **end for**

3.13.2 Algorithm implementation

Algorithm 3 shows the pseudocode for this implementation. The target 3D data set is processed slice by slice rather than voxel by voxel and uses backward mapping. For each xy slice in the target image, the actual z position is calculated using the slice number and the voxel size (step 2), then the (x, y, z)

vertex positions of the grid mesh are populated.

The grid vertices are then warped to their corresponding positions (x', y', z') in the source image using Algorithm 1 in Section 3.10 on page 60. The 3D texture coordinates (s', t', u') representing that position in the source image are stored with the *unwarped* vertex position (steps 5 to 7). Note that although the grid vertices will *not* change position, the texture coordinates *will* change. Figure 3.13 shows this diagrammatically – compare with Figure 3.12 which shows the vertices themselves being warped, but in the opposite direction.



(a) Unwarped grid texture coordinates. (b) Warped grid texture coordinates.

Figure 3.13: Regular grid, triangulated for better interpolation, demonstrating warping of texture coordinates (shown in red).

In step 9, the textured grid is rendered to an offscreen buffer – actually a block of texture memory on the graphics card – and then, in steps 10 to 11, the warped texture is used to populate the target voxels covered by the 2D slice. The process is repeated until all slices have been rendered.

3.14 Software accelerated "Fast RBF" evaluation

In 2006, Livne and Wright [100, 101] described a new method for fast multilevel evaluation of RBF expansions. The method is compared to the *Fast Multipole*

Method (FMM) of Beatson and Newsam [10] and the *Fast Gauss Transform* (FGT) of Roussos and Baxter [140] and claims advantages in simplicity of implementation along with ease of parallelisation.

The algorithmic cost of Livne and Wright's technique for smooth RBFs is given as $O((m+n)(\ln(1/\delta))^d)$ where n is the number of *centres* (landmarks in this case), m is the number of evaluation points (voxels in this case), δ is the desired evaluation accuracy and d is the dimensionality of the data. In comparison, the cost of the brute force evaluation is $O(mn)$, the FMM cost is $O((m+n)(\ln n)(\ln(1/\delta))^{d+1})$, and the FGT cost is $O(q(c_1^d n + c_2^d m))$ where q is the number of Gaussians, $c_1, c_2 \in \mathbb{N}$, $c_1 > c_2$, and the constants increase with dimension and desired accuracy.

The basic idea of this method is that a smooth RBF, ϕ can be represented accurately on a regular coarse grid, of fewer nodes than the full voxel set and that the expensive summation in Equation 3.18 need be performed only at these nodes while the remaining voxel values can eventually be determined using a less expensive formulation based on the values calculated for the surrounding nodes.

Unlike the grid based approach in Section 3.13, it is the TPS coefficients that are interpolated within the grid, not the intensity values of the voxels.

3.14.1 "Fast RBF" evaluation in three dimensions

Before we describe the implementation of this method, we must first extend the formulation to three dimensions.

Following the notation used by Livne and Wright [101], we will regard the image data to be at "level h", where h is a basic measure of voxel/point distribution. In the following equations, all quantities defined at level h are denoted by lowercase symbols and all boldface symbols are in $\mathbb{R}^3$.

Without loss of generality, we assume that the centres (landmark points)

$\{\mathbf{y}_j\}_{j=0}^n$ and the evaluation points (target voxels) $\{\mathbf{x}_i\}_{i=0}^m$ lie in $[0,1]^3$ and that $\mathbf{y}_j = (y_j^{(1)}, y_j^{(2)}, y_j^{(3)})$ and $\mathbf{x}_i = (x_i^{(1)}, x_i^{(2)}, x_i^{(3)})$.

As in [101], the task is to compute :

$$s(\mathbf{x}_i) = \sum_{j=0}^{n} \lambda(\mathbf{y}_j)\phi(\|\mathbf{x}_i - \mathbf{y}_j\|), \qquad i = 0, 1, \ldots, m. \tag{3.31}$$

Level H

We define the "level **H**", $\mathbf{H} = (H, H, H)$ which is a coarser level than level h and consists of two uniform grids, each with spacing H :

$$\left\{Y_{J_k}^{(k)}\right\}_{J_k=0}^{N_k}, \qquad \left\{X_{I_k}^{(k)}\right\}_{I_k=0}^{M_k}, \qquad k = 1, 2, 3$$

The first of these grids is used to hold coarse level expansion coefficients at level **H**, which we will approximate based on the TPS coefficients $\lambda(\mathbf{y}_j)$ at each landmark. The second grid represents the level **H** evaluation points, which hold the coarse summations from which we will finally calculate the warped position of each voxel at level h. We will see how these grids are populated shortly.

To provide a convenient way to reference the level **H** grids, we will use the following notation, where $\times$ denotes the Cartesian product :

$$\{\mathbf{Y_J}\}_{\mathbf{J}=0}^{\mathbf{N}} = \left\{Y_{J_1}^{(1)}\right\}_{J_1=0}^{N_1} \times \left\{Y_{J_2}^{(2)}\right\}_{J_2=0}^{N_2} \times \left\{Y_{J_3}^{(3)}\right\}_{J_3=0}^{N_3}, \qquad \begin{array}{l}\mathbf{J} = (J_1, J_2, J_3),\\ \mathbf{N} = (N_1, N_2, N_3)\end{array}$$

$$\{\mathbf{X_I}\}_{\mathbf{I}=0}^{\mathbf{M}} = \left\{X_{I_1}^{(1)}\right\}_{I_1=0}^{M_1} \times \left\{X_{I_2}^{(2)}\right\}_{I_2=0}^{M_2} \times \left\{X_{I_3}^{(3)}\right\}_{I_3=0}^{M_3}, \qquad \begin{array}{l}\mathbf{I} = (I_1, I_2, I_3),\\ \mathbf{M} = (M_1, M_2, M_3)\end{array}$$

So, for example, $\mathbf{Y}_{(J_1,J_2,J_3)}$ represents the level **H** expansion coefficient $(Y_{J_1}^{(1)}, Y_{J_2}^{(2)}, Y_{J_3}^{(3)})$ and $\mathbf{X}_{(I_1,I_2,I_3)}$ represents the level **H** coarse summation point $(X_{I_1}^{(1)}, X_{I_2}^{(2)}, X_{I_3}^{(3)})$.

The grids $\{\mathbf{Y_J}\}_{\mathbf{J}=0}^{\mathbf{N}}$ and $\{\mathbf{X_I}\}_{\mathbf{I}=0}^{\mathbf{M}}$ are created in such a way that they just overlap $\{\mathbf{y}_j\}_{j=0}^n$ and $\{\mathbf{x}_i\}_{i=0}^m$ respectively. This is so that we can use centered p-th order tensor product interpolation between the grids at level **H** and the

corresponding $\mathbf{y}_j$ and $\mathbf{x}_i$ at level h, for some even[10] $p \in \mathbb{N}$. Specifically, for $k = 1, 2, 3$, we choose [101] :

$$\begin{aligned} Y_J^{(k)} &= y_{\min}^{(k)} - \tfrac{(p-1)H}{2} + J^{(k)}H, \quad J^{(k)} = 0, 1, \ldots, N_k, \\ X_I^{(k)} &= x_{\min}^{(k)} - \tfrac{(p-1)H}{2} + I^{(k)}H, \quad I^{(k)} = 0, 1, \ldots, M_k \end{aligned}$$

where

$$\begin{aligned} N_k &= \left\lfloor \tfrac{y_{\max}^{(k)} - y_{\min}^{(k)}}{H} - 0.5 \right\rfloor + p, \quad y_{\min}^{(k)} = \min_{0 \le j \le n} y_j^{(k)}, \quad y_{\max}^{(k)} = \max_{0 \le j \le n} y_j^{(k)} \\ M_k &= \left\lfloor \tfrac{x_{\max}^{(k)} - x_{\min}^{(k)}}{H} - 0.5 \right\rfloor + p, \quad x_{\min}^{(k)} = \min_{0 \le i \le m} x_i^{(k)}, \quad x_{\max}^{(k)} = \max_{0 \le i \le m} x_i^{(k)} \end{aligned}$$

Having defined the grids at level $\mathbf{H}$, we can proceed to implement a less expensive summation than that shown in Equation 3.31. The first step to this is to calculate the level $\mathbf{H}$ expansion coefficients.

Calculating the "level H" expansion coefficients

As described in [101], because $\phi(\|\mathbf{x}_i - \mathbf{y}\|)$ is a smooth function of $y^{(1)}, y^{(2)}, y^{(3)}$ for every fixed $\mathbf{x}_j$, we can approximate its value at $\mathbf{y} = \mathbf{y}_j$ from its values at neighbouring $\mathbf{Y}_J$'s by using a centered p-th order tensor product interpolation in $y^{(1)}$, $y^{(2)}$ and $y^{(3)}$:

$$\phi(\|\mathbf{x}_i - \mathbf{y}_j\|) = \sum_{j : J_k \epsilon \sigma_j^{(k)}} \omega_{jJ_3} \omega_{jJ_2} \omega_{jJ_1} \phi(\|\mathbf{x}_i - \mathbf{Y}_{(J_1, J_2, J_3)}\|) \tag{3.32}$$

where $j = 0, 1, \ldots, n$ and for $k = 1, 2, 3 : \sigma_j^{(k)} := \left\{ J_k : |Y_{J_k}^{(k)} - y_j^{(k)}| < pH/2 \right\}$, ω_{jJ_k} are the centered pth-order interpolation weights from the coarse centres $Y_{J_k}^{(k)}$ to the landmark positions $y_j^{(k)}$. Using the method described in [101, Appendix A], the interpolation weights $\{\omega_j\}_{j=0}^{p-1}$ are calculated using the following algorithm :

$$\begin{aligned} \tilde{\omega}_j &\longleftarrow c_j/(x - x_j), \qquad j = 0, \ldots, p-1 \\ \tilde{\omega} &\longleftarrow \sum_{j=0}^{p-1} \tilde{\omega}_j \\ \omega_j &\longleftarrow \tilde{\omega}_j / \tilde{\omega}, \qquad j = 0, \ldots, p-1 \end{aligned}$$

[10] p must be even for a *centered* interpolation.

where

$$c_j = \left(\prod_{i \neq j} (x_j - x_i) \right)^{-1}, \qquad j = 0, 1, \ldots, p-1$$

Now we can perform an *anterpolation* of the TPS coefficients $\{\lambda(\mathbf{y}_j)\}_{j=0}^{n}$ to level $\mathbf{H}$ by substituting Equation 3.32 into Equation 3.31 :

$$s(\mathbf{x}_i) = \sum_{J_3=0}^{N_3} \sum_{J_2=0}^{N_2} \sum_{J_1=0}^{N_1} \Lambda\left(\mathbf{Y}_{(J_1,J_2,J_3)}\right) \phi\left(\|\mathbf{x}_i - \mathbf{Y}_{(J_1,J_2,J_3)}\|\right) \tag{3.33}$$

where $i = 0, 1, \ldots, m$ and [190] :

$$\Lambda(\mathbf{Y}_{(J_1,J_2,J_3)}) = \sum_{j:J_k \epsilon \sigma_j^{(k)}} \omega_{jJ_3} \omega_{jJ_2} \omega_{jJ_1} \lambda(\mathbf{y}_j) \qquad k = 1, 2, 3. \tag{3.34}$$

in which, $\lambda(\mathbf{y}_j)$ is the vector of TPS coefficients (3D weights) at landmark position $\mathbf{y}_j$ and $J_1 = 0, 1, \ldots, N_1$, $J_2 = 0, 1, \ldots, N_2$ and $J_3 = 0, 1, \ldots, N_3$.

Livne and Wright [101] implement Equation 3.34 by first initialising all of the Λ's to zero, and then by looping through all of the landmarks, distribute each TPS coefficient $\lambda(\mathbf{y}_j)$ amongst the p^2 neighbouring $\mathbf{Y_J}$'s. Performing the anterpolation in this way requires $O(p^d n)$ operations. Our implementation – described in detail in Section 3.14.2 – will employ the same technique.

Performing the "level H" summation

In the same way that we calculated the coarse level expansion coefficients, we can take advantage of the smoothness of $\phi(\|\mathbf{x} - \mathbf{Y_J}\|)$ as a function of $x^{(1)}, x^{(2)}, x^{(3)}$ for every fixed $\mathbf{Y_J}$ to obtain the following :

$$\phi(\|\mathbf{x}_i - \mathbf{Y_J}\|) = \sum_{I_k \epsilon \bar{\sigma}_i^{(k)}} \bar{\omega}_{iI_3} \bar{\omega}_{iI_2} \bar{\omega}_{iI_1} \phi(\|\mathbf{X}_{(I_1,I_2,I_3)} - \mathbf{Y_J}\|) \tag{3.35}$$

where $i = 0, 1, \ldots, m$ and for $k = 1, 2, 3 : \bar{\sigma}_i^{(k)} := \left\{ I_k : |X_{I_k}^{(k)} - x_i^{(k)}| < pH/2 \right\}$, $\bar{\omega}_{iI_k}$ are the centered pth-order interpolation weights from the coarse evaluation point $X_{I_k}^{(k)}$ to the level h evaluation point $x_i^{(k)}$.

Now, substituting Equation 3.35 into Equation 3.33 gives us [190] :

$$s(\mathbf{x}_i) = \sum_{I_k \epsilon \bar{\sigma}_i^{(k)}} \bar{\omega}_{iI_3} \bar{\omega}_{iI_2} \bar{\omega}_{iI_1} S(\mathbf{X}_{(I_1,I_2,I_3)}) \qquad k = 1, 2, 3. \tag{3.36}$$

where

$$S(\mathbf{X}_{(I_1,I_2,I_3)}) = \sum_{J_3=0}^{N_3} \sum_{J_2=0}^{N_2} \sum_{J_1=0}^{N_1} \Lambda(\mathbf{Y}_{(J_1,J_2,J_3)}) \phi(\|\mathbf{X}_{(I_1,I_2,I_3)} - \mathbf{Y}_{(J_1,J_2,J_3)}\|)$$
$$I_1 = 0, 1, \ldots, M_1, \quad I_2 = 0, 1, \ldots, M_2, \quad I_3 = 0, 1, \ldots, M_3 \tag{3.37}$$

So Equation 3.37 allows us to populate the grid $\left\{X_{I_k}^{(k)}\right\}_{I_k=0}^{M_k}$ from which we can finally interpolate to level h to perform the full RBF evaluation using Equation 3.36.

3.14.2 "Fast RBF" implementation

Implementation specifics

As we mentioned in Section 3.10.1, the method described by Livne and Wright [101] is developed for smooth kernels.

The TPS with RBF $\phi(r) = r$ is only C^0 continuous at $r = 0$, whilst $\phi(r) = r^2 \log r$ is C^1 continuous at $r = 0$, hence both functions are only piecewise smooth kernels. Therefore, the best approximation of a TPS using centered interpolation is linear, i.e. $p = 2$. Increasing p in the "Fast RBF" algorithm would create a worse approximation due to the Gibbs phenomenon [190].

For a more accurate approximation to a TPS, it would be necessary to employ the "kernel softening" technique described in [100], and to use a combination of larger p and H to balance the computational cost. This is a potential area for future research.

So for the above reasons, the implementation of the "Fast RBF" technique will use $p = 2$, which simplifies some of the equations in Section 3.14.1. We will describe these in the following few pages.

Implementation overview

In this implementation of the "Fast RBF" warp evaluation, the method proceeds as follows :

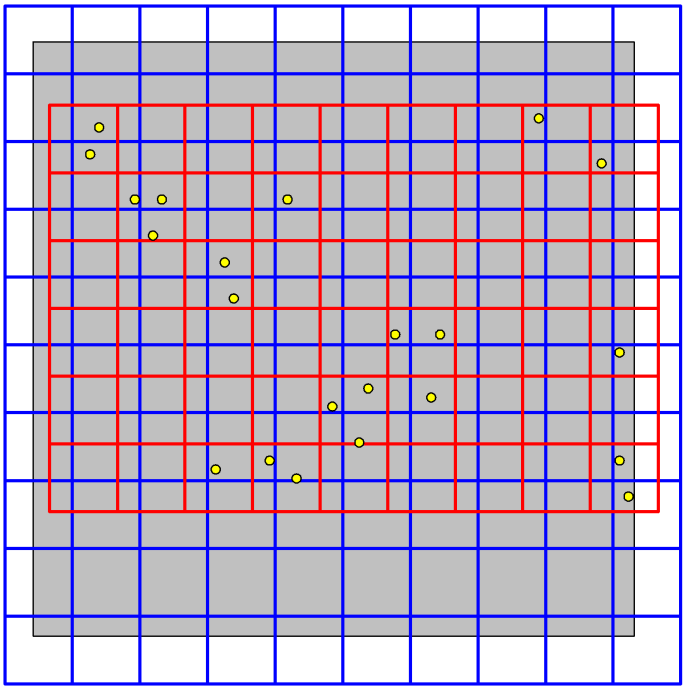


Figure 3.14: Example of **X** (blue) and **Y** (red) grids for "Fast RBF" in 2D – yellow points indicate landmarks.

1. The source and target images are normalised so that all voxel positions lie in $[0, 1]^3$.

2. Using normalised landmark positions, the TPS coefficients are calculated as described in Section 3.9.

3. Two uniform grids are defined with spacing H – the first grid, **Y**, covers the convex hull of the landmark positions[11] and extends just beyond such that each landmark lies *within* a grid cell. The second grid, **X**, similarly covers the range of target voxel positions, again ensuring a slight overlap.

[11] If backward mapping, we use the target points. If forward mapping, the source points.

See Figure 3.14.

4. For each landmark, the TPS coefficients are distributed to the surrounding nodes of grid $\mathbf{Y}$ by a process of anterpolation – see Section 3.14.3.

5. The nodes of grid $\mathbf{X}$ are populated using the coefficients from grid $\mathbf{Y}$ in the coarse level summation step – see Section 3.14.4.

6. All evaluation points (i.e. all voxels) are warped by using centered, second order, tensor product interpolation from the nodes of grid $\mathbf{X}$ – see Section 3.14.5.

The spacing H of grids $\mathbf{X}$ and $\mathbf{Y}$ is configurable and controls the accuracy and execution time of the warping function evaluation. A smaller value for H results in a more accurate result but lengthier run time. Experimental tests will use a range of values for H to examine these effects in more detail. This is discussed further in Chapters 4 and 5.

3.14.3 Anterpolation

Anterpolation (transpose of interpolation) is the mechanism used to distribute the known TPS coefficients at each landmark position to the surrounding nodes of grid $\mathbf{Y}$.

As described earlier in this section, this relies on the spatial smoothness of the RBF, $\phi(r)$, so that the value of the coefficients at any point $\mathbf{y} = \mathbf{y}_j$ within a grid cell can be approximated by a centered second-order tensor product interpolation from the values at the surrounding nodes of $\mathbf{Y}_J$.

By reversing the view of the above "smoothness" assumption, we take the known coefficients at each landmark and, using Equation 3.34, anterpolate by distributing them among the eight surrounding grid nodes according to the barycentric position of that landmark within the cell.

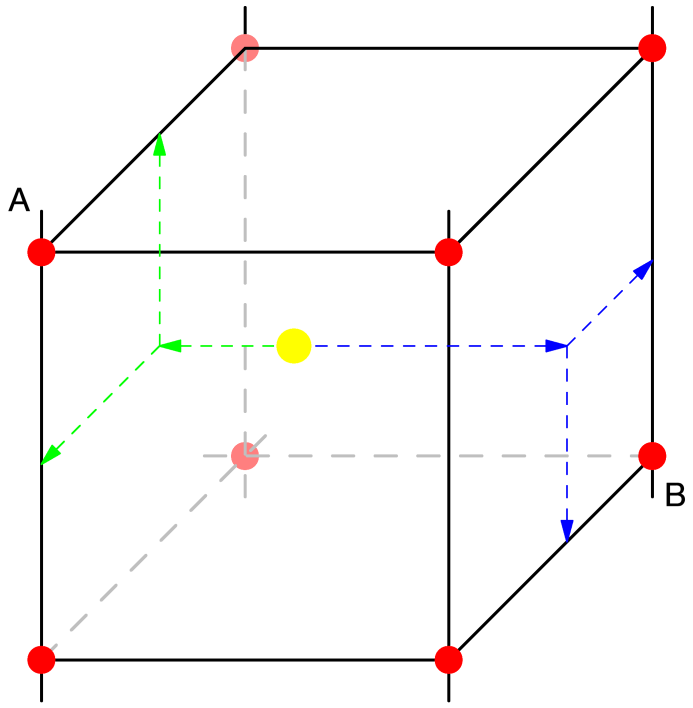


Figure 3.15: A single (yellow) landmark within grid $\mathbf{Y}$, showing the elements of interpolation weights for nodes A (green) and B (blue).

Figure 3.15 shows a single landmark[12] – shown in yellow – within a cell of grid $\mathbf{Y}$. The green arrows indicate the elements used to calculate the interpolation weights that apply to node A and the blue arrows show those for node B. As, by definition, the $\mathbf{Y}$ grid cell is a cube of dimension H and we are using $p = 2$, we just need to divide the (x, y, z) offset of the landmark position from the lower left front node by H to get the barycentric coordinates necessary to calculate the interpolation weights for each node.

So, by considering the cell in Figure 3.15 as a unit cube, the interpolation weight for node A is the product of the magnitude of the green vectors, and the interpolation weight for node B is the product of the magnitude of the blue vectors. The weights for the remaining nodes are determined similarly.

The anterpolation stage consists of the steps shown in Algorithm 4. First

[12]Depending on the size H of grid $\mathbf{Y}$, there may be more than one landmark in a cell.

the coarse expansion coefficients (the nodes of grid $\mathbf{Y}$) are initialised to zero. Then the TPS coefficients associated with each landmark are distributed to the surrounding nodes using the interpolation weights calculated as described above.

Algorithm 4 "Fast RBF" evaluation - anterpolation for $p = 2$.

$\Lambda(\mathbf{Y}) \leftarrow 0$ {initialise coarse expansion coefficients}
for each landmark (x, y, z) with coefficients $\lambda(\mathbf{y})$ **do**
  determine enclosing $\mathbf{Y}$ grid cell, $\mathbf{Y_J}$
  determine offset from cell origin $(\delta x, \delta y, \delta z)$
  $(\omega_1, \omega_2, \omega_3) \leftarrow \frac{(\delta x, \delta y, \delta z)}{H}$ {calculate barycentric positions}
  $(\omega_4, \omega_5, \omega_6) \leftarrow (1, 1, 1) - (\omega_1, \omega_2, \omega_3)$
  for each node $\mathbf{Y}_{Jn}$ in $\mathbf{Y_J}$, $\quad n = 1, \ldots, 8$ **do**
    determine which 3 of $\omega_1 \ldots \omega_6$ apply to this node
    $\omega \leftarrow \omega_a \omega_b \omega_c$ {combine weights}
    $\Lambda(\mathbf{Y}_{Jn}) \leftarrow \Lambda(\mathbf{Y}_{Jn}) + \omega\lambda(\mathbf{y})$
    mark node $\mathbf{Y}_{Jn}$ as 'valid'
  end for
end for

Once all landmarks have been processed, grid $\mathbf{Y}$ is complete and can be used to populated grid $\mathbf{X}$ in the coarse level summation stage.

3.14.4 Coarse level summation

Now that the TPS coefficients of the landmarks have been distributed to the nodes of the uniform grid $\mathbf{Y}$, we can use these to perform a coarse level summation to the nodes of the uniform grid $\mathbf{X}$, which encloses *all* of the evaluation points.

To do this, we use Equation 3.37, which shows the worst-case cost of the coarse level summation step. As detailed in [101], if the landmarks are well

distributed in the data, we could use the *same* grid for both $\mathbf{X}$ and $\mathbf{Y}$ and simplify this summation step considerably. In any case, we only need to consider the nodes of $\mathbf{Y}$ in which we have accumulated the coarse level expansion coefficients – that is, those nodes marked as 'valid' in step 11 of Algorithm 4.

Algorithm 5 shows the implementation of the coarse level summation stage. Each of the coarse level evaluation points in grid $\mathbf{X}$ is evaluated using the coarse level coefficients from grid $\mathbf{Y}$ rather than using the original landmarks.

Algorithm 5 "Fast RBF" evaluation - coarse level summation stage.

1: $S(\mathbf{X}) \leftarrow 0$
2: **for** each node $\mathbf{X_I}$ in $\mathbf{X}$ **do**
3: **for** each valid $\mathbf{Y_J}$ with coefficients $\Lambda(\mathbf{Y_J})$ **do**
4: determine distance r between $\mathbf{Y_J}$ and $\mathbf{X_I}$
5: $u \leftarrow \phi(r)$ {evaluate basis function}
6: $S(\mathbf{X_I}) \leftarrow u\Lambda(\mathbf{Y_J})$
7: **end for**
8: **end for**

Once this stage is complete, the final step is to process all of the evaluation points (voxels) by interpolating from the nodes in grid $\mathbf{X}$.

3.14.5 Interpolation - completing the warp evaluation

The last step in this method is to evaluate the warp for each of the voxels in the target data set so that we can sample the intensity value from the corresponding point in the source data and so complete the warp.

To do this, we use Equation 3.36 to take the coarse evaluation values at each node of each cell of grid $\mathbf{X}$ and use the same method of interpolation.

Algorithm 6 shows the pseudocode for this process. Each target voxel is matched to its enclosing cell from grid $\mathbf{X}$. We then perform a centered second-order tensor product interpolation from the coarse evaluation points in the

Algorithm 6 "Fast RBF" evaluation - interpolation stage.

1: **for** each target voxel $V_t(x, y, z)$ **do**
2: $(x_t, y_t, z_t) \leftarrow V_t \times (voxelsize)$ {calculate target voxel position}
3: determine enclosing $\mathbf{X}$ grid cell, $\mathbf{X_I}$
4: $(x_s, y_s, z_s) \leftarrow 0$
5: **for** each node $\mathbf{X}_{In}$ in $\mathbf{X_I}$ **do**
6: calculate interpolation weight, ω
7: $(x_s, y_s, z_s) \leftarrow (x_s, y_s, z_s) + \omega S(\mathbf{X}_{In})$
8: **end for**
9: $(x_s, y_s, z_s) \leftarrow (x_s, y_s, z_s) +$ T(x_t, y_t, z_t)
10: $V_t(x, y, z) \leftarrow V_s(x_s, y_s, z_s)$ {set new voxel value}
11: **end for**

surrounding eight nodes of $\mathbf{X}$.

Finally, we apply the affine part of the TPS transformation and sample the source data to retrieve the intensity value of the target voxel position. Once all voxels have been processed, the warp is complete.

3.15 Hardware accelerated "Fast RBF" evaluation

3.15.1 Overview

The final evaluation technique is a GPU-based implementation of the "Fast RBF" technique described in the previous section.

Like the GPU-based brute force technique in Section 3.12, this implementation utilises DirectX pixel shaders and graphics card texture memory as the method of "hardware" acceleration.

3.15.2 Algorithm implementation

Figure 3.16 on page 81 shows a schematic diagram of this process. The area between the dotted lines represents the processes taking place on the graphics hardware. The full source code of the shaders is listed in Appendix C.

The implementation begins in the same way as the software implementation discussed in Section 3.14.2. So, the following steps are identical and are all performed using system memory and the CPU :

1. The source and target images are normalised so that all voxel positions lie in $[0, 1]^3$.

2. Using normalised landmark positions, the TPS coefficients are calculated as described in Section 3.9.

3. The two uniform grids, $\mathbf{X}$ and $\mathbf{Y}$, are defined with spacing H.

4. For each landmark, the TPS coefficients are anterpolated to the surrounding nodes of grid $\mathbf{Y}$.

At this point we switch to the hardware accelerated phase of the method. The positions of the *valid*[13] nodes of grid $\mathbf{Y}$ are loaded to a 32-bit floating point texture on the graphics card. Similarly, the coarse expansion coefficients associated with each such node are loaded to a second floating point texture.

The first of two rendering passes is now made. The nodes of grid $\mathbf{X}$ are mapped 1:1 with pixels and the whole grid is rendered plane-by-plane. A pixel shader – marked as 'Pixel Shader 1' in Figure 3.16 – performs the coarse level summation step and writes the resulting values to an offscreen texture buffer on the graphics card[14]. This represents the coarse level evaluation points of grid $\mathbf{X}$.

[13] Refer to Algorithm 4 on page 77.

[14] Due to technical limitations of the graphics card, this is implemented as a 3D *flat* texture[59].

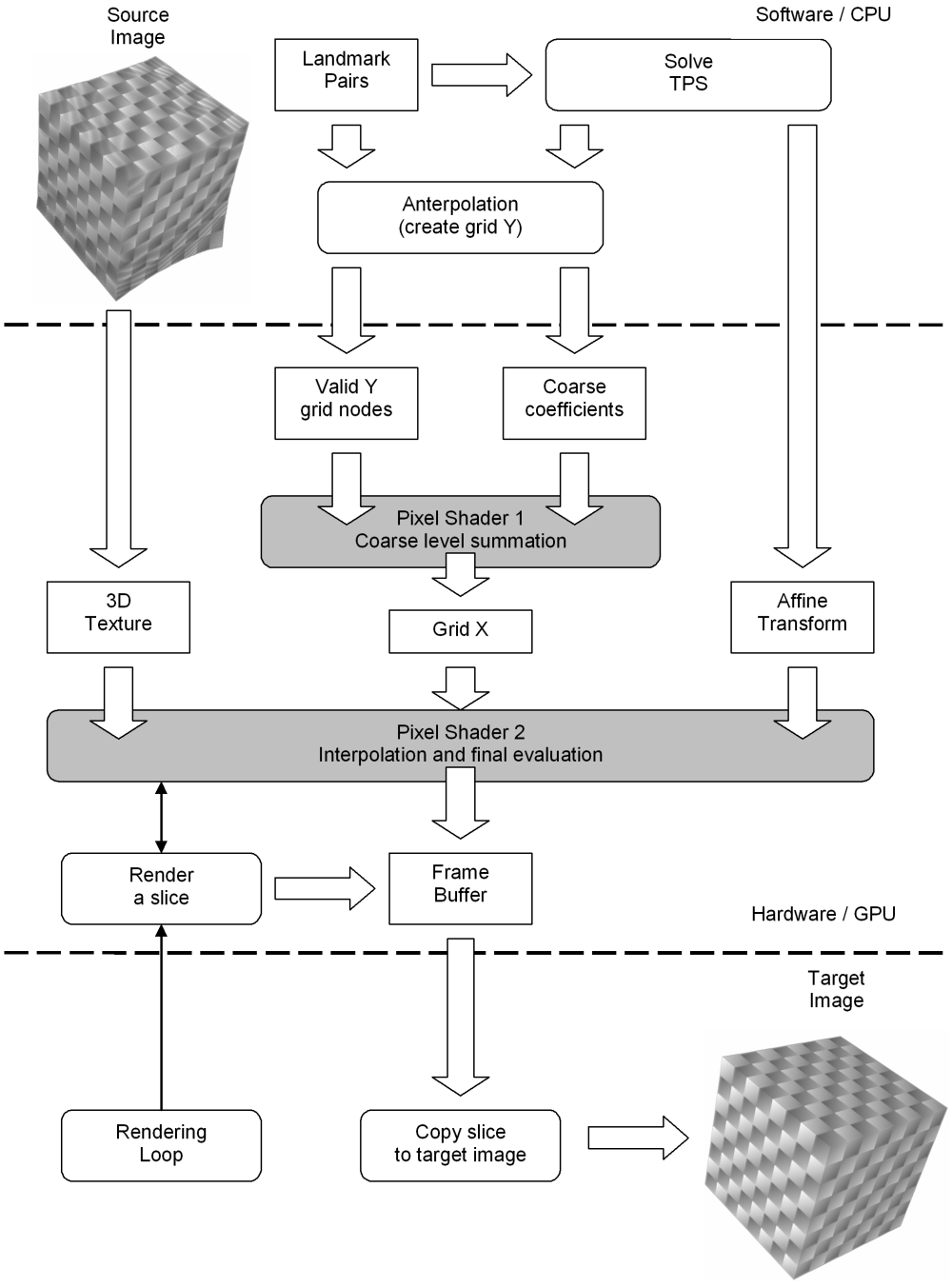


Figure 3.16: "Fast RBF" evaluation - hardware accelerated.

The second rendering pass is similar to the brute force method described earlier in Section 3.12 – the target data set is rendered slice by slice and the pixel shader – marked as 'Pixel Shader 2' in Figure 3.16 – performs the second-order

tensor product interpolation using grid $\mathbf{X}$, applies the affine transformation from the TPS stage, and finally samples the source texture to obtain the correct intensity value for the target voxel.

As each target slice is rendered to an offscreen buffer, it is read back into system memory and used to populate the target data set. Once all slices have been rendered, the warp evaluation is complete.

3.16 Test plan

The primary aim of this study is to determine whether or not we can significantly improve the run time of a warping function evaluation without compromising the accuracy to an extent which renders the evaluation useless for any particular application.

How one defines acceptable accuracy is inevitably application dependent and/or subjective. However, we can at least attempt to quantify the effects of performance increase on accuracy to allow such judgements to be made.

The first part of the test plan is to establish a *gold standard* - that is, the absolute best we can expect to achieve in terms of warping accuracy. This will also give us a benchmark timing, which is what we will target for improvement.

For the reasons given in Sections 3.10.1 and 3.11, the brute force software implementation is the logical choice for optimal accuracy. However, we must also determine the best choices for the other experimental variables we noted – the option of tri-linear interpolation versus nearest-neighbour sampling, and the choice of RBF. Therefore, the first set of tests will be as follows

Test 1 Using the brute force software implementation, warp all eight synthetic data sets – four using $\phi(r) = r^2 \log r$ and four using $\phi(r) = r$ – once using tri-linear interpolation during backward mapping and again using nearest neighbour sampling.

This will allow us to do the following :

- Compare accuracy and performance for interpolation option.
- Compare accuracy and performance for RBF option.
- Compare similarity metrics for different test data.

Having performed this test, we wish to be able to choose the most suitable combination of interpolation option, test data set and similarity metric for making comparisons of performance[15] between the different evaluation techniques. Assuming we can do so, we will then run through the remaining evaluation techniques as follows

Test 2 Using the brute force hardware implementation, warp all eight synthetic data sets – four using $\phi(r) = r^2 \log r$ and four using $\phi(r) = r$.

In a way, this provides us with a second benchmark. The main difference between this implementation and its software equivalent is the floating point precision. The results of this set of tests will allow us to assess a reasonable standard for the best – in terms of warping accuracy – that we can achieve in a hardware implementation[16].

Test 3 Using the grid-based implementation, warp all eight synthetic data sets – four using $\phi(r) = r^2 \log r$ and four using $\phi(r) = r$. Repeat the tests using a range of different grid sizes.

This set of tests will allow us to compare the overall performance[17] of the grid-based method with the results reported by the original authors [95, 96], and will also show the effect that the grid size has on performance. By using

[15] We will continue to assess the performance of *both* RBFs to avoid trivialising the optimised implementations.

[16] Future hardware advances will almost certainly remove this distinction in due course.

[17] As we are using different data, we can only compare speed, not registration accuracy.

all of the synthetic data sets, we may also be able to determine whether or not the premise in Section 3.6 is correct – that is, are some test data sets more amenable to evaluation techniques that employ linear interpolation?

Test 4 Using the software "Fast RBF" implementation, warp all eight synthetic data sets – four using $\phi(r) = r^2 \log r$ and four using $\phi(r) = r$. Repeat the tests using a range of different grid sizes – that is, varying the parameter H in the algorithm (Section 3.14.2).

This will allow us to compare the overall performance and accuracy of the software "Fast RBF" method and also show the effect that the parameter H has on performance. Again, by using all of the synthetic data sets, we may be able to determine whether this technique is sensitive to the choice of test data.

Test 5 Using the hardware "Fast RBF" implementation, warp all eight synthetic data sets – four using $\phi(r) = r^2 \log r$ and four using $\phi(r) = r$. Repeat the tests using a range of different grid sizes – again, varying the parameter H in the algorithm.

As expected, this will allow us to compare the overall performance and accuracy of the hardware "Fast RBF" method and show the effect that the parameter H has on performance.

All of the above tests must of necessity use the same inputs. However, it may be that one or more of the evaluation techniques is more dependent on the inputs than others. The most obvious choice is the number of landmarks, which we address in the following tests :

Test 6 Repeat all of the previous tests, taking timings only, with varying number of landmark pairs as inputs. These tests should include all grid sizes previously used for the grid-based technique and all values of H previously used for the "Fast RBF" techniques.

The above test will highlight how the execution time of each algorithm varies with different numbers of landmarks. These tests will use multiple sets of landmarks, from 10 to 500 in steps of 10, and will allow us to examine both the warping time and the time necessary to solve the TPS transformation.

Test 7 Using the benchmark evaluation technique and the fastest/most accurate optimised technique, warp and merge CT and MRI data sets from the Visible Human Project and compare the results visually as well as by similarity metrics.

The final test takes the competing techniques in terms of accuracy and speed and uses these on real medical image data to provide a subjective (visual) and objective (similarity metric) measurement of the suitability of the optimised technique for multi-modal non-rigid medical image registration in three dimensions.

3.17 Summary

In this chapter, we have discussed RBFs in the context of non-rigid registration and described the design of synthetic test data and a number of image similarity metrics before finally detailing the experimental software and the algorithmic and implementation changes we have made to accelerate performance.

Finally, we have devised a test plan, the results of which will answer, among others, the following crucial questions :

1. How much can we optimise the run time of an RBF based warp evaluation?
2. What effect do such optimisations have on warping accuracy?
3. How do the optimised techniques behave under different inputs?
4. Are the optimised methods accurate and/or fast enough for AR applications?

To answer these questions, we have done the following :

1. Designed and implemented an experimental software application to control all testing and to record results.

2. Designed and built synthetic test data sets, which will allow us to evaluate the accuracy of a warp and the suitability of different image similarity metrics.

3. Implemented brute force warp evaluation techniques in both software and hardware versions.

4. Implemented the grid-based warping technique of [95, 96] to compare with the new optimised implementations.

5. Extended the formulation of [101] to three dimensions and implemented in both software and hardware versions.

6. Performed a structured test plan to exercise all algorithms under a range of inputs and parameters.

Chapter 4 gives details of the results of the tests described in Section 3.16 and Chapters 5 and 6 discuss these results and how they have (or have not) answered the above questions.

Chapter 4

Experimental results

This chapter presents the results of the experimental testing, using both synthetic data and real CT and MRI data sets from the Visible Human Project [1].

Results are given in both tabular and graphical formats and in alternative views, for example run times and registration accuracy are presented individually as well as a combined view which includes both aspects.

In this chapter, analysis and discussion is kept as concise as possible. Chapter 5 provides a discussion of particular issues raised by these results and Chapter 6 shows how we have answered the questions posed in Chapter 2.

As the use of purpose-designed synthetic test data allows a much more rigorous comparison of registration accuracy than real-world data, the following section provides the bulk of the material for discussion in Chapter 5. Section 4.2 shows sample images of merged CT and MRI data following registration of the two data sets.

4.1 Synthetic data tests

As mentioned in Chapter 3, we have devised four sets of *ground truth* test data, each of which has been warped twice – once for each RBF[1] – using a given set

[1] $\phi(r) = r$ and $\phi(r) = r^2 \log r$.

of landmarks. So we have eight synthetic data sets. Each algorithm is then exercised to warp each set *back* to its unwarped equivalent (the ground truth). As discussed in Section 3.6, we are using the same landmark pairs[2] we used to create the warped test data to ensure that we do not introduce any irregularities due to misplaced landmarks.

This situation should give us the best possible result. With no landmark placement errors and known ground truth data, we *should* be able to produce a perfect warp of each test data set back to the ground truth.

However, this is not possible in practice as the warping function is not lossless – see Section 3.9.1 – and there may also be slight numerical differences when solving the TPS in the reverse direction. In any case, as we first noted in Section 3.16, the *gold standard* against which we measure performance of different implementations of the warp evaluation must be the full voxel-by-voxel evaluation, performed in software – this is the most direct implementation of the warping function (Equation 3.18), with no performance optimisations and using double precision floating point arithmetic throughout.

The following sections give selected timings and image similarity metrics for each of the five evaluation techniques, beginning with the gold standard. Sections are ordered according to the test plan from Section 3.16. Full listings of *all* experimental results are given in Appendix B.

4.1.1 Test 1 - Brute force evaluation in software

As mentioned in the previous section, the first test is intended to set the standard for all remaining tests. The brute force software implementation is theoretically the most accurate, being effectively a reversal of the method used to prepare the test data sets.

However, we still need to determine the effect of the interpolation option

[2]Each landmark pair is swapped to effectively *unwarp* the warped test data.

used during warping (tri-linear interpolation versus nearest neighbour) and compare performance of the chosen RBFs.

Overview

Table 4.1 shows the warping evaluation time, in seconds, and the Normalised Mutual Information metric (NMI) for each combination of test data, RBF and interpolation option[3].

Data set	RBF	Interp.	Warp time	NMI
Regular linear gradient	$r^2 \log r$	TRI	790.143372	1.716064
Irregular linear gradient	$r^2 \log r$	TRI	789.893372	1.550476
Checkered cube + noise	$r^2 \log r$	TRI	791.614441	1.265181
Non-linear gradient	$r^2 \log r$	TRI	790.550395	1.688309
Regular linear gradient	$r^2 \log r$	NNI	788.178650	1.589758
Irregular linear gradient	$r^2 \log r$	NNI	786.234314	1.495663
Checkered cube + noise	$r^2 \log r$	NNI	786.240051	1.278012
Non-linear gradient	$r^2 \log r$	NNI	786.884338	1.593237
Regular linear gradient	r	TRI	331.373566	1.717397
Irregular linear gradient	r	TRI	333.835164	1.553860
Checkered cube + noise	r	TRI	331.353333	1.264702
Non-linear gradient	r	TRI	332.187354	1.687073
Regular linear gradient	r	NNI	327.192413	1.585659
Irregular linear gradient	r	NNI	327.035370	1.492595
Checkered cube + noise	r	NNI	327.078644	1.277094
Non-linear gradient	r	NNI	327.102142	1.590584

Table 4.1: Brute force evaluation in software - performance and accuracy.

Test data sets are all $256 \times 256 \times 256$ voxels, each voxel is 8 bits. The same 729 landmark pairs were used for each run, and all tests were performed on the same physical computer system.

[3]TRI = Tri-linear interpolation, NNI = Nearest neighbour interpolation.

Note that each of the four sections in Table 4.1 represents the same algorithm performing the same amount of work. Only the content of the test data differs. While the run times are all consistent for each section, there is more variability in the NMI metric – this is discussed further in Section 5.1 of Chapter 5.

Performance versus accuracy

As one might intuitively expect, in most cases, using tri-linear interpolation during backward mapping tends to increase the accuracy at the expense of extending the warping time.

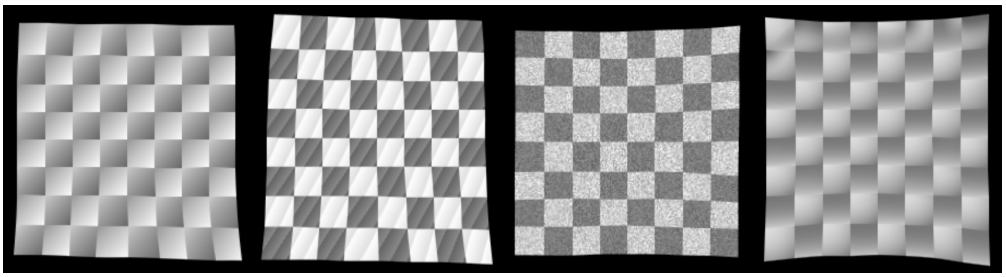
However, note the NMI scores in Table 4.1 for the test data with random noise – the figures imply that tri-linear interpolation during backward mapping gives a slightly worse result than using nearest-neighbour interpolation. Possible reasons for this are discussed in Section 5.2 of Chapter 5.

Backward mapping using tri-linear interpolation

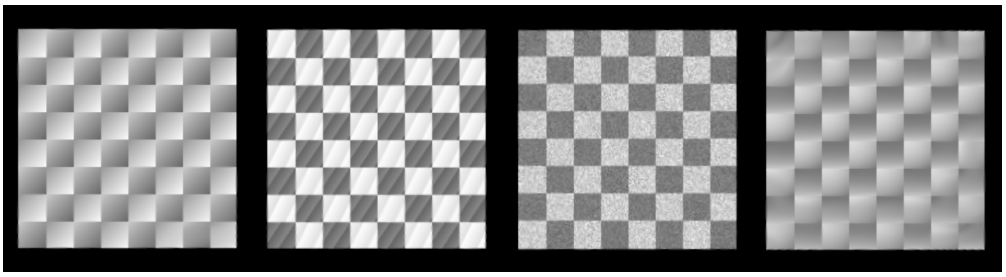
Figure 4.1 shows examples of some typical slices of the four $\phi(r) = r^2 \log r$ synthetic data sets, before and after warping, together with difference images of the same slices compared with the known ground truth. All of these images are from the brute force software evaluation with tri-linear interpolation during backward mapping.

Note that the difference images in Figure 4.1c have been inverted and gamma adjusted for clarity ($\gamma = 0.5$) – the actual differences in voxel intensities are too faint to be seen clearly when printed unenhanced.

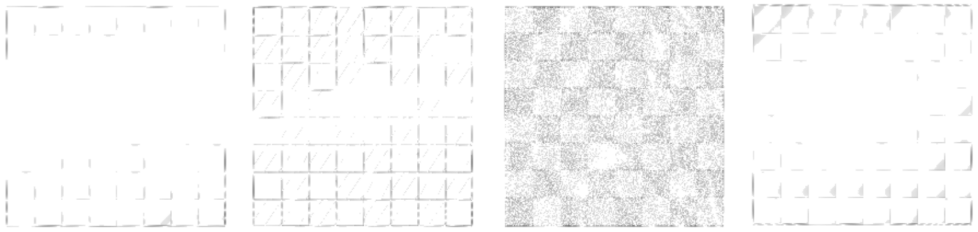
From left to right, Figure 4.1 shows slice $X = 128$ from the regular gradient data, slice $X = 30$ from the irregular gradient data, slice $Y = 55$ from the random noise data and finally slice $Z = 128$ from the non-linear gradient data. Note that the rightmost image represents a slice at the boundary of the checkered pattern, which is why some parts of the image appear to switch from



(a) Slices from $\phi(r) = r^2 \log r$ warped data sets - from left: regular gradient, irregular gradient, random noise, non-linear gradient.



(b) After warping using brute force software evaluation with tri-linear interpolation.



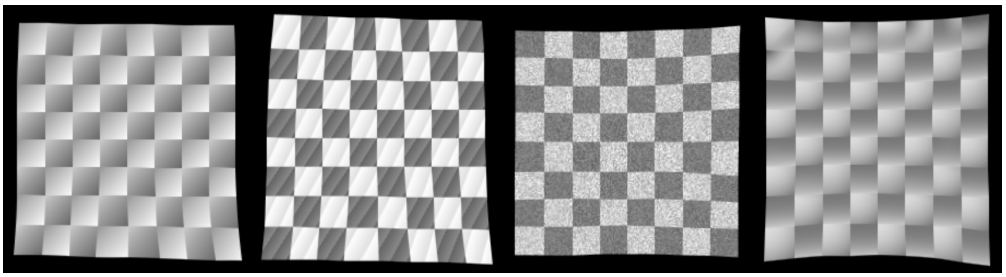
(c) Difference images, results versus ground truth (inverted and gamma adjusted for clarity, $\gamma = 0.5$).

Figure 4.1: Brute force software evaluation, with tri-linear interpolation, using RBF $\phi(r) = r^2 \log r$.

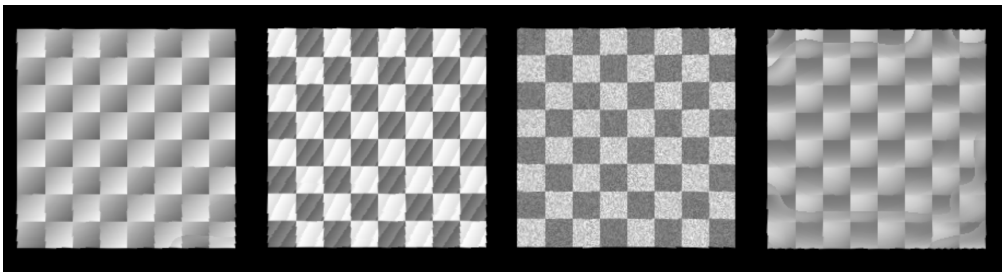
dark to light following registration.

Backward mapping using nearest-neighbour interpolation

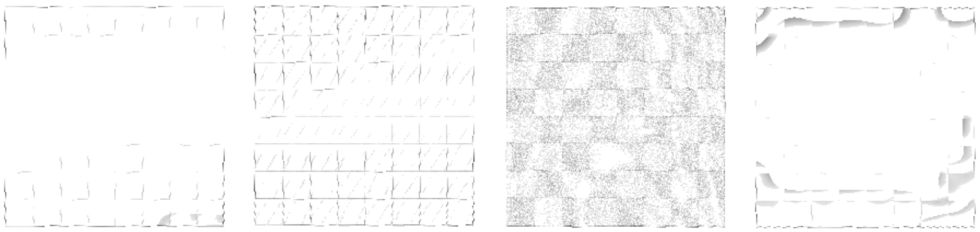
Figure 4.2 shows the same slice data as Figure 4.1, but this time the brute force evaluation algorithm was run using nearest-neighbour interpolation during backward mapping.



(a) Slices from $\phi(r) = r^2 \log r$ warped data sets - from left: regular gradient, irregular gradient, random noise, non-linear gradient.



(b) After warping using brute force software evaluation with nearest-neighbour sampling.



(c) Difference images, results versus ground truth (inverted and gamma adjusted for clarity, $\gamma = 0.5$).

Figure 4.2: Brute force software evaluation, with nearest-neighbour sampling, using RBF $\phi(r) = r^2 \log r$.

The results of the warp in Figure 4.2b, using nearest neighbour interpolation, and the difference images in Figure 4.2c show characteristic aliasing artifacts. Visually similar results are obtained from the $\phi(r) = r$ data sets, which are not shown here.

We can see in Table 4.1 that the registration accuracy of the nearest neighbour results compares poorly to tri-linear interpolation, hence the visual results

seem less appealing. However, it should be noted that the synthetic test data has been intentionally designed to include sharp discontinuities in voxel intensity and this will tend to expose aliasing artifacts more clearly.

Choice of radial basis function

Figures 4.1 and 4.2 both show data from the $\phi(r) = r^2 \log r$ tests. However, Table 4.1 suggests that the $\phi(r) = r$ tests produce results of comparable accuracy but with significantly shorter run times.

Figure 4.3 shows the equivalent of Figure 4.1 but using the $\phi(r) = r$ data sets and evaluation algorithms.

We can compare the visual results of the two RBF warp evaluations by looking at a 3D view of the difference image. Figure 4.4 shows a view of the difference image between the $\phi(r) = r^2 \log r$ and $\phi(r) = r$ results, using the regular linear gradient test data and tri-linear interpolation, which shows that the two RBF implementations produce visibly similar results.

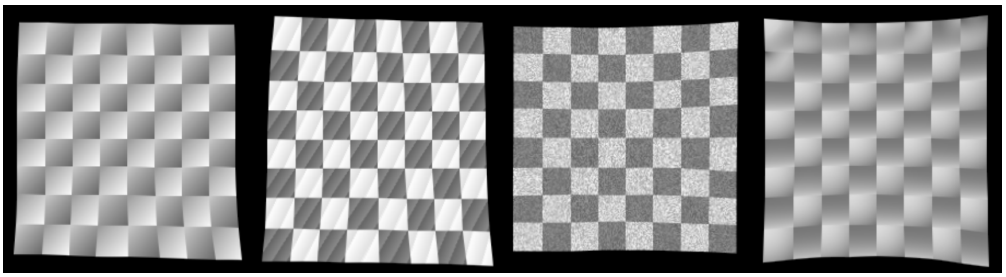
The "gold standard"

What we should note from the results of the brute force software evaluation is that the *best* combination of performance and accuracy[4] is achieved by using tri-linear interpolation during backward mapping. Therefore, this is what we shall use as the gold standard for comparing the performance of the optimised evaluation techniques.

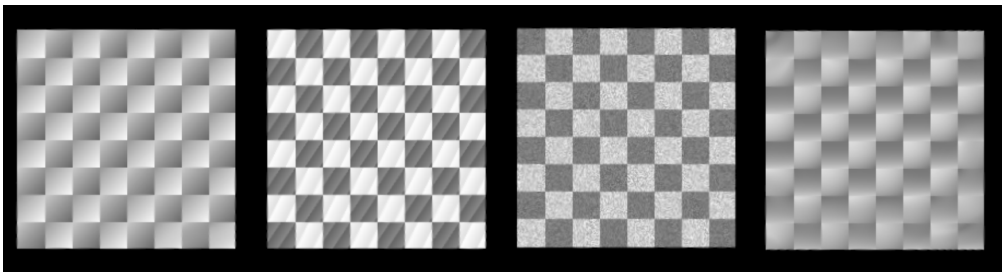
We also note that the run times for $\phi(r) = r$ are significantly faster than $\phi(r) = r^2 \log r$ and, according to Rohr [138, p. 195], the former is the correct TPS basis function for 3D data[5]. However, we will continue to monitor accuracy and run time performance for *both* basis functions to see how the optimised techniques are affected.

[4] Determined here by the run time and NMI metric in Table 4.1.

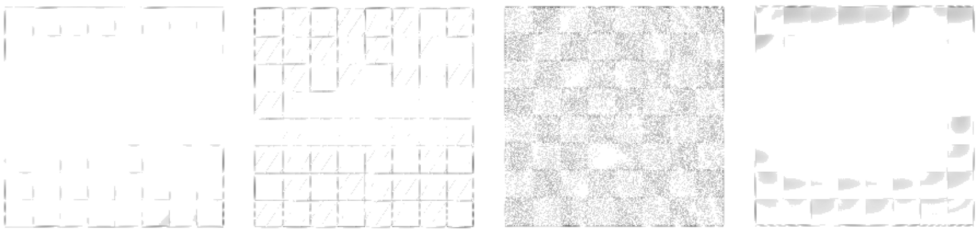
[5] For second order energy potentials.



(a) Slices from $\phi(r) = r$ warped data sets - from left: regular gradient, irregular gradient, random noise, non-linear gradient.



(b) After warping using brute force software evaluation with tri-linear interpolation.



(c) Difference images, results versus ground truth (inverted and gamma adjusted for clarity, $\gamma = 0.5$).

Figure 4.3: Brute force software evaluation, with tri-linear interpolation, using RBF $\phi(r) = r$.

Table 4.2 gives a complete set of run times and NMI metrics for the $\phi(r) = r$ gold standard and Table 4.3 shows the same details for the $\phi(r) = r^2 \log r$ gold standard. The overall registration accuracy – measured against the optimal values in Table 3.3 – is also shown as a percentage figure to facilitate later comparison. Note that the maximum possible NMI score is 2.0 and the minimum is 1.0.

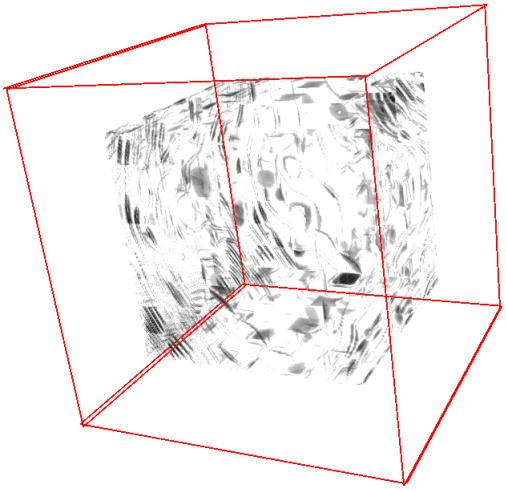


Figure 4.4: 3D difference image of $\phi(r) = r^2 \log r$ and $\phi(r) = r$ results(inverted and gamma adjusted for clarity, $\gamma = 0.5$).

Test data	Warp time	NMI	%Overall
Regular linear gradient	331.373566	1.717397	71.74
Irregular linear gradient	333.835164	1.553860	55.39
Checkered cube + noise	331.353333	1.264702	26.47
Non-linear gradient	332.187354	1.687073	68.71

Table 4.2: Gold standard for $\phi(r) = r$.

4.1.2 Test 2 - Brute force evaluation - hardware accelerated

As we mentioned in Section 3.16, the brute force hardware implementation using tri-linear interpolation could also be considered as an informal gold standard for hardware (GPU) based evaluation techniques. The main difference being the limitation of 32-bit floating point precision using this technique.

This section details the main results from Test 2, which show how the loss of numerical precision affects accuracy and how the use of the GPU-based implementation affects run times.

Test data	Warp time	NMI	%Overall
Regular linear gradient	790.143372	1.716064	71.61
Irregular linear gradient	789.893372	1.550476	55.05
Checkered cube + noise	791.614441	1.265181	26.52
Non-linear gradient	790.550395	1.688309	68.83

Table 4.3: Gold standard for $\phi(r) = r^2 \log r$.

Brute force hardware results

Table 4.4 shows a summary of the results of the brute force hardware tests. Comparing with the software gold standards in Tables 4.2 and 4.3, it is evident that accuracy is degraded although run time is improved.

Data set	RBF	Warp time	NMI	%Overall	%Gold
Regular linear gradient	r	12.887721	1.622725	62.27	86.81
Irregular linear gradient	r	13.285579	1.493154	49.32	89.03
Checkered cube + noise	r	12.905877	1.199441	19.94	75.35
Non-linear gradient	r	12.241180	1.526633	52.66	76.65
Regular linear gradient	$r^2 \log r$	12.908881	1.622828	62.28	86.99
Irregular linear gradient	$r^2 \log r$	12.903515	1.492722	49.27	89.52
Checkered cube + noise	$r^2 \log r$	12.844133	1.199271	19.93	75.14
Non-linear gradient	$r^2 \log r$	12.150575	1.526939	52.69	76.55

Table 4.4: Summary of hardware accelerated brute force results.

Looking at the average NMI scores for $\phi(r) = r$ we note that overall registration accuracy – measured against the optimal values in Table 3.1 – has decreased by approximately 9.5% compared to the gold standard (that is from 55.6% down to 46.0%) while the run time has decreased by a factor of just under 26 (down from 332.2 seconds to 12.8 seconds).

We can see a similar decrease in overall registration accuracy for $\phi(r) = r^2 \log r$ (down to 46.0% on average) but a much more marked effect on run times

compared to the gold standard (improved by a factor of 62.3 from 790.5 to 12.7 seconds). In fact, Table 4.4 shows that the average run times for $\phi(r) = r^2 \log r$ on this brute force hardware implementation are roughly equivalent to those for $\phi(r) = r$ (12.7 versus 12.8 seconds respectively). Section 5.3 on page 141 discusses possible reasons why the effect of a more costly RBF may not be so noticeable in GPU based warp evaluation techniques.

The rightmost column in Table 4.4 indicates the main effect of the loss of floating point precision. These figures suggest that the best we can expect from GPU based techniques is an average accuracy of 89% compared to the software gold standard when using the NMI metric for comparison. However, as we know the original spatial displacements for each voxel in the synthetic data sets, we can assess the accuracy in geometric terms. Table 4.5 shows the spatial error in millimetres of the brute force warp in hardware.

RBF	Minimum	Maximum	Range	Average	Std. Dev.
$\phi(r) = r$	0.000000	1.483833	1.483833	1.118361	0.793620
$\phi(r) = r^2 \log r$	0.000000	1.489594	1.489594	1.131088	0.096907

Table 4.5: Spatial errors in millimetres from the brute force hardware warp.

4.1.3 Test 3 - Grid based evaluation - hardware assisted

This test is intended to show two things. First, we would like to find the best possible accuracy that we can achieve by using this technique and compare this with the gold standard. Secondly, we want to see what effect the choice of grid size has on both accuracy and speed.

We already know that the use of the GPU for evaluation of the TPS transformation suffers from a loss of numerical precision and we noted the effect of this on warp accuracy in the preceding section. Whereas, in the original method of Levin et al. [95, 96], the TPS transformation was GPU based, in this

implementation we have specifically chosen to keep this part of the algorithm in *software* so that we can maximise the accuracy at the expense of run time.

The reason we made this choice is due to the fact that we are seeking the best *combination* of accuracy and performance and we do not yet have any information on the maximum possible *accuracy* of the technique (disregarding run time for now). In [96] and [95] we have analyses of run time performance which can guide us to an expectation of potential gains but these will almost certainly be at the expense of accuracy as we saw in Section 4.1.2 earlier.

Therefore, in the results which follow, it must be borne in mind that the run times of this technique *can* be improved but that the accuracy *cannot*. We further discuss the rationale behind these decisions in the context of experimental results in Section 5.4 on page 143.

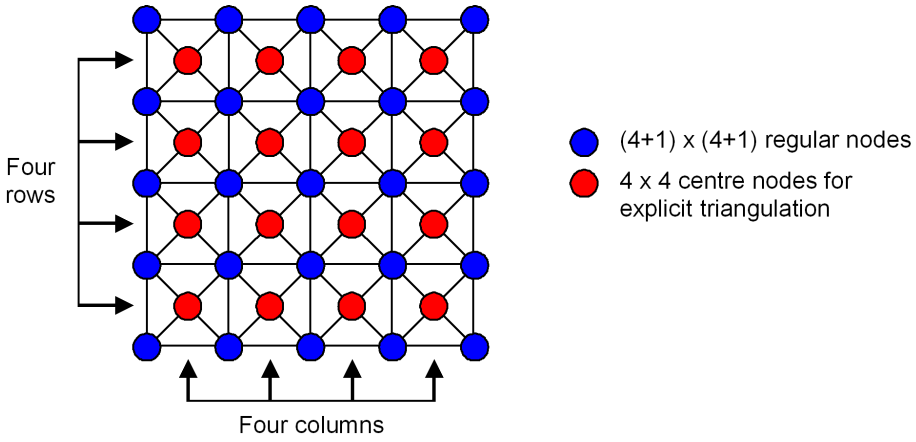


Figure 4.5: Diagram showing what is meant by 'a grid with a size of four'.

Overview

The synthetic test data sets are each 256^3 voxels so a grid size of 125 rows and 125 columns will result in cells which cover approximately 2×2 voxels. This is the smallest grid we can achieve which should give us some theoretical advantage over the brute force method. A grid of 255 rows by 255 columns is

equivalent to the brute force method as we are effectively transforming every voxel individually.

In all of the tests in this section the grids will be square – that is, they will have the same number of rows as columns. From now on, we will refer only to the number of rows, the column count will always be the same. Also, the rows and columns are counted as the spaces between the grid nodes, so a grid with four rows will have 5×5 'regular' grid nodes plus 4×4 additional nodes, which are used for explicit triangulation as described in Section 3.13. Figure 4.5 shows this schematically for a grid of size four. In general, a grid of size n will have $(n+1)^2 + n^2 = 2(n^2+n)+1$ nodes (vertices).

Data set	RBF	Warp time	NMI	%Overall	%Gold
Regular linear gradient	r	157.723965	1.621512	62.15	86.64
Irregular linear gradient	r	157.730304	1.486984	48.70	87.94
Checkered cube + noise	r	157.711837	1.217262	21.73	82.08
Non-linear gradient	r	155.340286	1.608784	60.88	88.61
Regular linear gradient	$r^2 \log r$	373.404022	1.623591	62.36	87.08
Irregular linear gradient	$r^2 \log r$	375.784742	1.487991	48.78	88.65
Checkered cube + noise	$r^2 \log r$	375.772021	1.217586	21.76	82.06
Non-linear gradient	$r^2 \log r$	375.802197	1.614749	61.47	89.31

Table 4.6: Summary of grid based evaluation using a grid of size 125.

Table 4.6 shows the warp times, NMI scores, percentage ratings for overall warp accuracy and as a percentage of the gold standard, for a grid of size 125.

Here we can see that we can achieve just over 86% accuracy on average, compared to the gold standard. Note that, for the synthetic data sets, we cannot use a grid size of more than 125 without necessitating sub-voxel sampling. However, the following section shows how varying the grid size, both higher and lower, affects the accuracy and run time of this method.

Effect of grid size

Table 4.7 shows the run times and the accuracy compared to the gold standard for various sizes of grid[6].

These figures are the averages of all four test data sets for each RBF. The timings and scores broken out by test data set are given in the full results in Appendix B. The full results show that the average figures in Table 4.7 are a fair representation of the experimental findings.

	$\phi(r) = r$			$\phi(r) = r^2 \log r$		
Grid size	**Warp time**	**%Overall**	**%Gold**	**Warp time**	**%Overall**	**%Gold**
13	3.681314	46.80	83.88	5.952039	46.99	84.28
25	8.145888	48.15	85.97	16.798895	48.40	86.48
38	16.236258	48.26	86.15	36.260987	48.56	86.72
50	26.709111	48.32	86.24	61.480988	48.56	86.72
63	41.382379	48.35	86.29	96.401994	48.60	86.78
75	57.741379	48.34	86.27	136.080603	48.60	86.78
88	78.863673	48.36	86.30	186.673607	48.61	86.79
100	101.037948	48.34	86.28	240.583289	48.60	86.78
113	128.758472	48.38	86.34	306.935086	48.61	86.79
125	157.126598	48.36	86.31	375.190746	48.60	86.77
138	193.318782	48.37	86.32	456.998209	48.61	86.80

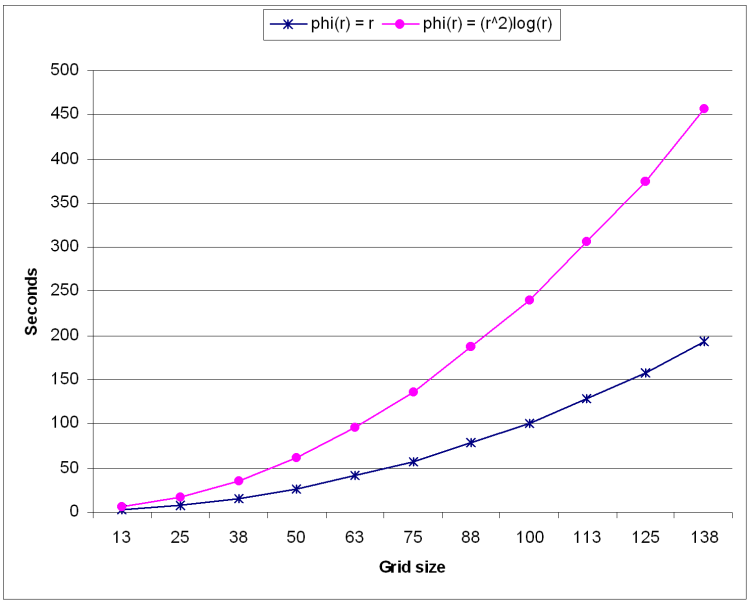
Table 4.7: The effect of grid size on run time and accuracy.

Figure 4.6 on page 101 shows the same information graphically. In Figure 4.6a we can see that the run time increases in a predictable manner according to the number of nodes in the grid[7].

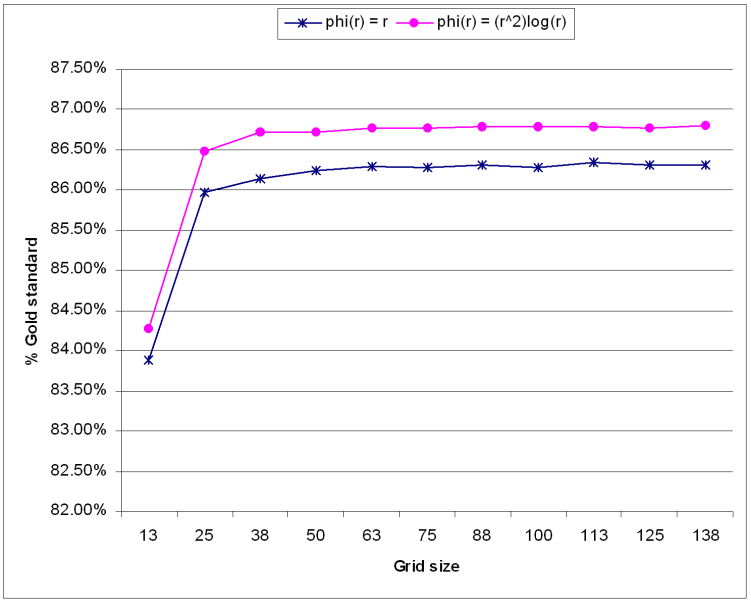
Figure 4.6b suggests that the optimum accuracy is reached with a grid size of around 60 and then appears to reach a plateau, with higher grid resolutions

[6] A grid of size 138 requires sub-sampling.

[7] The number of slices rendered is independent of the grid size.



(a) Run time versus grid size.



(b) Accuracy versus grid size.

Figure 4.6: Results for $\phi(r) = r$ and $\phi(r) = r^2 \log r$ for different grid sizes.

producing little or no gain in accuracy. We discuss this observation in Section 5.4 of Chapter 5. Note that this warping evaluation technique does not allow us to recover the spatial error statistics in the same way that we can for the remaining techniques, we must rely on the NMI metric.

4.1.4 Test 4 - Software accelerated "Fast RBF" evaluation

This is the first test of the technique we have termed "Fast RBF", which follows the basic principles outlined by Livne and Wright [101]. This technique has not previously been used for medical image registration, nor for any other practical application.

For the same reasons we mentioned when discussing the grid based evaluation in the previous section, we want to discover the maximum accuracy we can obtain using this technique in the absence of any numerical precision limits imposed by a GPU implementation. Therefore, this test looks primarily at the accuracy of a totally software based implementation. We also see how the parameter H, which controls the resolution of the warp evaluation, affects run times and accuracy.

Overview

Table 4.8 shows the results of this test using an arbitrary value of $H = 0.025$. In terms of grid cell size, this is approximately equivalent to a $6 \times 6 \times 6$ block of voxels.

As before, the table shows the warp times, NMI scores and percentage ratings for overall warp accuracy and also as a percentage of the gold standard.

We can see that we have achieved over 98% accuracy on average, compared to the gold standard when measured by NMI. This is a much better result than the best of the grid-based warps in the previous test. The run times are also an order of magnitude lower than the gold standard. As with the brute

Data set	RBF	Warp time	NMI	%Overall	%Gold
Regular linear gradient	r	45.793739	1.706122	70.62	98.42
Irregular linear gradient	r	45.585709	1.547287	54.73	98.82
Checkered cube + noise	r	45.603939	1.263534	26.35	99.57
Non-linear gradient	r	48.760342	1.677530	67.75	98.60
Regular linear gradient	$r^2 \log r$	69.015884	1.706521	70.65	98.66
Irregular linear gradient	$r^2 \log r$	69.004898	1.546156	54.62	99.21
Checkered cube + noise	$r^2 \log r$	69.159927	1.264482	26.45	99.71
Non-linear gradient	$r^2 \log r$	77.914455	1.672714	67.27	97.74

Table 4.8: Summary of software "Fast RBF" evaluation using $H = 0.025$.

force hardware technique, we can also assess the accuracy in geometric terms. Table 4.9 shows the spatial error in millimetres of the "Fast RBF" software evaluation using $H = 0.025$.

RBF	Minimum	Maximum	Range	Average	Std. Dev.
$\phi(r) = r$	0.000069	1.622394	1.622325	0.033750	0.033254
$\phi(r) = r^2 \log r$	0.000108	0.828189	0.828081	0.042332	0.032069

Table 4.9: Spatial errors in millimetres from the "Fast RBF" software evaluation using $H = 0.025$.

However, we must bear in mind that this technique is sensitive to the number of landmarks in combination with the value of H. The process of anterpolating TPS coefficients out to the nodes of the surrounding cell can have the effect of either increasing or reducing the number of TPS transformation evaluations necessary. For example, with large H and many densely distributed landmarks we may encompass more than eight landmarks per cell and so reduce the overall number after anterpolation. At the other extreme, an evenly distributed sparse set of landmarks and small H could result in at most one landmark per cell and an eight-fold increase in the number of TPS evaluations necessary.

Later, in Section 4.1.6 we will look at the effect of the number of landmarks on *all* of the evaluation techniques. For now, the following section shows how varying the grid size by adjusting the parameter H (both higher and lower) affects the accuracy and run time of this method.

Effect of grid size - parameter H

Table 4.10 shows the run times and the NMI accuracy compared to the gold standard for various values of H. These figures are the averages of all four test data sets for each RBF.

	$\phi(r) = r$			$\phi(r) = r^2 \log r$		
H	**Warp time**	**%Overall**	**%Gold**	**Warp time**	**%Overall**	**%Gold**
0.105	11.433087	41.28	91.24	11.855772	40.58	90.85
0.095	11.401259	41.30	91.26	11.920685	41.44	91.38
0.085	11.506339	42.56	92.02	12.075928	38.88	89.69
0.075	11.617253	42.82	92.20	12.325167	37.26	88.77
0.065	12.490433	47.98	95.36	13.516259	37.10	88.59
0.055	13.106529	74.84	96.41	15.400303	45.30	93.77
0.045	15.942261	49.68	97.43	20.884502	51.24	97.41
0.035	22.830587	53.02	98.45	33.098789	51.46	97.52
0.025	46.435932	54.86	99.57	71.273791	54.74	99.55
0.015	499.046082	55.44	99.92	608.992973	55.22	99.83

Table 4.10: The effect of parameter H on run time and accuracy for the software "Fast RBF" method.

Figure 4.7 on page 107 shows the same information graphically. The shape of the plot in Figure 4.7a suggests that the run time is strongly influenced by the cube of the grid size (i.e. $(\frac{c}{H})^3$ where c is some constant), which agrees with Livne and Wright [101, pp. 6,15] in which the complexity is derived from

the following :

$$W \sim (n+m)p^d + \left(\frac{c}{H}\right)^d \tag{4.1}$$

where n is the number of landmarks (centres), m is the number of voxels (evaluation points), p is the degree of the polynomial, d is the dimension and c is a constant.

Figure 4.7b shows an upward trend in accuracy with decreasing H although there is a marked dip in the $\phi(r) = r^2 \log r$ plot which suggests that the characteristics of this technique are not as straightforward as the previous grid-based evaluation. Looking at the $\phi(r) = r$ plot on the same chart also shows a possible dip in accuracy at a similar point, though this is not clear.

As Figure 4.7b is based on an average of all four test data sets, the effect may be clearer by looking at individual plots for each type of test data. Figure 4.8 on page 108 shows exactly that. Figure 4.8a shows the breakdown of accuracy by test data set for $\phi(r) = r$ and Figure 4.8b shows the same for $\phi(r) = r^2 \log r$.

We will discuss this effect in more detail in Section 5.5 of Chapter 5.

We can also look at the spatial errors arising from different values of H. Tables 4.11 and 4.12 show the spatial errors in millimetres of the "Fast RBF" software evaluation using the same values of H as in Table 4.10.

4.1.5 Test 5 - Hardware accelerated "Fast RBF" evaluation

The hardware implementation of the "Fast RBF" method moves all of the processing, except the anterpolation stage, onto the GPU. As described in Section 3.15, this technique uses 32-bit floating point textures to store the grid $\mathbf{Y}$, the coarse level expansion coefficients $\Lambda(\mathbf{Y})$ and the coarse level summation results in the grid $\mathbf{X}$.

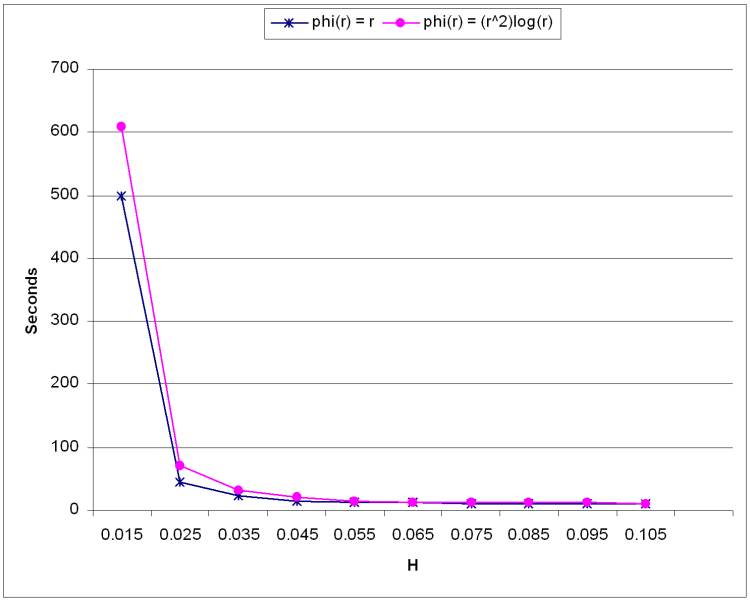
Due to the reduced numerical precision in this implementation, we might anticipate a reduction in accuracy. This test will attempt to quantify that reduction – if it exists – and see how it is affected by the choice of H.

H	Minimum	Maximum	Range	Average	Std. Dev.
0.015	0.000037	1.071275	1.071238	0.013455	0.013465
0.025	0.000069	1.622394	1.622325	0.033750	0.033254
0.035	0.000149	1.404107	1.403958	0.066507	0.049985
0.045	0.000243	2.843206	2.842963	0.099676	0.082942
0.055	0.000215	2.846024	2.845809	0.144525	0.121993
0.065	0.000636	3.772129	3.771493	0.192216	0.165122
0.075	0.001314	4.529274	4.527960	0.311585	0.223631
0.085	0.000966	3.982248	3.981282	0.360420	0.249317
0.095	0.000463	5.363233	5.362770	0.400725	0.300441
0.105	0.000715	4.659923	4.659208	0.398366	0.273161

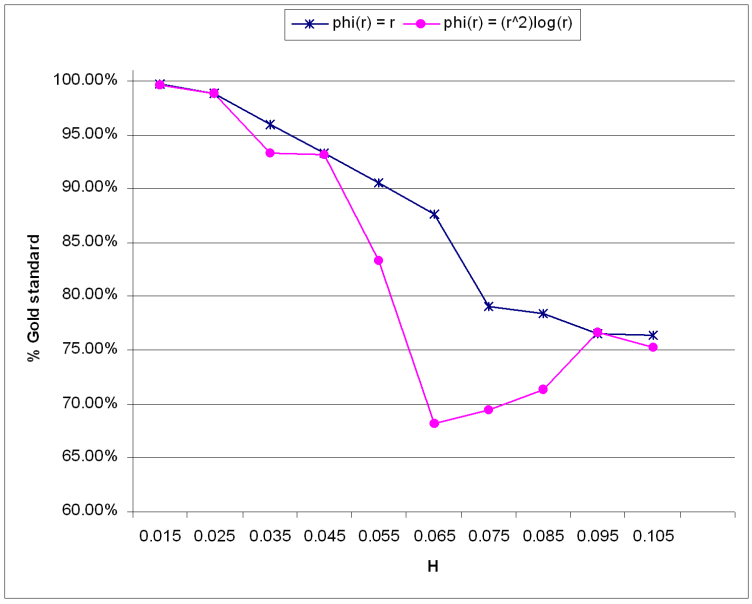
Table 4.11: Spatial errors in millimetres from the "Fast RBF" software evaluation for different values of H, using $\phi(r) = r$.

H	Minimum	Maximum	Range	Average	Std. Dev.
0.015	0.000113	0.436184	0.436071	0.028855	0.017048
0.025	0.000108	0.828189	0.828081	0.042332	0.032069
0.035	0.000293	0.873480	0.873187	0.133756	0.071494
0.045	0.000174	1.796661	1.796487	0.117894	0.088253
0.055	0.000875	2.197619	2.196744	0.285848	0.157878
0.065	0.001454	2.672732	2.671278	0.707298	0.324460
0.075	0.001936	4.234098	4.232162	0.576603	0.325622
0.085	0.003252	3.447342	3.444090	0.657323	0.338104
0.095	0.001066	4.751118	4.750052	0.437001	0.318578
0.105	0.000732	4.726483	4.725751	0.460985	0.331315

Table 4.12: Spatial errors in millimetres from the "Fast RBF" software evaluation for different values of H, using $\phi(r) = r^2 \log r$.

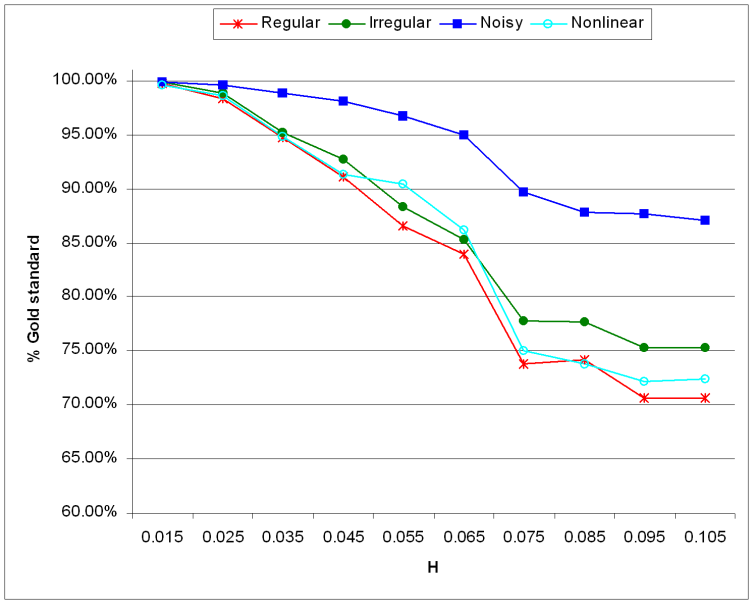


(a) Run time versus H.

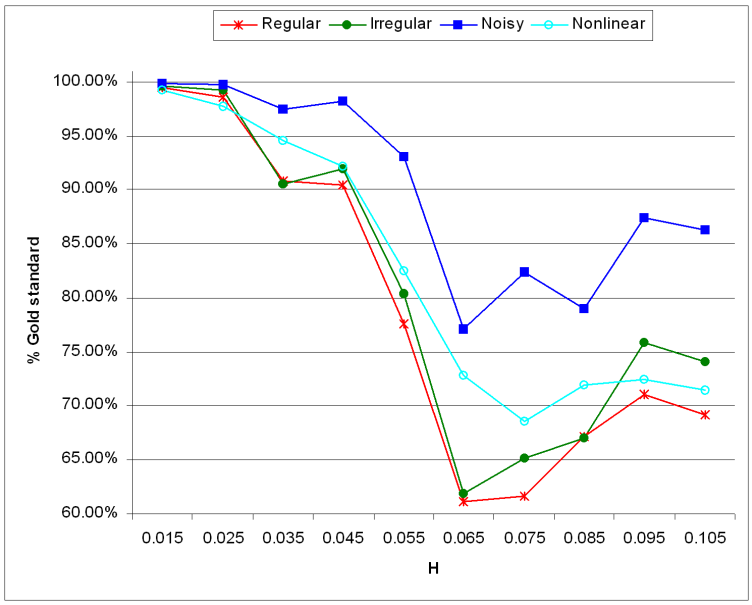


(b) Accuracy versus H.

Figure 4.7: Results for software "Fast RBF" showing $\phi(r) = r$ and $\phi(r) = r^2 \log r$ metrics for different values of H.



(a) Accuracy versus H for $\phi(r) = r$.



(b) Accuracy versus H for $\phi(r) = r^2 \log r$.

Figure 4.8: Accuracy of $\phi(r) = r$ and $\phi(r) = r^2 \log r$ for different values of H by test data type – software implementation.

Overview

Table 4.13 shows the results of this test using a value of $H = 0.025$. This is the same value we used for the summarised results of the software implementation in the previous section.

The table shows the warp times, NMI scores and percentage ratings for the overall warp accuracy and as a percentage of the software gold standard.

Data set	RBF	Warp time	NMI	%Overall	%Gold
Regular linear gradient	r	0.954813	1.618305	61.84	86.19
Irregular linear gradient	r	0.973264	1.490224	49.02	88.50
Checkered cube + noise	r	0.972872	1.199092	19.90	75.20
Non-linear gradient	r	0.893234	1.523904	52.40	76.26
Regular linear gradient	$r^2 \log r$	1.006130	1.617567	61.76	86.24
Irregular linear gradient	$r^2 \log r$	0.951575	1.491939	49.20	89.35
Checkered cube + noise	$r^2 \log r$	0.957887	1.199444	19.94	75.19
Non-linear gradient	$r^2 \log r$	0.892530	1.525948	52.60	76.40

Table 4.13: Summary of hardware "Fast RBF" evaluation using $H = 0.025$.

We can see that the 98% average accuracy against the gold standard, which we noted for the software "Fast RBF" implementation, has now dropped to just under 82%. However, it is also worth noting that we are now measuring sub-second warping times which makes this the fastest implementation so far. In addition, we saw from the hardware implementation of the brute force technique in Section 4.1.2 that the best accuracy we could anticipate from a GPU based implementation is 89% of the gold standard, which is consistent with what we have found.

As with the previous technique, we can also assess the accuracy in geometric terms. Table 4.14 shows the spatial error in millimetres of the "Fast RBF" hardware evaluation using $H = 0.025$.

The following section shows how varying the grid size by adjusting the

RBF	Minimum	Maximum	Range	Average	Std. Dev.
$\phi(r) = r$	0.000098	1.622334	1.622236	0.034282	0.033201
$\phi(r) = r^2 \log r$	0.000092	0.827431	0.827339	0.042255	0.032132

Table 4.14: Spatial errors in millimetres from the "Fast RBF" hardware evaluation using $H = 0.025$.

parameter H (both higher and lower) affects the accuracy and run time of this method. Given the above summary, we would expect to see similar results to the software "Fast RBF" implementation but with generally lower accuracy metrics.

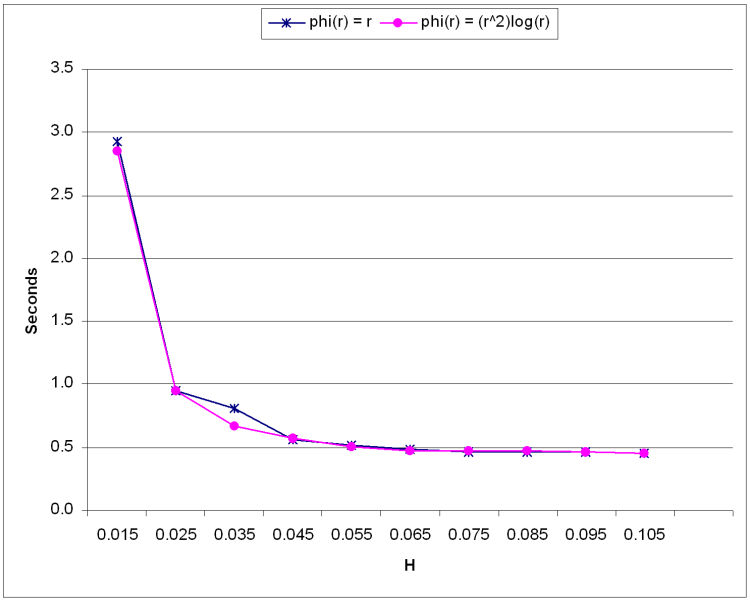
Effect of grid size - parameter H

Table 4.15 shows the run times and the accuracy compared to the gold standard for various values of H using the hardware accelerated "Fast RBF" implementation. As usual, these figures are the averages of all four test data sets for each RBF.

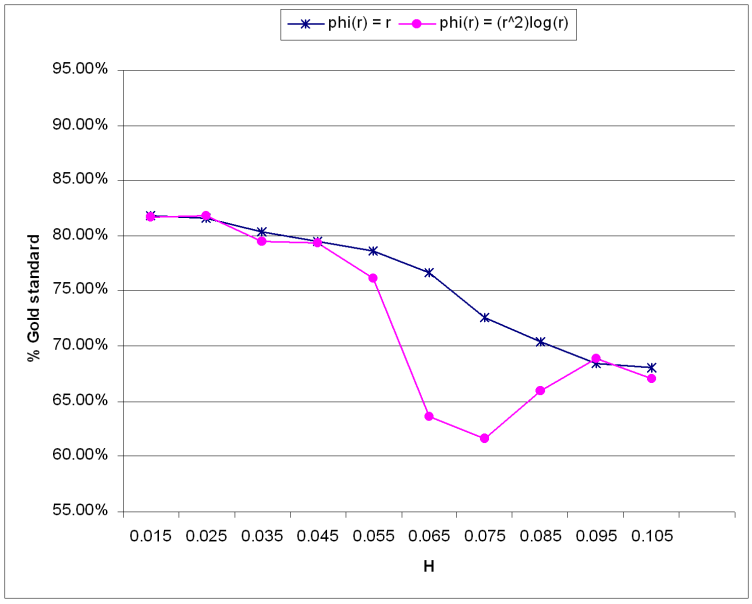
Figure 4.9 on page 111 shows the same information graphically. As we saw with the software implementation of this technique, the shape of the plot in Figure 4.9a supports the theory that the run time of this implementation is inversely proportional to the cube of the grid size.

Figure 4.9b shows the upward trend in accuracy with decreasing H and the same dip for $\phi(r) = r^2 \log r$, although the overall accuracy figures are lower than their software counterparts from the previous test.

As with the software results in the previous section, Figure 4.9b is based on an average of all four test data sets and the effect is clearer by looking at individual plots for each type of test data. In Figure 4.10 on page 112, Figure 4.10a shows the breakdown of accuracy by test data set for $\phi(r) = r$ and Figure 4.10b shows the same for $\phi(r) = r^2 \log r$.

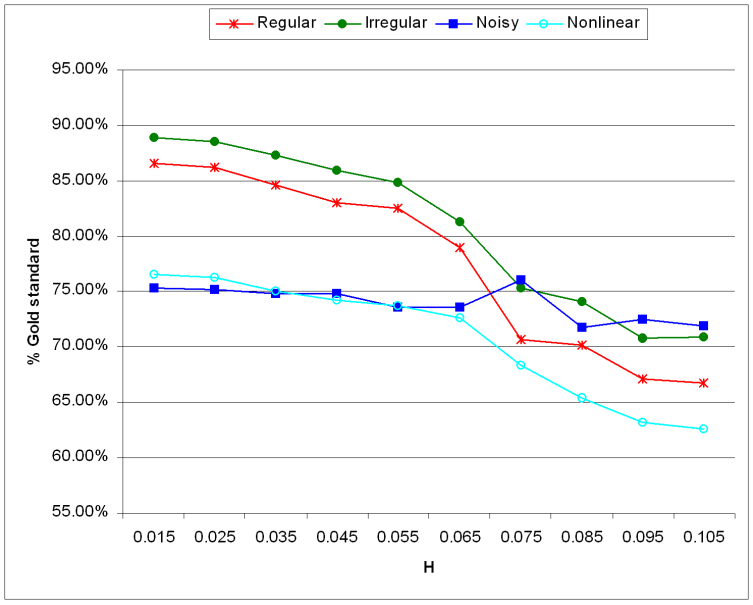


(a) Run time versus H.

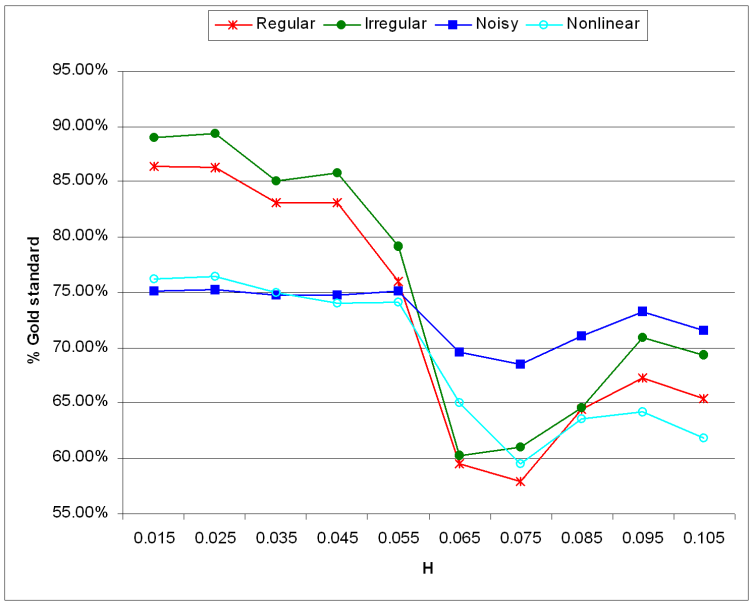


(b) Accuracy versus H.

Figure 4.9: Results for hardware "Fast RBF" showing $\phi(r) = r$ and $\phi(r) = r^2 \log r$ metrics for different values of H.



(a) Accuracy versus H for $\phi(r) = r$.



(b) Accuracy versus H for $\phi(r) = r^2 \log r$.

Figure 4.10: Accuracy of $\phi(r) = r$ and $\phi(r) = r^2 \log r$ for different values of H by test data type – hardware implementation.

	$\phi(r) = r$			$\phi(r) = r^2 \log r$		
H	**Warp time**	**%Overall**	**%Gold**	**Warp time**	**%Overall**	**%Gold**
0.105	0.456613	37.31	68.04	0.456223	36.66	67.08
0.095	0.460664	37.49	68.39	0.469113	37.71	68.93
0.085	0.463629	38.82	70.35	0.479706	36.07	65.92
0.075	0.463415	39.88	72.61	0.471908	33.53	61.71
0.065	0.488655	42.76	76.62	0.479117	34.76	63.61
0.055	0.516709	44.08	78.66	0.504753	42.24	76.13
0.045	0.561979	44.49	79.49	0.575920	44.38	79.42
0.035	0.814210	45.08	80.40	0.674261	44.46	79.51
0.025	0.948546	45.79	81.54	0.952030	45.87	81.81
0.015	2.928600	45.95	81.81	2.847230	45.82	81.71

Table 4.15: The effect of parameter H on run time and accuracy for the hardware "Fast RBF" method.

As before, we can look at the spatial errors arising from different values of H. Tables 4.16 and 4.17 show the spatial errors in millimetres of the "Fast RBF" hardware evaluation using the same values of H as in Table 4.15.

4.1.6 Test 6 - Varying the number of landmarks

The number of landmark pairs supplied to the warping function (Equation 3.18 on page 37) is one of the factors that determines the algorithmic cost of evaluating the function. This also applies to the algorithmic cost of solving the TPS transformation as we described in Section 3.3.1.

The purpose of this test is to produce some empirical evidence of the way that each of the warp implementation techniques perform with different numbers of landmarks. Here, we are not concerned with registration accuracy, just with execution time. All of the tests in this section use the same RBF, $\phi(r) = r$, so that we remove any timing differences caused by different implementations

H	Minimum	Maximum	Range	Average	Std. Dev.
0.015	0.000015	1.071309	1.071294	0.014006	0.013453
0.025	0.000098	1.622334	1.622236	0.034282	0.033201
0.035	0.000202	1.404055	1.403853	0.067216	0.050067
0.045	0.000269	2.843270	2.843001	0.100291	0.083141
0.055	0.000296	2.845291	2.844995	0.144578	0.121903
0.065	0.000755	3.772151	3.771396	0.192442	0.165332
0.075	0.001263	4.529306	4.528043	0.312066	0.224119
0.085	0.001024	3.982281	3.981257	0.361054	0.249665
0.095	0.000401	5.363257	5.362856	0.401427	0.300769
0.105	0.000785	4.659948	4.659163	0.399027	0.273436

Table 4.16: Spatial errors in millimetres from the "Fast RBF" hardware evaluation for different values of H, using $\phi(r) = r$.

H	Minimum	Maximum	Range	Average	Std. Dev.
0.015	0.000104	0.435423	0.435319	0.029072	0.017022
0.025	0.000092	0.827431	0.827339	0.042255	0.032132
0.035	0.000260	0.874012	0.873752	0.134088	0.071565
0.045	0.000236	1.795971	1.795735	0.118055	0.088311
0.055	0.000669	2.198010	2.197341	0.285559	0.157897
0.065	0.001243	2.672881	2.671638	0.706471	0.324522
0.075	0.002013	4.233896	4.231883	0.577266	0.325446
0.085	0.003227	3.447519	3.444292	0.658233	0.338105
0.095	0.001042	4.751009	4.749967	0.436963	0.318877
0.105	0.000706	4.726399	4.725693	0.460962	0.331480

Table 4.17: Spatial errors in millimetres from the "Fast RBF" hardware evaluation for different values of H, using $\phi(r) = r^2 \log r$.

of the more complex RBF and just concentrate on the effect of the number of landmarks.

Before we look at each warping technique in turn, we will examine the run times of the function that solves the TPS transformation.

TPS solution time

The experimental software records run times for each part of the warping procedure separately. Therefore, we have plenty of data from which to extract details of the time taken to solve the TPS transformation for different numbers of landmarks.

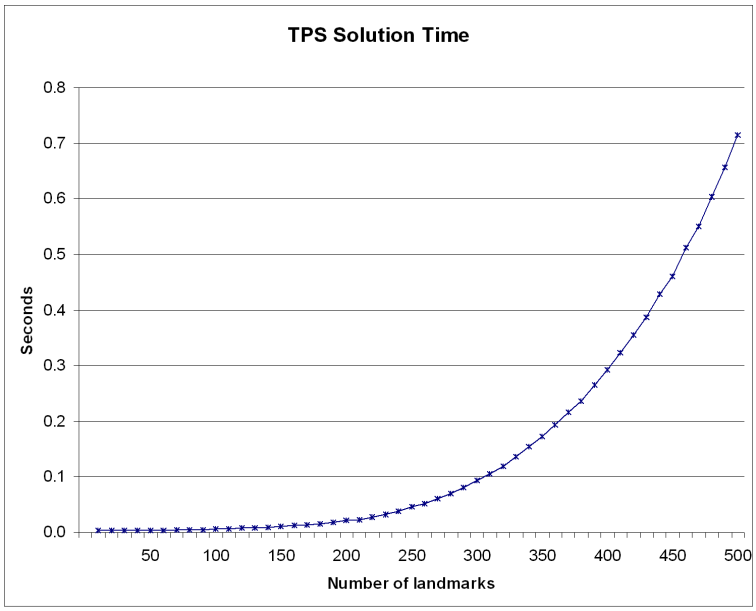
Figure 4.11 shows two charts. In Figure 4.11a, the time taken to solve the TPS transformation is plotted against the number of landmarks, n. In this chart, each point represents the average of 33 runs – one brute force software, one brute force hardware, 11 grid-based, 10 "Fast RBF" software and 10 "Fast RBF" hardware.

From [101, p. 2], we expect the complexity of the TPS solver algorithm to be $O(n^3)$. So the second chart, Figure 4.11b, attempts to verify this by plotting the same timings against the cube of the number of landmarks (i.e. n^3) to show that this implementation of the TPS solver performs reasonably against expectation.

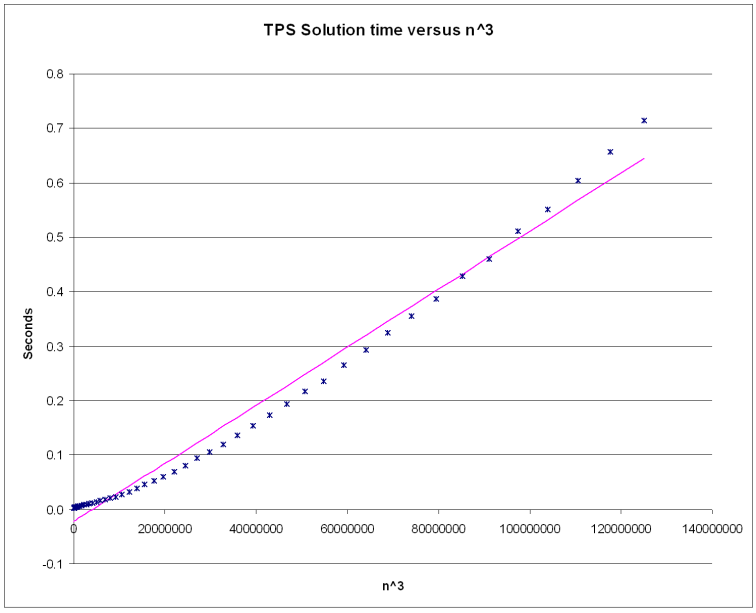
Now we will look at each of the warp evaluation techniques in turn and see how they perform when varying the number of landmarks.

Brute force software implementation

The brute force implementation evaluates Equation 3.18 once for every voxel in the target data set and the summation term in the equation is limited by the number of landmarks. Therefore, as we noted in Section 3.14, the brute force method has a cost of $O(mn)$, where m is the number of voxels and n is the



(a) TPS solution time versus number of landmarks, n.



(b) TPS solution time versus n^3 with regression line.

Figure 4.11: Empirical evaluation of TPS solution time with varying number of landmarks.

number of landmarks, and so we expect to see a linear relationship between the number of landmarks and the warp evaluation time.

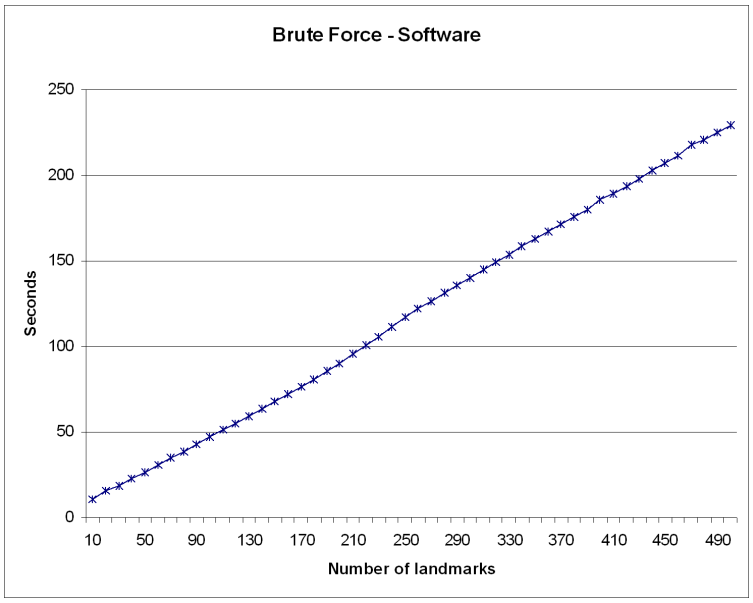


Figure 4.12: Effect of the number of landmarks on run time of the brute force software implementation.

Figure 4.12 shows the results of this test, which is clearly in line with expectations. Average run times range from 10 seconds for 10 landmarks to 230 seconds for 500 landmarks.

Brute force hardware implementation

As the underlying algorithms are identical, the hardware implementation of the brute force evaluation should behave in exactly the same way as the software implementation.

Figure 4.13 shows the results of this test, which shows exactly what we would expect. The run times for the hardware implementation are, on average, just 4% of the equivalent software times – ranging from 0.48 seconds for 10 landmarks to 8.85 seconds for 500 landmarks.

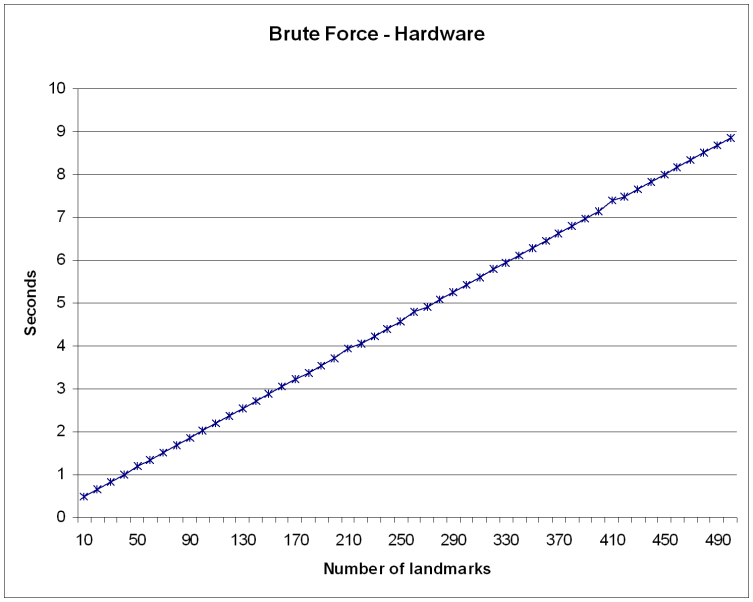


Figure 4.13: Effect of the number of landmarks on run time of the brute force hardware implementation.

Grid-based implementation

The run time of the grid based warp evaluation should theoretically depend on the number of landmarks and the size of the grid. Figure 4.14 shows multiple plots, each data series representing a particular size of grid.

This chart shows a clear linear relationship, from which it should be feasible to predict the expected run time for higher numbers of landmarks by extrapolation.

Table 4.18 shows predicted warp evaluation times for each grid size used in the tests, using 729 landmarks. The third column in the table shows the actual times we measured during the tests with the synthetic data sets (which had 729 landmarks). We can see that it is possible to predict the warp evaluation times quite accurately where we use relatively fine grids but less accurately where we use coarse grids.

One reason we see this effect may be because, with coarse grids, the warp

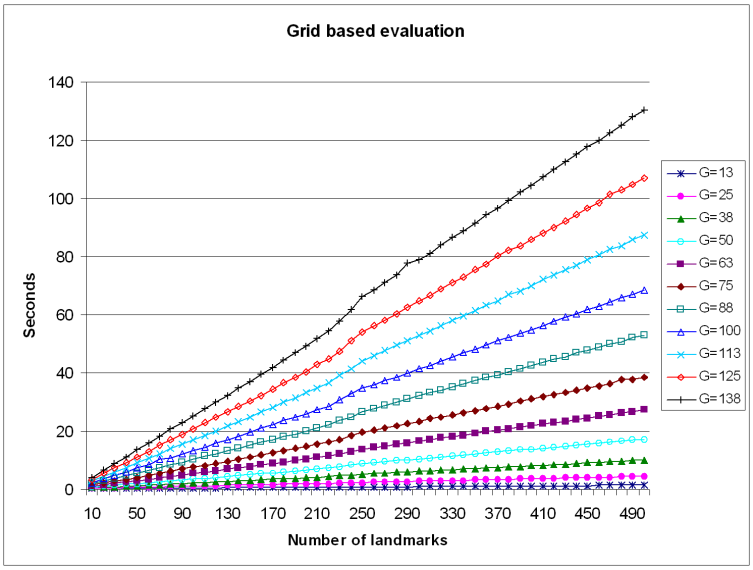


Figure 4.14: Effect of the number of landmarks on run time of the grid-based implementation.

Grid	Predicted	Actual	Error	%Error
13	2.227066	3.681314	1.454248	39.50
25	6.701287	8.145888	1.444601	17.73
38	14.809274	16.236258	1.426983	8.79
50	25.264745	26.709111	1.444367	5.41
63	39.826748	41.382379	1.555631	3.76
75	56.314812	57.741379	1.426567	2.47
88	77.247539	78.863673	1.616134	2.05
100	99.842098	101.037948	1.195850	1.18
113	127.367765	128.758472	1.390707	1.08
125	155.905561	157.126598	1.221037	0.78
138	189.905122	193.318782	3.413661	1.77

Table 4.18: Predictions of warp times for the grid-based method.

evaluation time is small compared to the 'fixed cost' time taken to prepare the data – i.e. load textures, render slices, recover GPU buffer, etc. When we use finer grids and so have a longer overall warp evaluation time, the effect of this fixed cost becomes negligible.

"Fast RBF" software implementation

As we saw in Section 4.1.4, the run times for the "Fast RBF" implementation vary with $(\frac{c}{H})^3$ and therefore cover a very wide range with the chosen values of H (0.015 to 0.105). The run time range is too wide to be shown clearly on a single chart, we have to break it down further.

The three charts in Figures 4.15, 4.16 and 4.17 show all the results for grid sizes of 0.015 up to 0.105. Note that the scale of the y axis differs significantly between these charts – Figure 4.15 covers a range of run times from zero to 500 seconds while Figure 4.16 ranges from zero to 50 seconds and Figure 4.17 covers just 1.5 seconds in total.

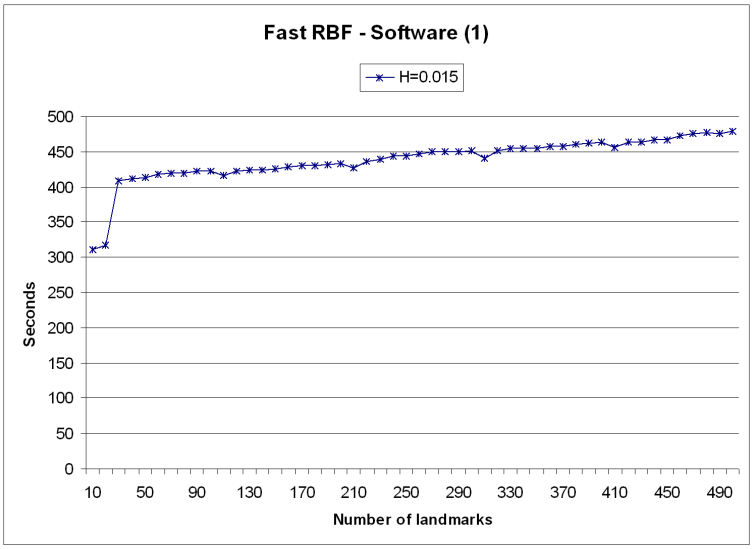


Figure 4.15: Effect of the number of landmarks on run time of the "Fast RBF" software implementation – $H = 0.015$.

In Figure 4.17, there is still a clearly defined relationship between the num-

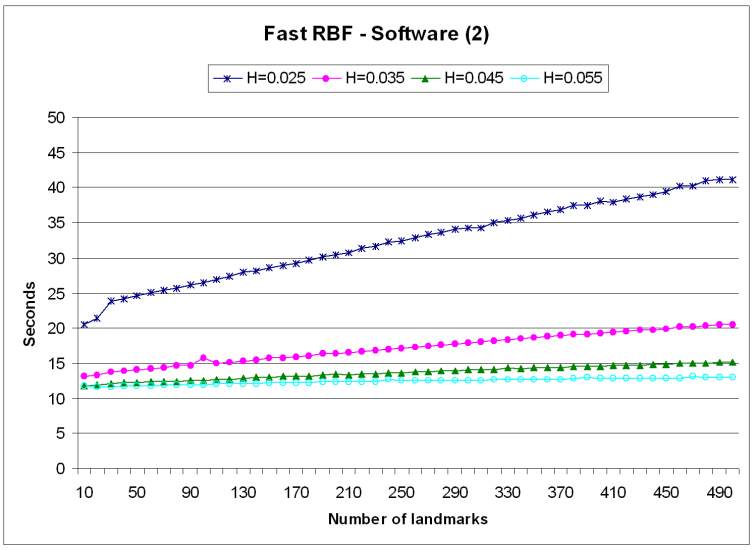


Figure 4.16: Effect of the number of landmarks on run time of the "Fast RBF" software implementation – $H = 0.025 \ldots 0.055$.

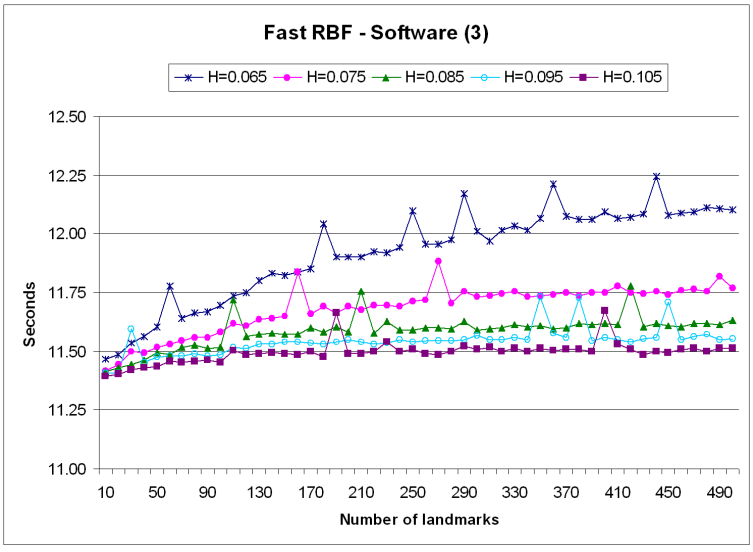


Figure 4.17: Effect of the number of landmarks on run time of the "Fast RBF" software implementation – $H = 0.065 \ldots 0.105$.

ber of landmarks and the run time, although some variations in run time appear as peaks on the chart. This may or may not be due to the magnified y axis scale – we discuss this further in Section 5.6 of Chapter 5.

In general, these charts suggest that the major factor affecting run time for this technique is the value of the parameter H rather than the number of landmarks. The individual plots do not cross each other[8] but tend to diverge with increasing numbers of landmarks.

"Fast RBF" hardware implementation

As with the software implementation of the "Fast RBF" method, the hardware implementation exhibits a wide range of run times depending on the value of the parameter H.

Again, the effect of varying the number of landmarks when using this technique is best shown on three separate charts, each with a different scale in the y axis.

Following the same breakdown as the software "Fast RBF" implementation, the three charts in Figures 4.18, 4.19 and 4.20 show the results for grid sizes of 0.015 up to 0.105. The scale of the y axis in Figure 4.18 covers a range of run times from 0.5 to 2.0 seconds while Figure 4.19 ranges from 0.25 to 1.0 seconds and Figure 4.20 covers just 0.15 seconds overall.

Notice that the chart in Figure 4.18, which is at the same scale as the 'peaky' chart in Figure 4.17, shows hardly any anomalies in the run time measurements. However, as we decrease the run time scale to accommodate larger values of H, the peaks tend to reappear – see Figures 4.19 and 4.20. As mentioned in the previous section, we discuss this further in Section 5.6 of Chapter 5.

[8] Apart from the 'noise' in Figure 4.17.

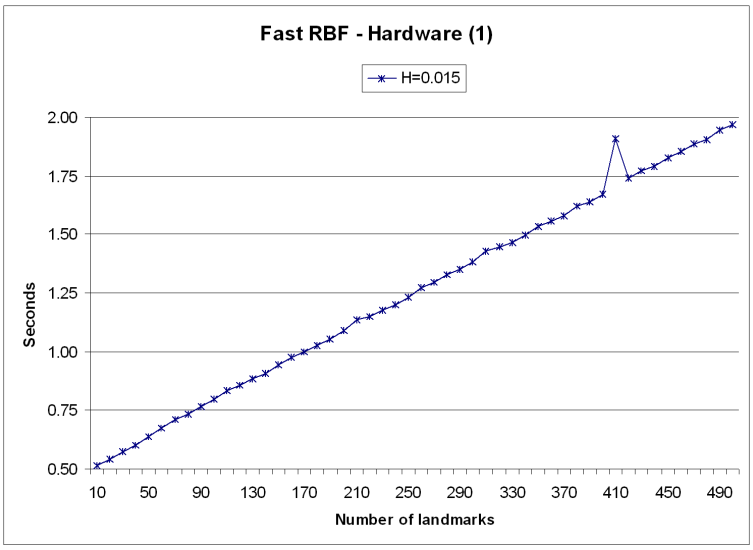


Figure 4.18: Effect of the number of landmarks on run time of the "Fast RBF" hardware implementation – $H = 0.015$.

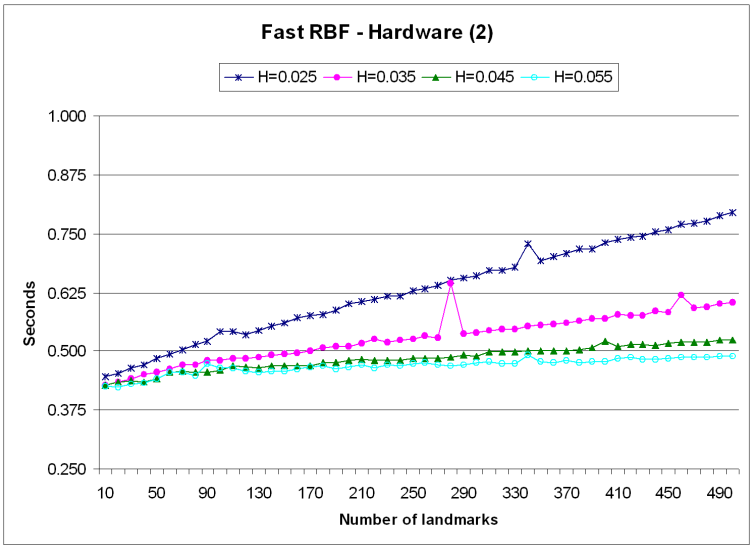


Figure 4.19: Effect of the number of landmarks on run time of the "Fast RBF" hardware implementation – $H = 0.025 \ldots 0.055$.

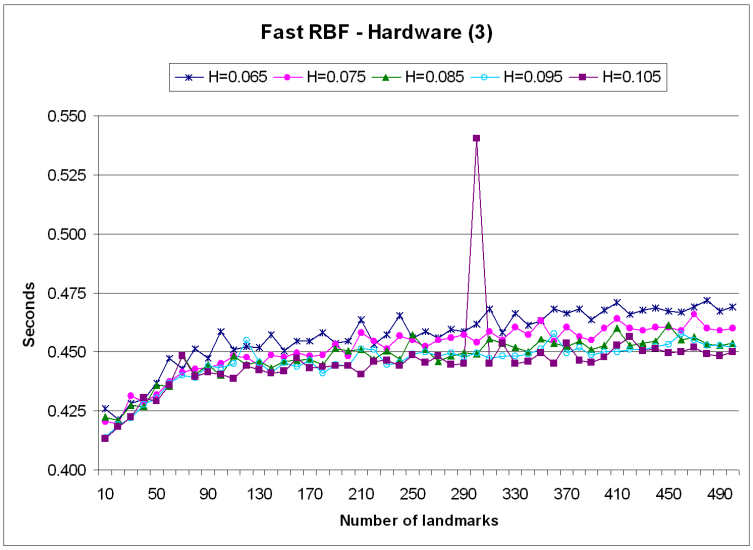
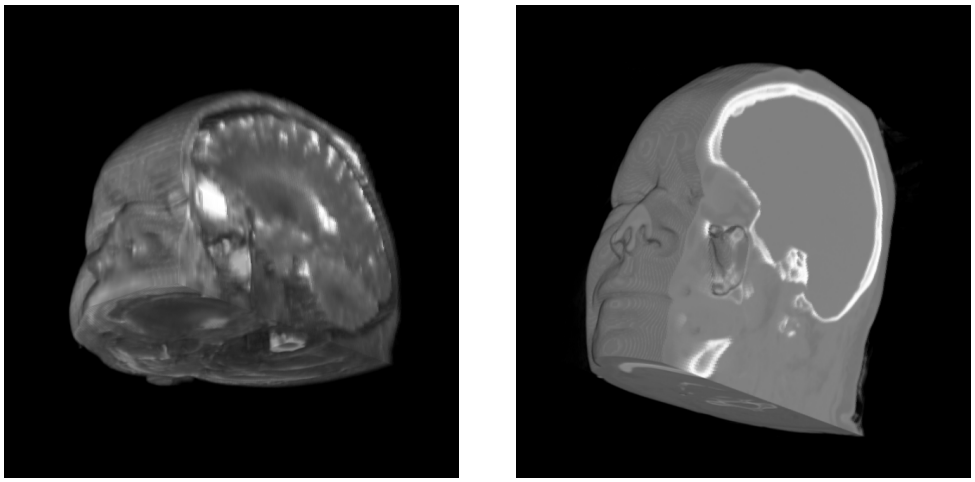


Figure 4.20: Effect of the number of landmarks on run time of the "Fast RBF" hardware implementation – $H = 0.065 \ldots 0.105$.

4.2 Visible Human data tests

The final set of tests use MRI and CT data sets from the Visible Human Project [1]. Figure 4.21 shows example cut-away views of 3D reconstructions built from the data we used for this test.



(a) MRI Data.

(b) CT Data.

Figure 4.21: 3D reconstructions using Visible Human data sets.

4.2.1 Test 7 - CT and MRI registration

The purpose of this test is to register 3D medical image data sets of the same subject acquired by different imaging modalities. As we did with the synthetic data tests, we will use the NMI metric to provide a quantitative view of the registration accuracy, but we will also concentrate on a visual inspection of the data both before and after warping.

We chose the NMI metric for two primary reasons – first, this metric is suitable for multi-modal image registration and second, as Figure 4.21 shows, we do not have a complete overlap in the image data. As mentioned in Section 3.5.5, the NMI metric is overlap invariant.

Overview

As mentioned in the test plan in Section 3.16, this test will involve three registration procedures.

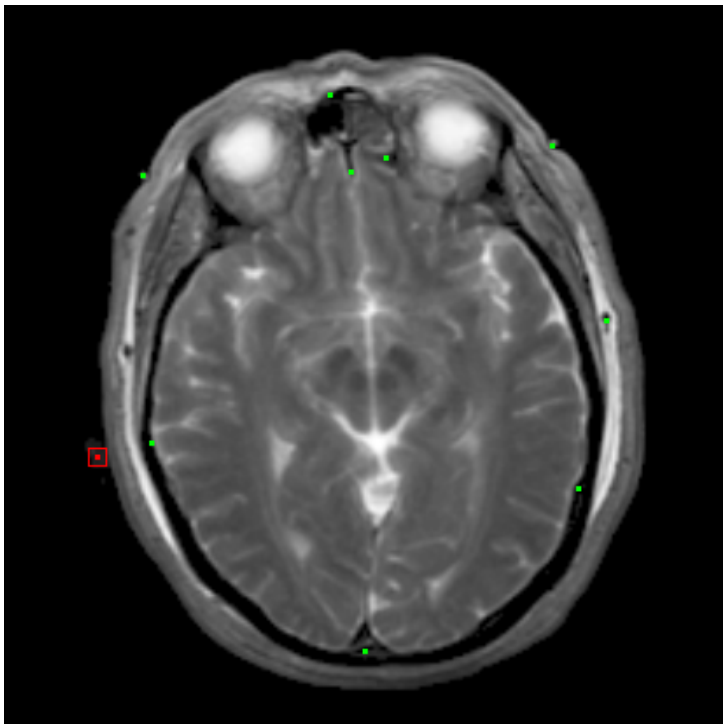
1. Perform the brute force software warp evaluation to transform the MRI data into the CT image space.

2. Perform the same warp using the most accurate optimised technique from the earlier tests – the "Fast RBF" software implementation.

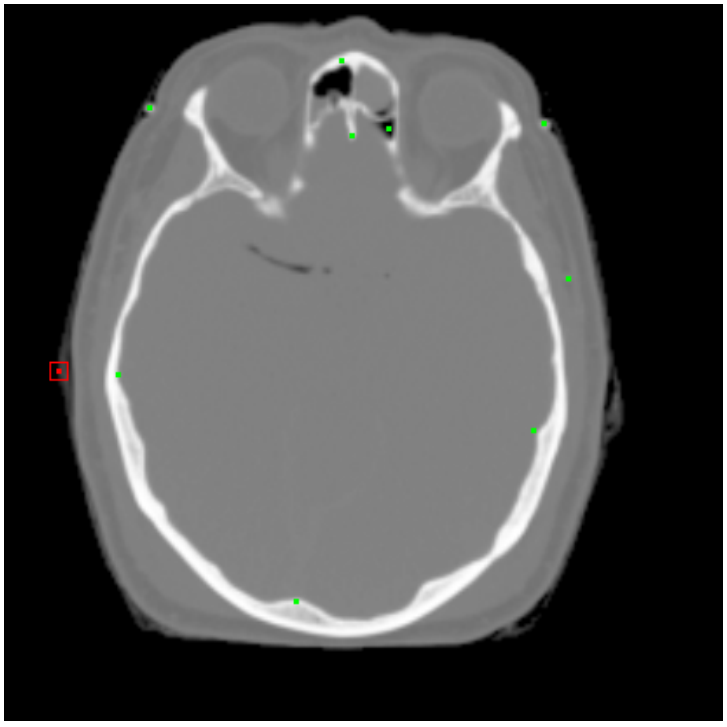
3. Finally, perform the warp again using the fastest optimised technique from the earlier tests – the "Fast RBF" hardware implementation.

After we have completed the three warp evaluations, we will compare run times, image similarity metrics and visual results of all techniques.

The first step is to identify homologous points on the two data sets to create a set of landmark pairs. Figure 4.22 shows a sample view of this process. The intention is to warp the MRI data set (Figure 4.22a) into the image space occupied by the CT data set (Figure 4.22b).



(a) Source data set – MRI.



(b) Target data set – CT.

Figure 4.22: Manual selection of homologous landmarks on CT and MRI data.

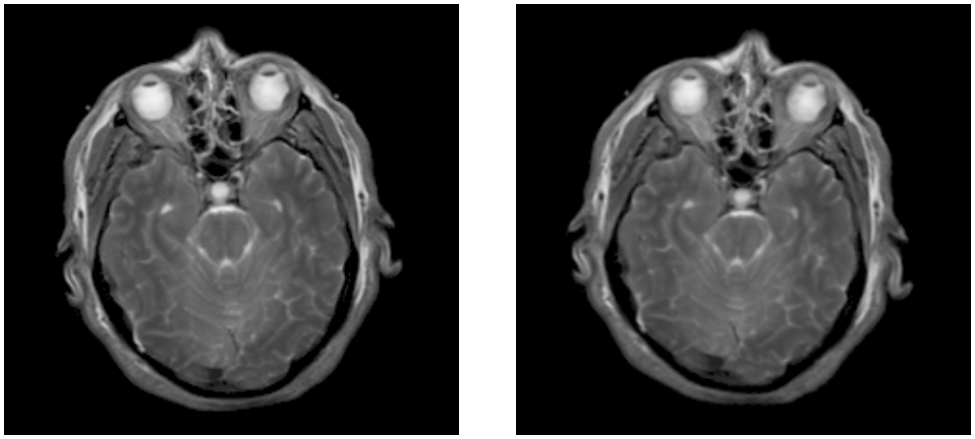
Before we start warping, we calculate the NMI metric on the CT and MRI data sets to measure the baseline similarity:

NMI metric for unwarped MRI and CT data : **1.087436**

Brute force CT/MRI registration - software

As with the previous tests, we will use the brute force software implementation to set the target standard.

For ease of comparison, both the CT and MRI data sets have been resampled into 256^3 3D images, each therefore containing 16,777,216 equally sized voxels. Using the experimental software, we have manually marked 53 pairs of matching points from the two sets of data, which we will use for performing all warp evaluations.

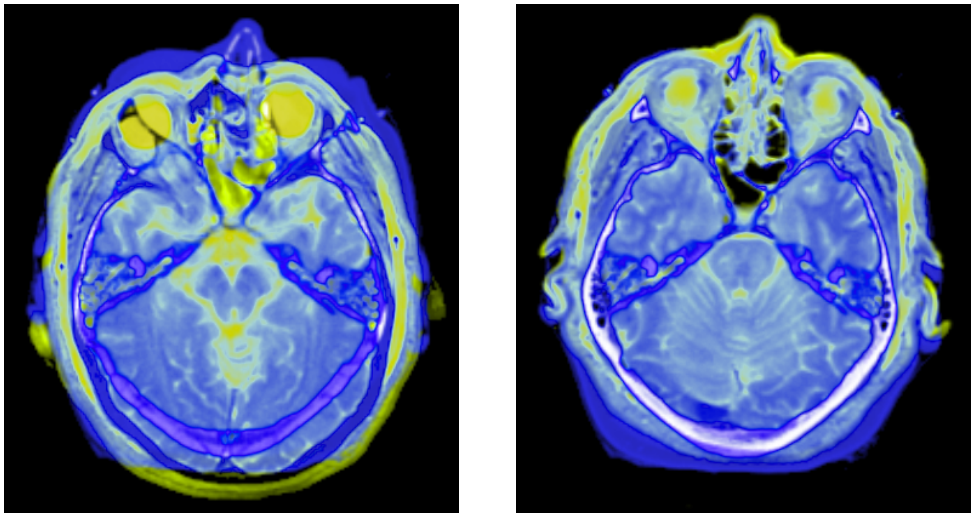


(a) MRI before warp. (b) MRI after warp.

Figure 4.23: A sample 2D slice from the MRI data set, before and after warping.

Figure 4.23 shows a single slice through the MRI data set before and after applying the brute force warp evaluation. By colourising slices from the CT and MRI data sets, we can visualise the result of the registration a little more clearly. Figure 4.24 on page 128 shows two merged images – Figure 4.24a shows the CT data (blue) and MRI data (yellow) *before* the MRI data set had been

warped and Figure 4.24b shows the same slices *after* warping the MRI data.



(a) Before registration. (b) After registration.

Figure 4.24: Merged images of colourised CT data (blue) and MRI data (yellow), before and after registration.

Measurements for the software brute force warp evaluation are as follows

Brute force software evaluation time : **32.287069 seconds**

NMI for warped MRI and CT data : **1.123700** (gold standard)

"Fast RBF" CT/MRI registration - software

The second part of this test is to perform the same warp evaluation on the MRI data set, this time using the "Fast RBF" software evaluation technique. The synthetic data tests in Section 4.1 showed this technique to be the most accurate when compared to the gold standard although we need to decide on a value of the parameter H.

If we look back at Table 4.10 on page 104, we can see that the greatest accuracy (99.78%) was achieved with $H = 0.015$. However, we also obtained an accuracy of 98.85% with $H = 0.025$ but with a significantly reduced run time

and a maximum spatial error of 1.6mm (average 0.033750, sigma 0.033254).

A cost of 0.93% in accuracy at the gain of more than tenfold improvement in run time seems like a fair compromise. We have no better scientific method of choosing H so, for the "Fast RBF" tests, we will use $H = 0.025$ as a good balance between execution time and accuracy.

Measurements for the software "Fast RBF" warp evaluation are as follows

"Fast RBF" software evaluation time : **19.475016 seconds**
NMI for warped MRI and CT data : **1.123614** (99.99% of gold standard)

Figure 4.25 shows a greatly enhanced (gamma adjusted, $\gamma = 0.2$) difference image of a typical 2D slice of the warped MRI data set. The image represents the intensity differences between the brute force software result and the "Fast RBF" software result.

Overall, 91.14% of voxels match exactly and a further 8% are no more than 3 intensity points different (intensities range from $0 \ldots 255$). The chart in Figure 4.25b shows the count of voxels against intensity difference for mismatched voxels only.

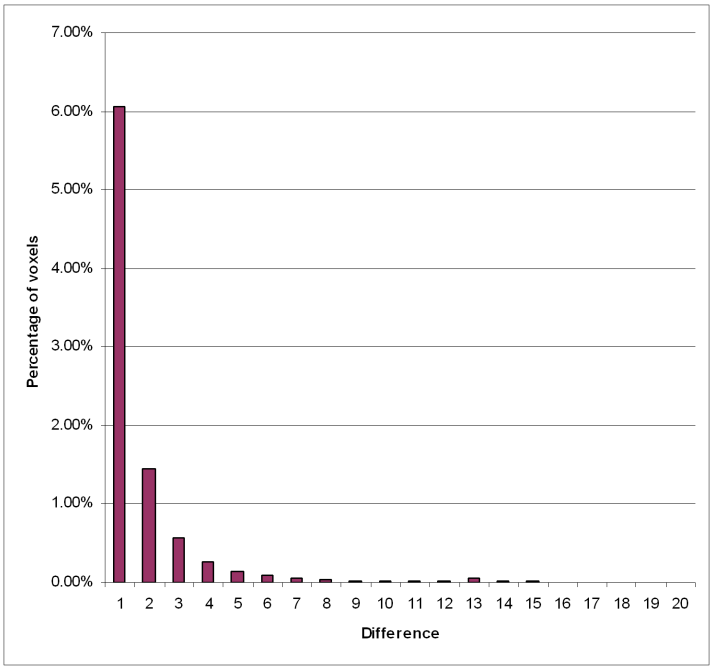
"Fast RBF" CT/MRI registration - hardware

The final part of this test is to perform the warp of the MRI data using the "Fast RBF" evaluation technique implemented in hardware. As with the previous test we will use $H = 0.025$.

As we showed with the software implementation in the previous section, Figure 4.26 shows a gamma adjusted ($\gamma = 0.5$) difference image of a typical 2D slice of the warped MRI data set. This image represents the intensity differences between the brute force software result and the "Fast RBF" hardware result.

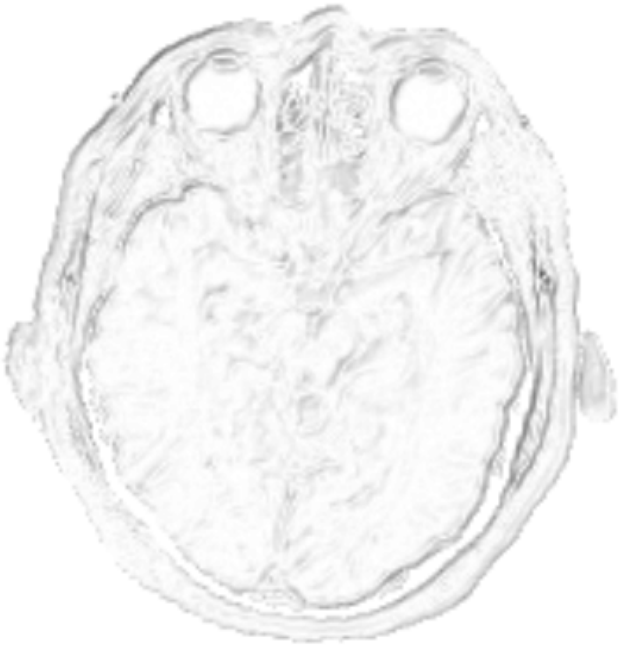
In contrast to the software "Fast RBF" test, only 81.79% of voxels match

(a) Difference image.

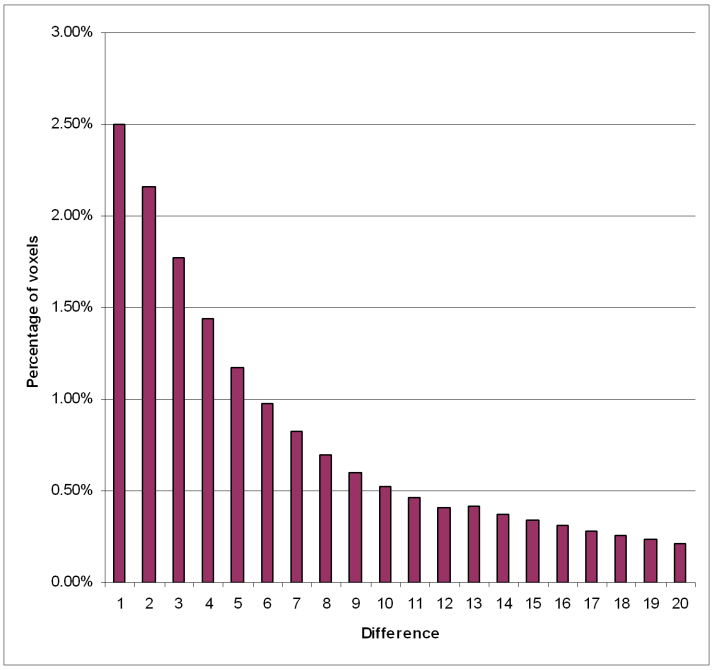


(b) Intensity differences.

Figure 4.25: Differences between brute force and "Fast RBF" software implementation using $H = 0.025$.



(a) Difference image.



(b) Intensity differences.

Figure 4.26: Differences between brute force and "Fast RBF" hardware implementation using $H = 0.025$.

exactly and a further 16% are within 20 intensity points different. The chart in Figure 4.26b shows the count of voxels against intensity difference for mismatched voxels.

Measurements for the hardware "Fast RBF" warp evaluation are as follows

"Fast RBF" hardware evaluation time : **0.646321 seconds**
NMI for warped MRI and CT data : **1.123768** (> 100% of gold standard)

Looking at these figures, it is interesting to note that we appear to have a *higher* NMI score than we did for the equivalent brute force warp evaluation. Intuitively, we suspect this must be misleading – we have already seen with the synthetic data tests that the optimised software routines are less accurate than the software brute force method and that the hardware accelerated techniques suffer further due to loss of numerical precision.

In addition, examination of the difference image, created from the brute force software result and the "Fast RBF" hardware result, shows more significant mismatches of intensity values than the software "Fast RBF". How, then, can this warp appear to be more accurate than the brute force method? We will discuss this in Section 5.1 of Chapter 5.

CT/MRI registration - visual results

Presenting the result of warping and merging multi-modal medical image data in 3D is not easy. However, by first segmenting the bony structures from the CT data we can make the visualisation of the merged data much more easy to examine.

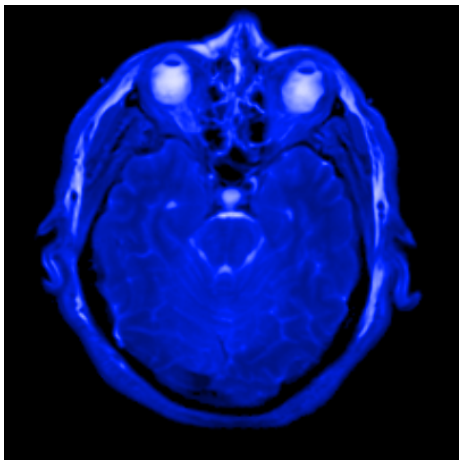
The images in this section use the warped MRI data set that was produced by the "Fast RBF" hardware implementation.

Figure 4.27 shows a 3D reconstruction of the CT data after the bony struc-

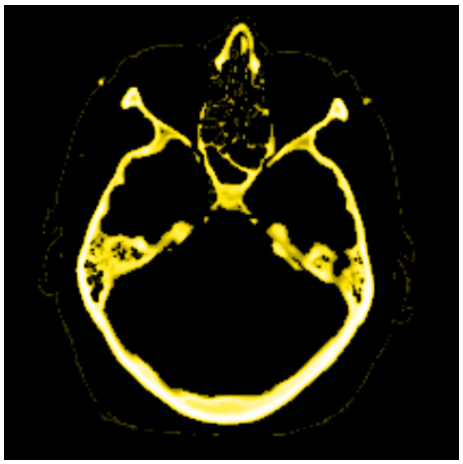
Figure 4.27: A cutaway view of the bony structures segmented from the CT data set.

tures have been segmented using a simple threshold. By merging the warped MRI data with the segmented CT data, we can get a clearer picture of the registration accuracy.

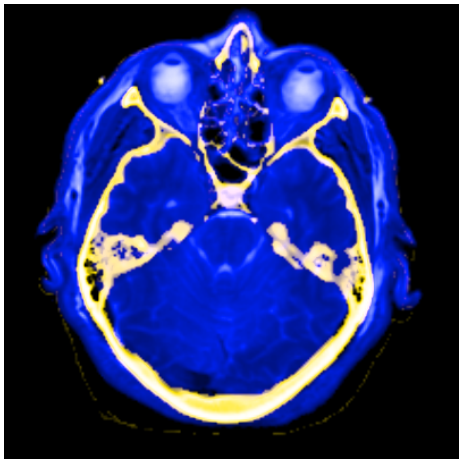
Figure 4.28 shows 2D slices from the warped MRI data set and segmented CT data set, which we have colourised to make it easier to discriminate between them. Figures 4.28a and 4.28b show the warped MRI and CT data respectively, while Figure 4.28c shows the merged data for the same image slice.



(a) Colourised MRI slice.



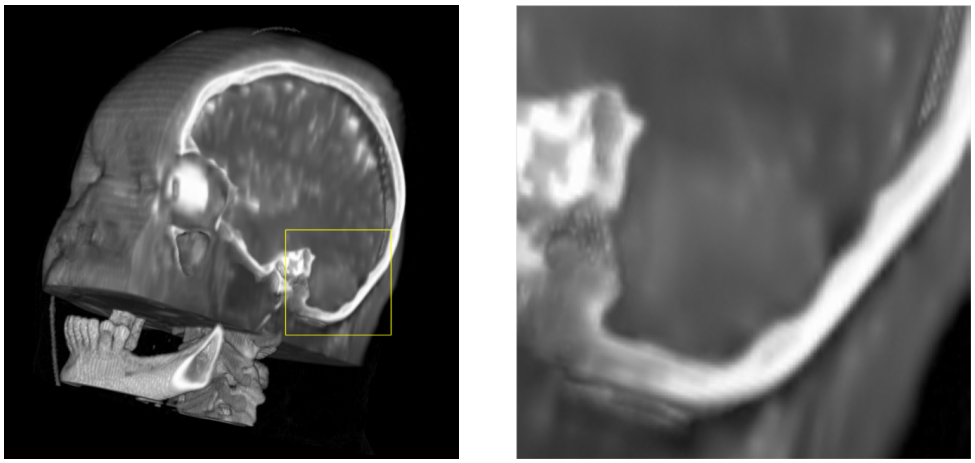
(b) Colourised segmented CT slice.



(c) Merged data.

Figure 4.28: Colourised slices from the warped MRI data and segmented CT data.

Finally, Figure 4.29 shows a 3D view of the combined data sets. The detailed view in Figure 4.29b represents a magnified image of the highlighted rectangular region in Figure 4.29a. The images in these figures were produced using the software *3DView*, which was developed during the course of this research [141].



(a) Combined MRI and segmented CT data in 3D.

(b) Detail view of highlighted area.

Figure 4.29: A cutaway 3D view of the combined MRI and CT data showing a detailed area.

4.3 Summary

Using the results of the synthetic data tests, we can see certain patterns emerging. If we consider the maximum accuracy achieved from each technique and look at the effect of the type of test data used, we can see that the most "accurate" results in terms of NMI score are produced using the data with the regular linear gradient and the non-linear gradient, whilst the least accurate results are obtained using the noisy data.

Figure 4.30 shows this information as a series of bar charts for $\phi(r) = r$. The equivalent data for $\phi(r) = r^2 \log r$ is almost identical. The brute force software implementation consistently scores highest, followed by the software

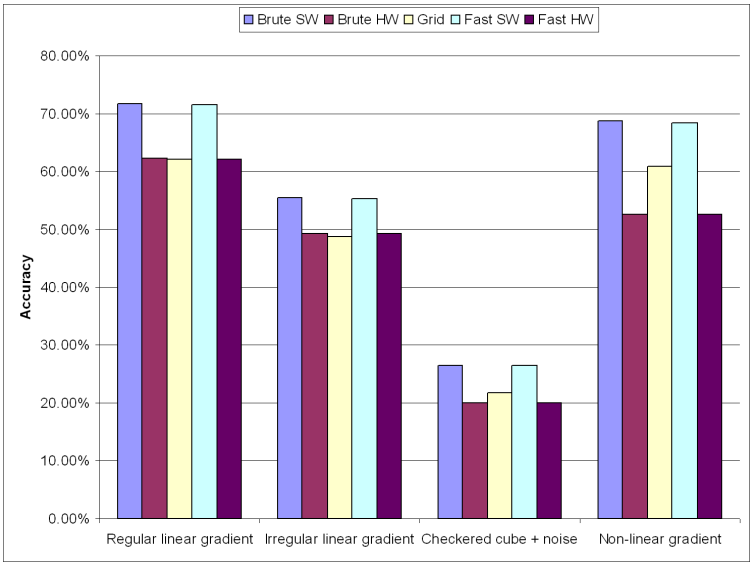


Figure 4.30: Maximum accuracy measured based on NMI score for each technique using $\phi(r) = r$.

implementation of the "Fast RBF" warp evaluation. The brute force hardware implementation is always less accurate than the software equivalent and the hardware implementation of the "Fast RBF" technique matches it across all types of test data. The grid based warp evaluation appears to be more sensitive to the type of test data than the other techniques. It performs better with noisy data and the data with a non-linear gradient than it does with the two data sets containing linear gradients. However, its accuracy never matches that of the software "Fast RBF" technique.

Figure 4.31 shows a scatter plot of accuracy against logarithmic time for all of the synthetic data tests using $\phi(r) = r$. Due to the differences in NMI scores for the synthetic data sets, the scatters form three distinct horizontal bands, as marked by the braces. These indicate – 1) the regular linear gradient and non-linear gradient data, 2) the irregular linear gradient data and 3) the data with random noise.

We can see that the software "Fast RBF" technique appears to be the most

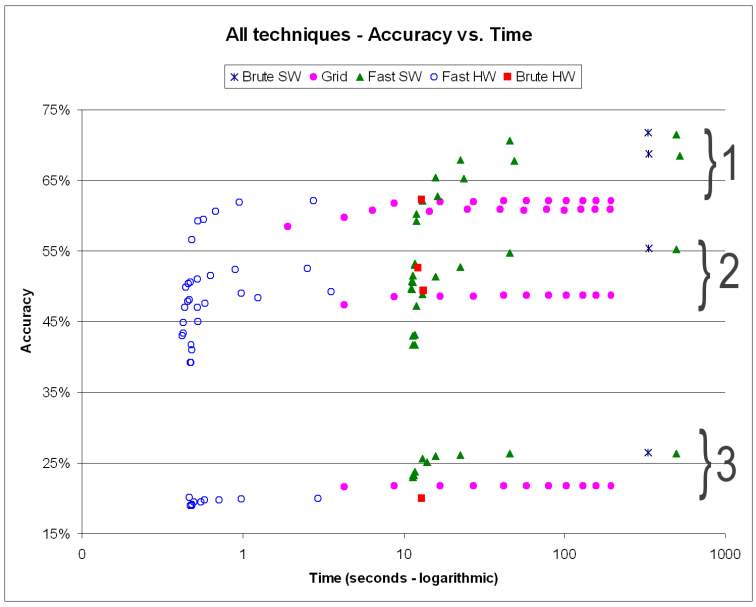


Figure 4.31: Accuracy versus time for all techniques using $\phi(r) = r$.

accurate apart from the brute force warp, and also exhibits a good compromise between run time and accuracy. Remember that the grid based technique *could* be made faster by moving more processing to the GPU but, as we have shown with the other hardware implementations, would possibly lose accuracy. The fastest technique – the hardware implementation of the "Fast RBF" method – is also the least accurate although the performance compared to the brute force software warp is impressive.

The following chapter discusses the issues raised by these results.

Chapter 5

Discussion

In the previous chapter, a number of questions were raised by observing the results, on which we deferred discussion. In this chapter, we will examine these questions in more detail. They are as follows :

- What is the effect of test data design on the NMI metric?
- How does tri-linear interpolation perform compared with nearest-neighbour sampling?
- How does the choice of RBF affect each warp evaluation technique?
- What is the explanation for the apparent limitations of the grid-based evaluation?
- How does the "Fast RBF" technique behave with varying H?
- How does the "Fast RBF" technique behave with varying number of landmarks?

The following sections address each of these items in turn.

5.1 The effect of test data design on the NMI metric

As with all mutual information metrics, NMI is not a perfect measurement of image similarity. The limitations of MI have been recognised and documented by a number of authors [4, 103, 128, 134, 135]. As MI is a statistical measure, it relies on the existence of sufficient information in the image data to provide a consistent response. It is also problematic when faced with *interpolation artifacts* [171], which are almost inevitable in multi-modal image registration due, at the least, to differing voxel dimensions. Interpolation artifacts can appear as a pattern of alternating local maxima and minima in the registration function and prohibit sub-voxel registration accuracy in techniques which rely on maximisation of MI [4, 126, 171].

In the tests using synthetic data, we saw a much weaker NMI response with the 'noisy' data set than we did with the others. See, for example, Figures 4.30 and 4.31 in Section 4.3. However, the full results in Appendix B show similar weak responses for *all* of the other metrics we measured.

For the same reasons, an image with too little structure (e.g. large areas of similar intensity) is just as undesirable as a noisy image. As Andronache describes in [4], this problem is very common in multi-modal image registration because not all tissue types can be seen in all modalities. As with noisy data, featureless data also gives a low MI response but Andronache demonstrated that this response starts to increase as soon as a structureless image overlaps a region of higher structural content.

This is a possible cause of the unexpected NMI result we saw in the CT/MRI registration tests in Section 4.2.1. Figure 4.22 shows the large, homogenous areas in the CT image and the corresponding but more structured MRI equivalent. In this situation it is not unreasonable for there to be some inconsistency reflected in the NMI metric. The tests with synthetic data led us to expect a decrease in accuracy of approximately 5% between the software brute force

technique and the hardware implementation of the "Fast RBF" technique. In practice, the NMI metrics we measured for the two techniques with this data only differed by a small amount.

5.2 Tri-linear interpolation versus nearest-neighbour

In Section 4.1.1 we considered two interpolation options when evaluating a warp function. We looked at the effect of interpolator on the NMI metric for each of the synthetic data sets and found that all except the 'noisy' data gave a higher score when using TRI than they did with NNI.

In Section 5.1, we pointed out one of the limitations of MI as a similarity metric to be its susceptibility to interpolation artifacts. Pluim et al. [127] compared TRI with PVI in this context and, in 2003, Tsao [171] presented a study which, for rigid multi-modal registration, characterised the effects of eight different interpolators on MI metrics including the two we used – nearest neighbour and tri-linear interpolation. Tsao gave strategies for reducing interpolation artifacts, which include jittered sampling and histogram blurring. In addition, Rohde et al. [134, 135] showed that these artifacts are not restricted to MI but are also found in other metrics such as cross correlation and least squares and that interpolation artifacts can also occur in images with no noise [132].

Tsao broadly classified the interpolators into two groups - those that introduce new intensity values into the data, for example TRI, and those that do not – e.g. NNI and PVI. For the noisy synthetic data, where there are random fluctuations in intensity values of neighbouring voxels, NNI will sometimes result in the correct intensity value and sometimes not. However, TRI will almost never result in the correct value as, unless the mapping hits a voxel exactly, the returned intensity will be some weighted average of the intensities in the neighbourhood.

This is possibly why the NMI metric for the NNI result was higher than the

TRI result when we used noisy data. However, this does not necessarily mean that NNI is inherently better in all circumstances – the remaining synthetic data sets showed significant aliasing artifacts when using NNI.

The choice of interpolator for non-rigid medical image registration depends not only on the image data itself – whether noisy, inherently structured or unstructured, etc. – but also on the method used to perform the registration. In all cases the limitations of MI as a similarity measure need to be considered.

5.3 How does the choice of the RBF affect each warp evaluation technique?

We chose to use two RBFs in the synthetic data experiments so that we could see how the different warp evaluation techniques behave with a more complex RBF than the Euclidean distance $\phi(r) = r$ that is the theoretically correct choice for a TPS in three dimensions [138, p. 195].

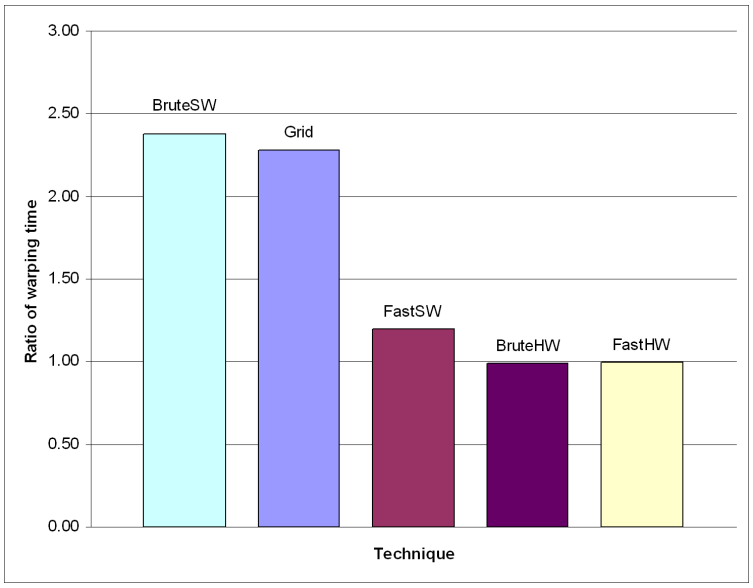


Figure 5.1: Ratios of average warp evaluation time, $\phi(r) = r^2 \log r \,:\, \phi(r) = r$, for each technique.

Figure 5.1 shows the ratio of warping times – the $\phi(r) = r^2 \log r$ time divided by the $\phi(r) = r$ time for the same data – for each of the five warp evaluation techniques. The software brute force and grid based techniques take more than twice as long to execute with the more complex RBF while the software "Fast RBF" technique shows a much reduced difference and the two GPU based implementations appear unaffected – the choice of RBF makes no difference to average run time.

For the software based implementations – the first three bars on the chart – we can postulate that the reduction in differential we see for the optimised techniques might be because there are fewer RBF evaluations occurring and so the time taken for RBF calculations forms a smaller percentage of the overall run time and has less effect on the total.

However, for the GPU based brute force technique, we cannot justify the same argument. As far as the algorithm is concerned, the time spent evaluating the RBF is a significant part of the total time because it is inside the main execution loop as shown in Algorithm 2 in Section 3.11. The explanation probably lies with the parallel nature of the GPU implementation. As each 256×256 voxel image slice is rendered, the graphics card is presented with 65536 TPS evaluations which it can perform in parallel using – in these experiments[1] – 48 individual fragment processors shared between 512 simultaneous pixel threads.

In [170], Trancoso and Charalambous present a study which aims to identify the best ways to exploit the possible gains from GPU based techniques. They concluded that the following three items are necessary to achieve a significant performance increase on GPU versus CPU based algorithms :

- Pass vector data to the graphics card reformatted as a 2D array.
- Use large arrays of data to minimise data transfer to/from the GPU.

[1] ATI Radeon X1900XT PCI-E 512Mb DDR.

- Perform as many operations as possible on the GPU.

In the GPU based warp evaluation techniques – the brute force and "Fast RBF" implementations – we have followed these guidelines, for example by reformatting the landmark positions and matching TPS coefficients so that they can be addressed as 2D textures. We also push as much processing as possible onto the GPU, the main exception being the anterpolation stage of the "Fast RBF" technique, which must remain CPU based due to current limitations of the graphics card[2].

Parallel evaluation of RBF functions has the same practical effect as the software optimisations – it lessens the proportion of execution time spent in the RBF function itself. This may be part of the reason why there is no measurable run time difference between the two RBFs in the GPU based implementations. Another possible reason is the fact that the GPU code is written in an assembler language and is highly optimised for vector based operations. This may also mitigate the effect of the $\phi(r) = r^2 \log r$ RBF on execution time.

5.4 Limitations of the grid-based evaluation

The grid based warp appears to have an upper limit on accuracy, if judged by the NMI metric against the brute force gold standard. Considering all synthetic test data sets, the grid based warp showed a maximum accuracy of $94.65 \pm 1.60\%$ with the smallest grid size (approximately 2×2 voxels). However, in [101], the authors report 99% MI accuracy when the technique is used with real medical images.

One possible reason for this discrepancy arises when we consider the unexpected NMI score using the "Fast RBF" hardware implementation with real medical image data. Here we obtained a result that was seemingly better than the brute force software warp, as we discussed in Section 5.1. We discovered

[2]Scatter versus gather [121].

that it can be difficult to assess the accuracy of image registration using a MI based metric unless using synthetic data with known ground truth.

Due to these difficulties, the tests of the grid-based method had to focus more on accuracy than run time performance. We already knew from the brute force GPU based warp evaluation that the reduced floating point precision compared to a CPU based technique could cause degradation in accuracy. Therefore, we decided to implement the TPS evaluations of the grid nodes in software, using the same double precision routines we used for the brute force technique. In doing this, any differences in accuracy must be due to some other factor than lack of numerical precision.

Looking at Figure 4.6b in Section 4.1.3, it appears that the grid based warp evaluation reaches a plateau in terms of accuracy at a grid size of 63 rows by 63 columns. In the 256×256 voxel 2D slices, this places the grid nodes such that they cover exactly 4×4 voxels. Coarser grids give increasingly less accurate results.

Although it is difficult to find an explanation of this 4×4 voxel accuracy plateau in theoretical terms, we can at least observe that this was a consistent feature of all of the synthetic data tests, across all types of test data and with both RBFs. This is an area which requires further study.

5.5 Behaviour of the "Fast RBF" technique with varying H

Figure 4.8 in Section 4.1.4 and Figure 4.10 in Section 4.1.5 both show how the warp accuracy – based on the NMI score – varies with different values of the parameter H, which controls the "Fast RBF" grid resolution. Here we noted that there is an apparent decline in accuracy around $0.045 < H < 0.095$, particularly for $\phi(r) = r^2 \log r$.

In [101], Livne and Wright give specific details of accuracy and complexity for a number of smooth RBFs, which makes it theoretically possible to calculate an optimal value of H in terms of accuracy. However, the kernels we use – $\phi(r) = r$ and $\phi(r) = r^2 \log\ r$ – are only piecewise smooth and are non-decaying functions, so we cannot make the same calculation. This makes it currently unwise to attempt to explain these observed anomalies without further work in this area.

5.6 Behaviour of the "Fast RBF" technique with varying number of landmarks

Another observation concerning the "Fast RBF" technique, which we highlighted in Section 4.1.6 was the presence of 'spikes' in the run times of both the software and hardware implementations when we varied the number of landmarks. This was most noticeable for coarser grid resolutions ($H > 0.055$), as we can see in Figures 4.17 and 4.20.

It is possible that these spikes are random fluctuations in run time, caused by background processes running on the CPU, which are only visible when the time scale on the y axis covers a short period. However, although the spikes appear often in the 1.5 second range of Figure 4.17, they do *not* appear with similar frequency in the 1.5 second range of Figure 4.18 or the 0.75 second range of Figure 4.19. If the spikes *are* caused by sporadic background activity, then by running this set of tests multiple times and plotting the average results, we would expect to see the effect of any random fluctuations smoothing out.

Figure 5.2 shows averaged results from three test runs, still showing anomalies – some of which coincide exactly in the three runs.

It is worth noting that, in all of the other run time measurements against varying numbers of landmarks, we have not encountered this type of 'noisy'

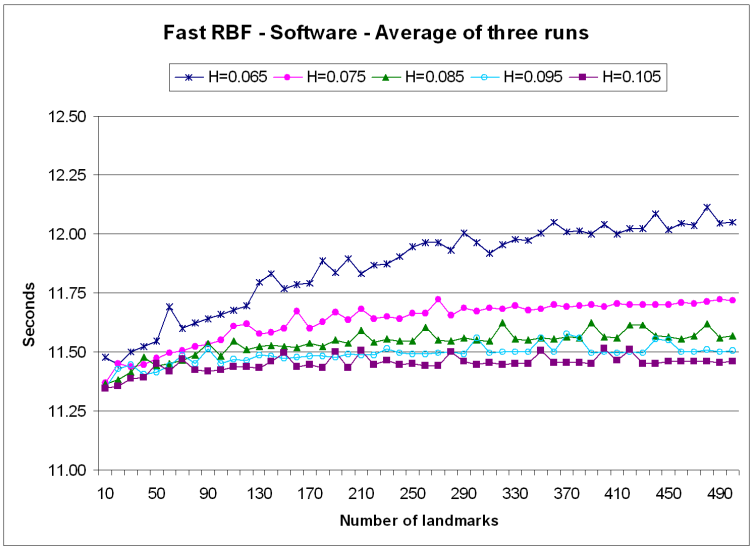


Figure 5.2: Average effect of the number of landmarks on run time of the "Fast RBF" software implementation – $H = 0.065 \ldots 0.105$.

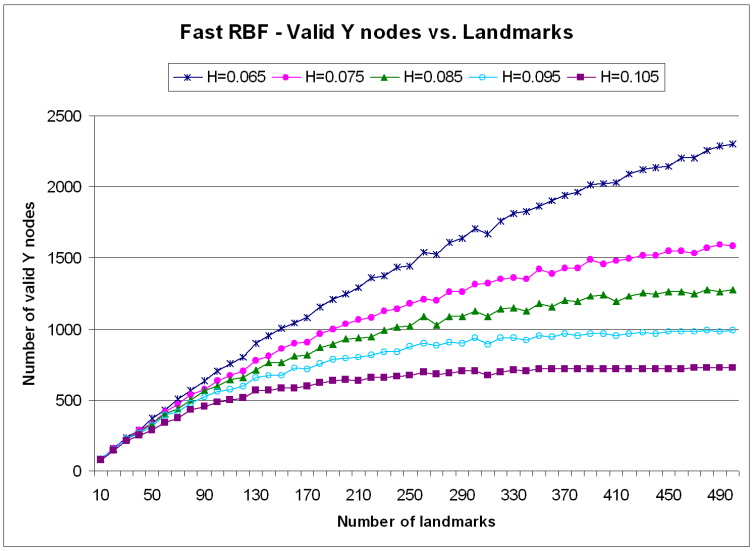


Figure 5.3: Effect of the number of landmarks on valid $\mathbf{Y}$ nodes in the "Fast RBF" implementations – $H = 0.065 \ldots 0.105$.

response – even at shorter time scales, and have not needed to perform multiple runs to obtain averaged results. This effect appears to be present only with the "Fast RBF" evaluations and is most significant where $H > 0.055$.

In Section 4.1.4, we noted that the run time of the "Fast RBF" algorithm is influenced by the number of nodes in the grid $\mathbf{Y}$ which have non-zero values of $\Lambda(\mathbf{Y_J})$ following the anterpolation stage. This in turn is determined by the coarseness of the grid – the parameter H – and the density of the landmarks. If more than eight landmarks fall within the same grid cell, there will be a reduction in the number of TPS evaluations needed, but if less than eight fall within a cell, the number will increase.

To produce the sets of landmarks for the tests in Section 4.1.6, we selected n landmarks at random from a larger set where $n = 10, 20, 30, \ldots, 500$. As a result, we cannot predict or control the spatial distribution of landmarks in each data set. In fact, it would have been wrong to do so. However, we can analyse the landmark sets to record, for each set, the number of 'valid' nodes created in the grid $\mathbf{Y}$ for each value of H.

Figure 5.3 shows this information for the range of H values in Figure 5.2. The chart shows that increasing the number of landmarks may sometimes increase and sometimes decrease the number of valid $\mathbf{Y}$ nodes, which will influence the overall warp execution time in a similar manner.

To see how the variation in the number of $\mathbf{Y}$ grid nodes influences the observed run times, we can correlate this information separately for each value of H to see if it contributes to the spikiness of the run times in Figure 5.2.

Figure 5.4 shows the correlation[3] between the observed run times with the number of landmarks (light pink bars) and the number of valid $\mathbf{Y}$ nodes (dark blue bars). The chart shows that there is a strong correlation between the run time and the number of valid $\mathbf{Y}$ nodes as well as between run time and the

[3]For this chart we used Kendall's tau rank correlation coefficient.

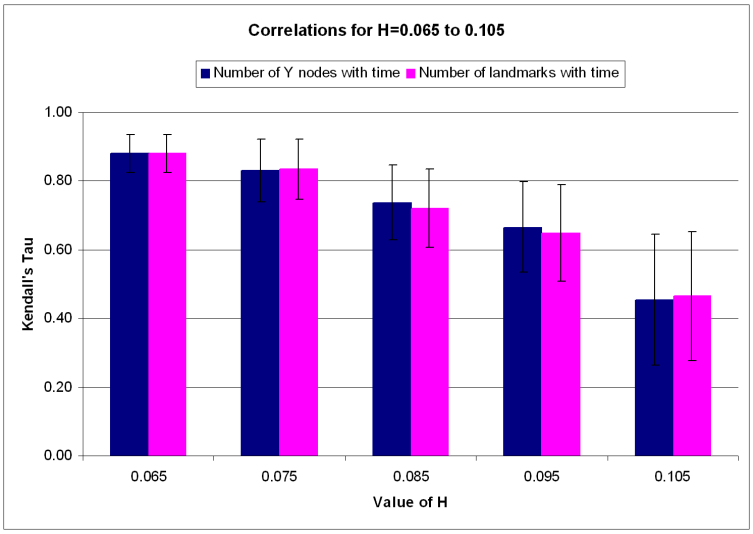


Figure 5.4: Correlations between valid $\mathbf{Y}$ nodes and landmark numbers with time in the "Fast RBF" implementations – $H = 0.065 \ldots 0.105$.

number of landmarks, which is what we expect. However there is no statistical evidence that the number of $\mathbf{Y}$ nodes has a stronger influence on observed run times.

Chapter 6

Conclusions and future work

6.1 Conclusions

We conclude the work presented in this thesis by reiterating the questions posed in Section 2.6.1 :

How much can we optimise the run time of an RBF based warp evaluation?

The experimental tests have indicated that the "Fast RBF" warp evaluation techniques are more accurate than the grid-based method of Levin et al. with the synthetic test data and can produce results which are very close to the brute force warp evaluation when measured by NMI.

Faster run times will always compromise warping accuracy. It is not possible to give a 'fastest' figure for warp time unless we first decide an acceptable level of accuracy for the particular application. As an example, we might assume that a warp that is 90% as accurate as the brute force result is acceptable and we could note the following[1].

By using the "Fast RBF" software implementation on 256^3 data and comparing with the brute force evaluation, we reduced the execution time for

[1] All results reported are averages across all synthetic test data.

$\phi(r) = r$ from 332 seconds to 13 seconds (using $H = 0.055$) while retaining $> 90\%$ accuracy compared to the brute force result.

Using the hardware "Fast RBF" implementation on the same data, we achieved an execution time of 0.81 seconds (using $H = 0.035$) while retaining $> 80\%$ accuracy. This represents a 400-fold improvement in warp evaluation time.

For $\phi(r) = r^2 \log r$ with the "Fast RBF" software implementation, the execution time was reduced from 790 seconds to 12 seconds (using $H = 0.105$). The hardware "Fast RBF" implementation reduced execution time to 0.5 seconds, nearly 1600 times faster than the brute force method.

What effect do such optimisations have on warping accuracy?

The results of the "Fast RBF" software implementation implied that warping accuracy of $> 98.8\%$ is achievable (compared to the brute force warp), with an execution time of 46 seconds for $\phi(r) = r$ with $H = 0.025$ – a seven-fold improvement in run time. For $\phi(r) = r^2 \log r$, again with $H = 0.025$, run time was 71 seconds – an 11-fold improvement.

However, although warping time decreased monotonically with increasing H, the associated accuracy did not follow a monotonic pattern, particularly for $\phi(r) = r^2 \log r$ – see Figures 4.8 and 4.10. This makes it difficult to balance accuracy and time, although – as we mentioned in Section 3.4 – $\phi(r) = r^2 \log r$ is not optimal as an energy minimising function for a second order energy potential in three dimensions and is only C^1 continuous in the origin.

The hardware implementation of the "Fast RBF" technique displayed similar characteristics to its software counterpart. However, the best possible accuracy appeared to be capped at just under 93% of the brute force evaluation. We showed, using a GPU implementation of the brute force technique, that this limitation is likely to be due to the reduced floating point precision used

for GPU calculations.

How do these optimised techniques behave under different inputs?

For the smaller values of H used in these tests ($H = 0.015 \ldots 0.055$), the run time of the "Fast RBF" warp evaluation displayed a linear relationship with the number of landmarks defined.

However, for larger values of H ($H = 0.065 \ldots 0.105$), the run time showed some spikes when plotted against the number of landmarks. This pattern appeared for the same values of H in the software and hardware implementations and was not reduced by repeating runs and plotting average times.

Are the optimised methods accurate and/or fast enough for AR applications?

In an intraoperative situation, the time to perform a non-rigid warp evaluation should not be so long that the operation itself is compromised or the patient (or surgeon) is adversely affected by any delay.

The absolute time will inevitably be dependent on the nature of the operation, although timescales measured in a few minutes or – better – a few seconds would intuitively appear satisfactory.

Balancing speed with accuracy is a necessary compromise and one on which it is not possible to be dogmatic. In the tests with real CT and MRI data, each of 256^3 voxels, we achieved visually acceptable results using the hardware "Fast RBF" technique with a warp evaluation time of just over 0.6 seconds. From the results of tests with the software "Fast RBF" technique using synthetic data, we can infer an improved registration accuracy ($> 99\%$) in less than 50 seconds for data sets of this size. These timescales are within a range suitable for intraoperative applications.

6.2 Summary of contributions

This thesis has made the following contributions :

1. Extended the "Fast RBF" evaluation formulation introduced by Livne and Wright to three dimensions and applied it to mono-modal synthetic data and multi-modal medical image data.

2. Developed a software implementation of the "Fast RBF" algorithm and demonstrated its speed and accuracy using synthetic data and real medical image data.

3. Developed a hardware implementation of the "Fast RBF" algorithm and demonstrated its speed and accuracy using synthetic data and real medical image data.

4. Demonstrated the ability to balance speed and accuracy in the "Fast RBF" technique by adjusting a single run time parameter.

5. Demonstrated the execution time behaviour of the "Fast RBF" technique when supplied with an increasing number of inputs.

6. Demonstrated the importance of appropriate design of synthetic test data when assessing the accuracy of optimised warp evaluation techniques.

6.3 Future research

This work has shown that the techniques to accelerate evaluation of RBF based warping functions have potential for non-rigid medical image registration. A number of areas for future research have been identified in the previous chapter. These are as follows :

- In the "Fast RBF" method, we found that it was not straightforward to determine the optimal value of H for a piecewise smooth non-decaying

RBF. Further work which examines the behaviour of these, and other, basis functions would be useful to develop this acceleration technique.

- With larger values of H, we noted that the run time of the "Fast RBF" warp evaluation did not vary smoothly with increasing numbers of landmarks. Suspicions that the spikes in run time were due to landmark distribution rather than quantity could not be statistically shown. More investigation of the effects of varying inputs to these methods is required to gain an understanding of the optimal conditions for their application.

- Test results showed an apparent ceiling on registration accuracy for the grid based warp, which we were not able to explain. This area requires further study.

- The fastest warping times were achieved using GPU based implementations but were limited in accuracy due to hardware restrictions. The development and application of efficient techniques for GPU based double precision computation would mitigate this effect.

- We have demonstrated that appropriate design of synthetic test data – which takes into account the warping technique, the similarity metric and the target application – can make it easier to reliably compare results from different warping methods. This idea could be extended to other application areas and to different classes of non-rigid registration techniques, for example parametric methods using B-splines or RBFs with local support, or the various non-parametric methods described in Section 2.3.6.

Appendix A

Software design

A.1 Experimental software design

Figure A.1 shows a screenshot of the experimental software developed to compare warping speed and quality using different implementations.

The application is written in C++ using Microsoft Visual Studio, MFC and DirectX. The top left quadrant shows a 2D slice through the source data set and the top right quadrant shows a 2D slice through the target data set. The lower left quadrant allows us to preview a grid-based warp of a single data slice. The control panel in the lower right allows the user to scroll through these 2D views and to select which plane is currently showing. Figure A.2 shows an enlarged representation of the control panel, with the controls used for scrolling through source and target data, removing landmarks, selecting warping algorithm and supplying run-time parameters and options.

A.2 Manual landmark identification

The top two views in figure A.1 are used to identify matching landmarks in the two data sets. The relevant 2D slices are selected in the left and right quadrants and the user places a landmark by clicking with the left mouse button on either

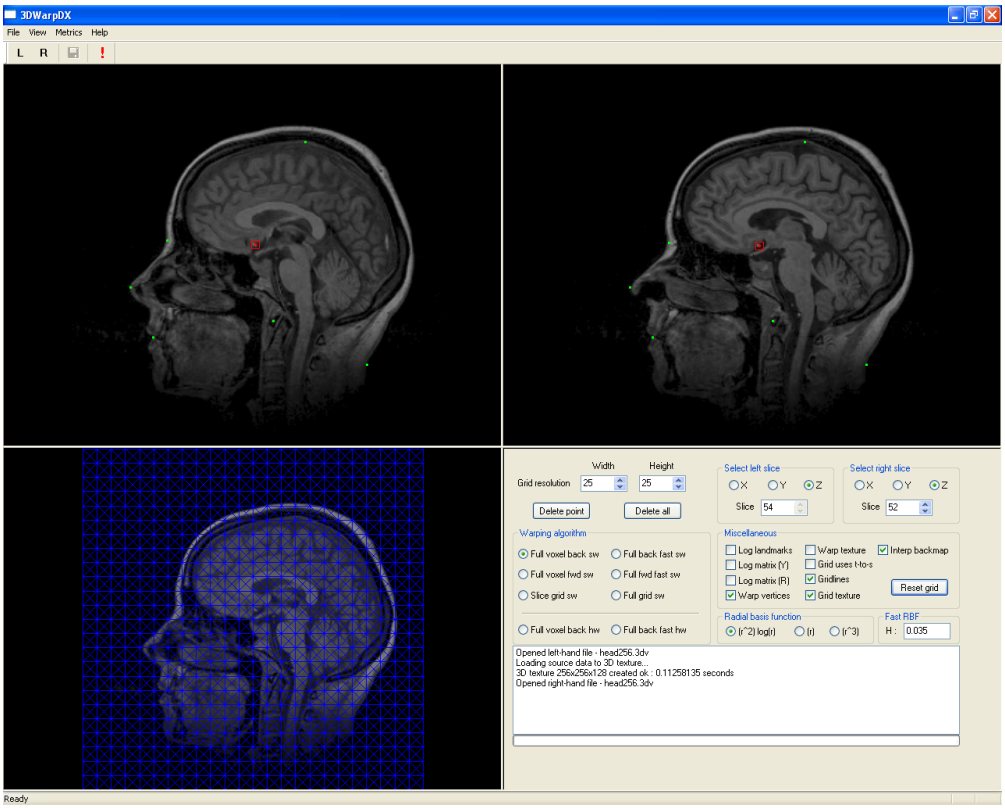


Figure A.1: Screenshot of 3DWarpDX, the experimental software

image. A matching point is automatically placed on the other image and each point may be moved to the ideal position independently by click/drag with the mouse.

To assist in identification of matching points, as the mouse moves near a point, it is automatically highlighted with a green border, together with its corresponding partner in the other image. Clicking once on an existing landmark will highlight the pair of points in red so that the user can delete the pair if necessary using the button on the control panel. Figure A.3 shows a portion of the application display showing the landmark matching process.

Once all landmark points have been placed, the set of points may be saved to a file via an option on the File menu. This allows repeat runs with the same data without the need to re-place landmarks.

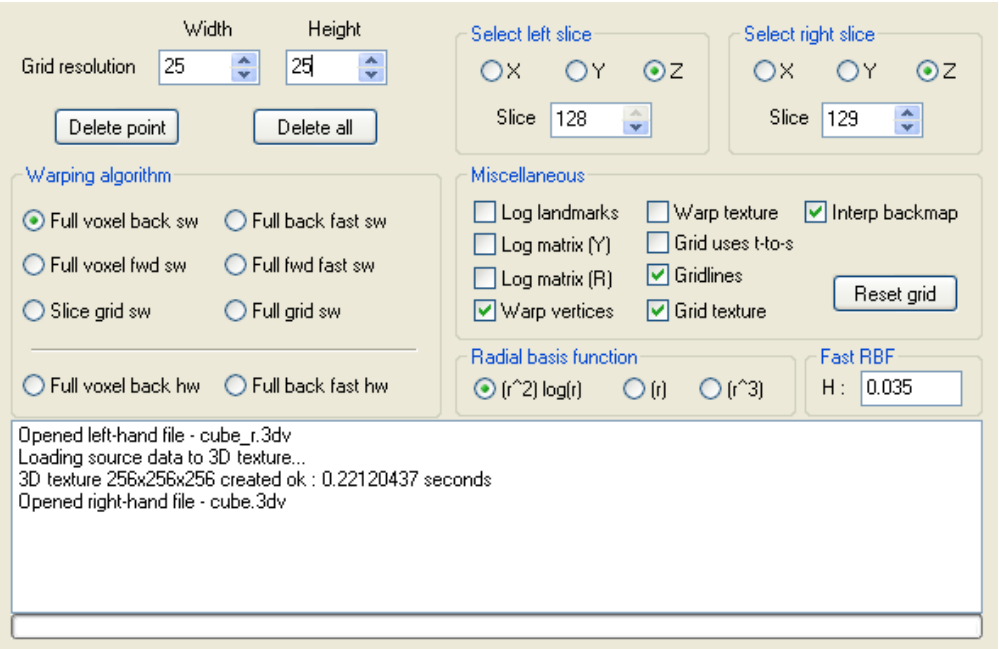


Figure A.2: The experimental software control panel

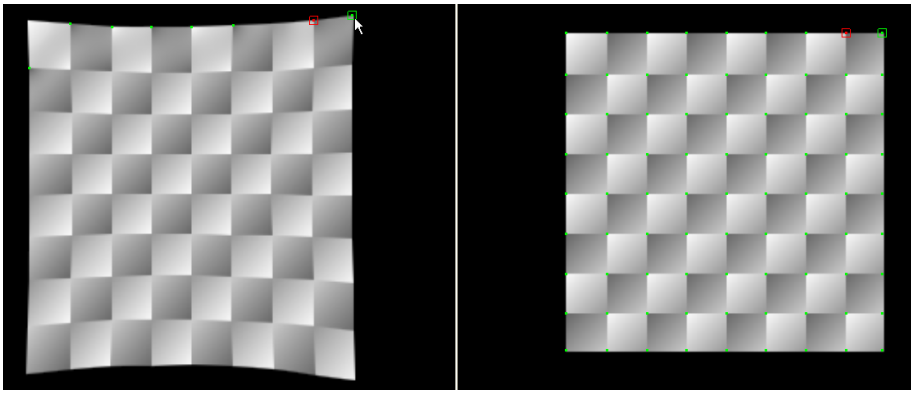


Figure A.3: A screenshot section showing matching landmark points on source and target images

Appendix B

Full experimental results

The full experimental results for the warp evaluation tests using synthetic test data are presented in the following sections. All tests used back-mapping and all times are given in seconds. Unless otherwise shown, separate tables are given for different interpolation options.

B.1 Results for $\phi(r) = r$

B.1.1 Regular linear gradient synthetic test data

Brute force warp evaluation - software implementation

The gold standard software results :

Interp.	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
TRI	2.349523	331.373566	20.052744	15404513	2.615712	-1.497450	1.717397	0.998777	0.907467
NNI	2.375349	327.192413	24.619551	13742805	3.654536	-1.512251	1.585659	0.998475	0.756415

Brute force warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
2.323861	12.887721	126.865898	8807569	3.402723	-1.433044	1.622725	0.991976	0.514413

Grid based warp evaluation

The following tests used tri-linear interpolation :

Grid	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
138	2.380048	193.918043	60.263374	11865201	3.311904	-1.475847	1.621600	0.996280	0.576432
125	2.383679	157.723965	60.202953	11866857	2.989006	-1.551790	1.621512	0.996283	0.576532
113	2.354627	129.345930	60.290417	11866263	3.159882	-1.546677	1.621508	0.996280	0.576514
100	2.365657	101.633105	60.217644	11866706	3.681778	-1.526102	1.621500	0.996283	0.576530
88	2.409841	79.466496	60.256607	11867658	3.502945	-1.497885	1.621495	0.996278	0.576599
75	2.353977	58.329029	60.244633	11867819	2.241484	-1.587103	1.621304	0.996281	0.576652
63	2.344851	41.957636	60.220158	11869562	2.423157	-1.504426	1.621220	0.996283	0.576785
50	2.356552	27.305252	60.257168	11872096	2.793593	-1.488383	1.620615	0.996281	0.577060
38	2.360429	16.825287	60.300396	11875918	3.316882	-1.501494	1.620045	0.996277	0.577428
25	2.351770	8.729403	60.406940	11893509	3.142670	-1.440678	1.617260	0.996275	0.579137
13	2.357576	4.262032	61.750652	11962917	3.245350	-1.522124	1.597898	0.996186	0.588755

"Fast RBF" warp evaluation - software implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.372517	494.232544	20.250599	15392330	2.341957	-1.499296	1.715465	0.998778	0.906725
0.025	2.361302	45.793739	20.750439	15344198	4.713700	-1.504982	1.706122	0.998748	0.902317
0.035	2.354435	22.560955	21.696177	15161427	3.718278	-1.493454	1.679303	0.998688	0.887105
0.045	2.334354	15.864787	23.132448	14950712	3.750756	-1.557090	1.653506	0.998602	0.869009
0.055	2.332495	13.103786	25.961502	14536341	4.436656	-1.555585	1.620976	0.998417	0.831073
0.065	2.310550	12.063823	29.994543	14448825	4.212432	-1.526293	1.602607	0.998176	0.827166
0.075	2.310906	11.680119	36.780121	12810417	3.890687	-1.553591	1.529716	0.997741	0.689067
0.085	2.313205	11.694855	41.057610	13368754	3.384850	-1.613864	1.532190	0.997481	0.749611
0.095	2.310031	11.462649	46.928150	12941531	4.650326	-1.520300	1.506749	0.997110	0.712090
0.105	2.309155	11.411191	45.816689	12917741	3.892295	-1.497397	1.506662	0.997174	0.707015

The following tests used nearest neighbour interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.340851	489.968811	24.656397	13741035	3.754997	-1.511065	1.585530	0.998474	0.756309
0.025	2.354538	42.154884	24.909294	13736900	3.792902	-1.592857	1.585054	0.998459	0.756031
0.035	2.350775	19.281717	25.590912	13716107	4.258894	-1.574508	1.582756	0.998417	0.754880
0.045	2.335472	12.399343	26.822565	13685396	4.537364	-1.534755	1.579619	0.998338	0.753082
0.055	2.333610	9.638347	29.968922	13635292	4.125027	-1.582356	1.576479	0.998143	0.749971
0.065	2.340024	8.598971	34.354515	13572623	3.235048	-1.658665	1.568091	0.997871	0.746223
0.075	2.337739	8.178324	42.603157	13060962	4.438252	-1.508802	1.536598	0.997351	0.717940
0.085	2.327874	8.070889	47.260742	13239596	3.969659	-1.512022	1.535432	0.997071	0.727345
0.095	2.334364	7.987331	55.369068	13042727	3.214167	-1.605390	1.520115	0.996567	0.715885
0.105	2.340537	7.930209	55.297810	13017904	2.810020	-1.604648	1.518683	0.996574	0.713297

"Fast RBF" warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.316180	2.749139	127.144791	8810329	2.900402	-1.449952	1.620864	0.991959	0.514448
0.025	2.310702	0.954813	128.533966	8806998	2.599971	-1.491014	1.618305	0.991870	0.514394
0.035	2.308746	0.680023	131.307587	8807838	3.328390	-1.429281	1.606666	0.991689	0.514580
0.045	2.308443	0.568055	129.673080	8844587	3.810427	-1.355180	1.595275	0.991796	0.514676
0.055	2.317935	0.528603	145.817307	8775759	4.429440	-1.425954	1.592312	0.990772	0.514385
0.065	2.327933	0.484454	139.112549	8836957	3.797102	-1.448963	1.566772	0.991206	0.514640
0.075	2.306459	0.473183	111.539719	9540854	3.379082	-1.439521	1.506854	0.992966	0.526206
0.085	2.299965	0.457530	154.263123	9082926	3.572644	-1.361175	1.503426	0.990259	0.520459
0.095	2.306144	0.464069	154.389313	9261439	3.175537	-1.474586	1.481622	0.990260	0.523539
0.105	2.298075	0.455741	155.318146	9178681	3.020954	-1.359375	1.479090	0.990190	0.520672

B.1.2 Irregular linear gradient synthetic test data

Brute force warp evaluation - software implementation

The gold standard software results :

Interp.	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
TRI	2.362640	333.835164	46.958843	13794750	1.952052	-1.329697	1.553860	0.997639	0.806856
NNI	2.465949	327.035370	55.217781	13080376	2.567493	-1.360311	1.492595	0.997175	0.720960

Brute force warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
2.357186	13.285579	310.795227	8950981	1.735395	-1.277285	1.493154	0.983875	0.517697

Grid based warp evaluation

The following tests used tri-linear interpolation :

Grid	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
138	2.369475	193.907469	142.972336	10925051	2.221509	-1.306882	1.487330	0.992655	0.545123
125	2.390018	157.730304	142.904602	10926332	2.051779	-1.367469	1.486984	0.992655	0.545206
113	2.353006	129.344308	142.995117	10925291	2.235125	-1.354113	1.487235	0.992651	0.545130
100	2.365638	101.633087	142.918137	10926322	2.476452	-1.334867	1.487073	0.992658	0.545201
88	2.389846	79.446502	142.955811	10927255	2.489680	-1.383748	1.487182	0.992654	0.545240
75	2.351630	58.326682	142.935791	10926476	1.876327	-1.415038	1.487041	0.992655	0.545245
63	2.352625	41.965410	142.932480	10926639	1.982229	-1.320202	1.486966	0.992656	0.545249
50	2.331013	27.279713	142.981262	10927631	2.589956	-1.315126	1.486464	0.992651	0.545382
38	2.352016	16.816875	143.080414	10928253	2.836976	-1.330971	1.486401	0.992646	0.545437
25	2.360546	8.738179	143.343079	10933353	2.683096	-1.368461	1.484525	0.992631	0.546075
13	2.369273	4.273729	147.050308	10962588	2.510492	-1.335861	1.473716	0.992448	0.550870

"Fast RBF" warp evaluation - software implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.355670	493.254517	47.402126	13786709	2.945127	-1.329987	1.553117	0.997616	0.806511
0.025	2.354691	45.585709	48.576015	13757342	4.102522	-1.336539	1.547287	0.997555	0.803787
0.035	2.347727	22.513794	51.062649	13614777	2.923309	-1.338571	1.527772	0.997435	0.790770
0.045	2.325602	15.831116	54.358501	13492166	3.628732	-1.377168	1.513538	0.997264	0.780146
0.055	2.334067	13.088737	59.687492	13206733	3.192368	-1.364025	1.489043	0.996992	0.750933
0.065	2.320339	12.043397	68.763916	13073261	3.251038	-1.347961	1.472544	0.996528	0.742959
0.075	2.330864	11.634527	92.168167	12019065	2.174287	-1.336613	1.430672	0.995309	0.653887
0.085	2.315074	11.467153	103.320618	12388760	2.832463	-1.417168	1.430347	0.994738	0.689494
0.095	2.328668	11.412859	110.595596	12267371	3.292900	-1.339128	1.417182	0.994356	0.680985
0.105	2.336036	11.696194	112.290146	12253011	2.412324	-1.334953	1.417139	0.994269	0.678255

The following tests used nearest neighbour interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.363455	495.023987	55.261024	13080383	2.958348	-1.382144	1.492578	0.997173	0.720956
0.025	2.345081	42.108280	55.772667	13078554	3.634188	-1.426638	1.492121	0.997145	0.720854
0.035	2.350495	19.095144	57.432816	13068163	3.192837	-1.416303	1.490235	0.997063	0.720464
0.045	2.353552	12.393270	60.400295	13049061	2.975712	-1.395801	1.487660	0.996908	0.719446
0.055	2.350919	9.648543	65.677254	13007497	2.826253	-1.405918	1.483400	0.996638	0.717227
0.065	2.359740	8.597961	75.700897	12954406	3.050709	-1.477128	1.475900	0.996121	0.714107
0.075	2.358448	8.190183	102.597382	12614053	3.453309	-1.359150	1.454823	0.994720	0.696125
0.085	2.347873	8.062902	115.308044	12717312	2.831404	-1.374779	1.450671	0.994072	0.700919
0.095	2.350632	7.978933	124.680595	12597363	2.584180	-1.413147	1.440587	0.993586	0.694487
0.105	2.355388	8.106890	128.832245	12590614	2.767558	-1.420338	1.440250	0.993375	0.692853

"Fast RBF" warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.341194	3.535046	310.817749	8955438	2.531791	-1.345747	1.492156	0.983874	0.517765
0.025	2.335294	0.973264	313.151642	8953676	3.386749	-1.340690	1.490224	0.983747	0.517911
0.035	2.359863	1.238837	320.747192	8940396	3.659957	-1.291791	1.483191	0.983336	0.518253
0.045	2.342167	0.580914	315.168762	8977865	3.273073	-1.296363	1.476133	0.983636	0.518730
0.055	2.338594	0.519875	353.803375	8906345	4.347034	-1.335528	1.469845	0.981605	0.517808
0.065	2.348237	0.528052	331.779175	8968432	3.739660	-1.345751	1.450174	0.982768	0.518780
0.075	2.344851	0.479194	263.593414	9515538	3.184233	-1.285622	1.417063	0.986358	0.529092
0.085	2.353009	0.484377	366.985352	9071795	3.876471	-1.300467	1.410164	0.980906	0.521923
0.095	2.356097	0.472069	359.929688	9251860	3.593153	-1.345021	1.392145	0.981301	0.525481
0.105	2.346526	0.475835	370.394775	9210612	2.285831	-1.274955	1.392510	0.980757	0.523799

B.1.3 Checkered cube with random noise synthetic test data

Brute force warp evaluation - software implementation

The gold standard software results :

Interp.	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
TRI	2.360610	331.353333	127.108925	9328357	2.472743	-1.280557	1.264702	0.991891	0.534591
NNI	2.354361	327.078644	109.370583	9753855	3.267127	-1.339762	1.277094	0.992986	0.552193

Brute force warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
2.343023	12.905877	404.136292	8841003	2.933362	-1.146813	1.199441	0.973597	0.518580

Grid based warp evaluation

The following tests used tri-linear interpolation :

Grid	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
138	2.373630	193.911624	253.761520	8972887	2.112984	-1.231810	1.217234	0.983571	0.524191
125	2.371551	157.711837	253.710587	8972546	2.842460	-1.181489	1.217262	0.983572	0.524167
113	2.361046	129.352349	253.781006	8972574	2.471511	-1.175042	1.217242	0.983570	0.524171
100	2.350702	101.618150	253.721634	8972541	3.594377	-1.199043	1.217252	0.983569	0.524166
88	2.428383	79.485039	253.753113	8972852	2.219838	-1.216137	1.217265	0.983573	0.524187
75	2.359702	58.334754	253.742905	8972971	2.674824	-1.242370	1.217247	0.983572	0.524195
63	2.380901	41.993686	253.740753	8972511	2.820373	-1.178613	1.217268	0.983572	0.524166
50	2.354081	27.302781	253.771622	8972891	2.311805	-1.248085	1.217216	0.983573	0.524202
38	2.373153	16.838011	253.858124	8972766	2.708368	-1.183878	1.217175	0.983562	0.524191
25	2.360703	8.738336	254.101807	8971714	1.806219	-1.169294	1.217053	0.983548	0.524130
13	2.380582	4.285038	257.383850	8969715	3.256285	-1.174290	1.215858	0.983336	0.524108

"Fast RBF" warp evaluation - software implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.358737	492.318298	127.564560	9323592	3.329511	-1.279526	1.264369	0.991862	0.534309
0.025	2.352375	45.603939	128.815781	9312827	4.464308	-1.302928	1.263534	0.991785	0.533693
0.035	2.356208	22.562088	131.793640	9253599	3.118101	-1.285656	1.261659	0.991591	0.530663
0.045	2.321992	15.858770	135.391785	9245553	4.282638	-1.337402	1.259585	0.991354	0.530444
0.055	2.322564	13.107615	141.426224	9150609	4.436867	-1.323858	1.256230	0.990950	0.527322
0.065	2.363482	13.906859	151.073410	9130793	3.700143	-1.281776	1.251453	0.990313	0.526630
0.075	2.321589	11.681573	181.139755	9032637	4.125660	-1.285818	1.237437	0.988334	0.524030
0.085	2.323447	11.523153	194.174454	9030852	3.566306	-1.306761	1.232656	0.987472	0.524395
0.095	2.330523	11.462830	199.358536	9181633	4.108850	-1.259705	1.232165	0.987130	0.530147
0.105	2.335913	11.428257	202.101257	9096899	4.545603	-1.237642	1.230503	0.986937	0.526740

The following tests used nearest neighbour interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.351057	489.343933	109.467201	9754011	2.912083	-1.334924	1.277042	0.992974	0.552210
0.025	2.364119	42.136120	110.088669	9753780	4.504101	-1.381322	1.276756	0.992937	0.552206
0.035	2.352633	19.097809	112.519196	9754499	4.583532	-1.371448	1.275415	0.992774	0.552303
0.045	2.356896	12.398351	116.166183	9752472	5.068680	-1.325312	1.273669	0.992537	0.552262
0.055	2.354249	9.646543	122.293396	9746678	5.544571	-1.366303	1.271270	0.992138	0.552012
0.065	2.343453	8.778589	134.044540	9738663	2.843383	-1.438701	1.266494	0.991372	0.551778
0.075	2.352368	8.183869	172.136948	9717561	4.045371	-1.308649	1.250988	0.988843	0.551691
0.085	2.351401	8.066149	188.480347	9705649	5.038779	-1.274166	1.245867	0.987763	0.551384
0.095	2.358431	8.011514	199.309021	9697347	4.121992	-1.329991	1.243408	0.987054	0.551198
0.105	2.354373	7.930961	204.160065	9688768	2.699624	-1.355432	1.241451	0.986733	0.551153

"Fast RBF" warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.321894	2.920949	404.172058	8841731	3.140191	-1.150514	1.199446	0.973596	0.518615
0.025	2.329457	0.972872	406.356415	8841521	2.538161	-1.174753	1.199092	0.973448	0.518588
0.035	2.327291	0.709789	412.380920	8843791	3.847987	-1.145706	1.197943	0.973046	0.518738
0.045	2.318076	0.577714	407.570831	8845694	4.236089	-1.132591	1.197927	0.973363	0.518788
0.055	2.450615	0.548590	439.383362	8836829	4.667472	-1.174438	1.194751	0.971247	0.518458
0.065	2.330112	0.498233	421.300018	8841151	3.194628	-1.144714	1.194824	0.972438	0.518580
0.075	2.348416	0.465809	360.990021	8871425	4.173345	-1.179090	1.201431	0.976451	0.519817
0.085	2.332604	0.484548	453.833862	8853054	3.739803	-1.127900	1.189925	0.970275	0.519407
0.095	2.327256	0.480071	441.968567	8860152	4.017978	-1.156522	1.191748	0.971059	0.519652
0.105	2.319905	0.472730	451.079407	8863654	1.852111	-1.122772	1.190274	0.970475	0.519948

B.1.4 Non-linear gradient synthetic test data

Brute force warp evaluation - software implementation

The gold standard software results :

Interp.	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
TRI	2.385287	332.187354	18.984509	15191091	2.979663	-1.613442	1.687073	0.998653	0.891942
NNI	2.388886	327.102142	23.333511	13659044	2.782177	-1.620051	1.590584	0.998306	0.756170

Brute force warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
2.385687	12.241180	127.440834	10288589	2.540931	-1.603371	1.526633	0.990650	0.539291

Grid based warp evaluation

The following tests used tri-linear interpolation :

Grid	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
138	2.368914	191.537994	63.703049	14312900	3.117686	-1.595245	1.608523	0.995400	0.811592
125	2.371000	155.340286	63.623184	14313144	3.011660	-1.605253	1.608784	0.995407	0.811705
113	2.364604	126.991302	63.804401	14317154	3.130313	-1.599780	1.609295	0.995395	0.811859
100	2.349782	99.267448	62.749512	14306119	3.322567	-1.649367	1.607947	0.995467	0.811466
88	2.408822	77.056656	63.003998	14304213	3.140551	-1.675840	1.608361	0.995451	0.811188
75	2.358987	55.975052	63.518181	14299948	2.818217	-1.656254	1.607952	0.995415	0.811023
63	2.363864	39.612785	63.527767	14310943	3.451259	-1.715200	1.608522	0.995412	0.811863
50	2.349845	24.948700	63.441753	14301916	2.806097	-1.734497	1.608355	0.995418	0.811077
38	2.368389	14.464858	63.436760	14288486	3.176606	-1.661967	1.606624	0.995421	0.810192
25	2.364611	6.377633	63.351830	14292200	3.482395	-1.602896	1.607036	0.995426	0.810215
13	2.365240	1.904456	64.570175	14069509	2.553038	-1.713190	1.584625	0.995337	0.791500

"Fast RBF" warp evaluation - software implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.392447	516.378967	19.162853	15173302	3.027346	-1.613453	1.684616	0.998636	0.890716
0.025	2.392657	48.760342	19.636452	15128372	4.314490	-1.613122	1.677530	0.998606	0.886954
0.035	2.388840	23.685511	20.621742	14917577	3.671751	-1.623797	1.652140	0.998532	0.868632
0.045	2.354849	16.214369	21.944826	14654770	4.810976	-1.673382	1.627286	0.998436	0.845270
0.055	2.363456	13.125979	24.580576	14625595	4.405814	-1.687690	1.621342	0.998245	0.842932
0.065	2.350722	11.947652	28.517147	14245425	4.819124	-1.661632	1.592412	0.997956	0.813181
0.075	2.350084	11.472793	35.221428	12919447	4.279626	-1.683306	1.515273	0.997450	0.708491
0.085	2.359809	11.340197	39.797939	13007606	3.682789	-1.708209	1.507191	0.997121	0.717536
0.095	2.347240	11.266697	44.112141	12742401	4.284968	-1.612157	1.495966	0.996807	0.696640
0.105	2.358863	11.196708	44.347378	12780619	4.589286	-1.598930	1.497191	0.996795	0.692583

The following tests used nearest neighbour interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.385199	508.666870	23.350586	13658494	3.906859	-1.639305	1.590467	0.998303	0.756145
0.025	2.384860	46.344097	23.593439	13657393	4.899445	-1.686209	1.590046	0.998285	0.756042
0.035	2.378093	21.283670	24.284628	13640887	3.971060	-1.699861	1.587380	0.998234	0.755109
0.045	2.383850	13.801839	25.591534	13616203	4.330639	-1.683079	1.583866	0.998138	0.753705
0.055	2.380190	10.721459	28.349133	13611022	4.933847	-1.716749	1.582428	0.997940	0.753548
0.065	2.384235	9.533742	32.782673	13561907	3.893878	-1.817585	1.575967	0.997618	0.750664
0.075	2.385205	9.064716	40.440777	13141791	4.429677	-1.665197	1.540646	0.997044	0.726817
0.085	2.393368	8.934108	45.711411	13154023	4.853023	-1.632482	1.533439	0.996659	0.728212
0.095	2.388659	8.842703	51.109234	13051278	3.471632	-1.738036	1.522600	0.996265	0.724061
0.105	2.388397	8.784272	52.178806	13033287	3.174085	-1.746699	1.521354	0.996189	0.719899

"Fast RBF" warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.402439	2.509266	127.623642	10292962	2.899455	-1.523961	1.525515	0.990640	0.539365
0.025	2.397686	0.893234	128.798645	10279055	4.127316	-1.545227	1.523904	0.990552	0.539041
0.035	2.392185	0.628190	132.398926	10306420	3.445459	-1.495340	1.515387	0.990289	0.538976
0.045	2.393439	0.521233	130.457947	10379257	4.631709	-1.555185	1.510206	0.990432	0.541380
0.055	2.386930	0.469770	146.033814	10116553	4.242591	-1.609260	1.506129	0.989272	0.534536
0.065	2.381622	0.443884	139.756027	10469417	4.190050	-1.580472	1.498649	0.989765	0.544935
0.075	2.385633	0.435474	110.955048	11307137	4.647873	-1.623419	1.469911	0.991893	0.594506
0.085	2.386163	0.428063	157.081131	10913469	4.024036	-1.667765	1.449349	0.988497	0.574804
0.095	2.388269	0.426446	151.667221	10883718	4.596086	-1.573595	1.434190	0.988874	0.576601
0.105	2.381747	0.422145	153.253586	10695861	5.251082	-1.534081	1.430396	0.988752	0.569746

B.2 Results for $\phi(r) = r^2 \log r$

B.2.1 Regular linear gradient synthetic test data

Brute force warp evaluation - software implementation

The gold standard software results :

Interp.	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
TRI	2.375827	790.143372	19.591047	15404284	3.622680	-1.514817	1.716064	0.998809	0.906939
NNI	2.372297	788.178650	23.941166	13796570	3.580482	-1.605565	1.589758	0.998518	0.761724

Brute force warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
2.405188	12.908881	126.513718	8802185	3.439864	-1.382141	1.622828	0.991996	0.514177

Grid based warp evaluation

The following tests used tri-linear interpolation :

Grid	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
138	2.385447	455.203369	60.381607	11845219	3.002713	-1.496159	1.623742	0.996272	0.574753
125	2.386728	373.404022	60.314842	11846322	2.496008	-1.587173	1.623591	0.996276	0.574872
113	2.382942	305.144806	60.412495	11846704	3.534164	-1.502830	1.623660	0.996272	0.574851
100	2.374989	238.804169	60.316711	11847104	2.875656	-1.480477	1.623635	0.996274	0.574908
88	2.381163	184.890976	60.373615	11848534	3.055329	-1.501650	1.623667	0.996273	0.574990
75	2.379978	134.294662	60.378086	11849277	3.771868	-1.446946	1.623527	0.996276	0.575072
63	2.373719	94.612480	60.309967	11851767	2.502073	-1.534162	1.623527	0.996278	0.575235
50	2.366801	59.717670	60.396671	11856881	2.562256	-1.528095	1.623074	0.996272	0.575641
38	2.382688	34.473492	60.430714	11863019	3.003667	-1.506037	1.622868	0.996271	0.576090
25	2.391750	14.995396	60.527515	11889227	2.743673	-1.552758	1.620659	0.996264	0.578289
13	2.381457	4.174581	62.266518	11990810	3.709186	-1.588241	1.600490	0.996152	0.590605

"Fast RBF" warp evaluation - software implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.344045	602.066711	19.692284	15387917	3.939583	-1.517238	1.712259	0.998803	0.905068
0.025	2.354419	69.015884	19.838425	15356157	4.333213	-1.597841	1.706521	0.998804	0.901898
0.035	2.345492	32.029785	22.594452	14983160	3.775148	-1.546317	1.650010	0.998628	0.865550
0.045	2.343558	20.214792	21.896992	14920970	4.458100	-1.540754	1.647742	0.998670	0.861678
0.055	2.337077	14.929594	31.467136	13419141	4.045789	-1.517868	1.555544	0.998077	0.719870
0.065	2.362302	13.981194	80.656013	11635348	4.025596	-1.558883	1.437399	0.995001	0.605083
0.075	2.363714	11.963979	70.185638	11487717	2.694458	-1.481696	1.441217	0.995642	0.604442
0.085	2.476465	11.739137	85.631615	12694484	2.276697	-1.464985	1.481289	0.994717	0.683477
0.095	2.668839	11.630121	45.458427	12938970	4.011122	-1.471309	1.508982	0.997205	0.712905
0.105	2.384763	11.484929	49.323299	12842567	3.744005	-1.474713	1.495371	0.996965	0.702858

The following tests used nearest neighbour interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.372937	595.957153	23.908245	13792304	3.530860	-1.554770	1.589324	0.998520	0.761457
0.025	2.371555	65.640869	23.931543	13789928	4.383592	-1.566025	1.589138	0.998518	0.761282
0.035	2.425696	28.505960	26.233250	13728796	4.260804	-1.531270	1.582035	0.998376	0.757594
0.045	2.398237	16.735376	25.496815	13728405	4.009974	-1.612295	1.582309	0.998420	0.757446
0.055	2.378571	11.433476	35.179676	13425267	3.146307	-1.552704	1.558201	0.997817	0.738808
0.065	2.389989	9.509941	104.604263	11859532	1.798000	-1.437564	1.449587	0.993502	0.651253
0.075	2.379765	8.496383	82.378311	12417840	4.022595	-1.494746	1.474230	0.994882	0.684052
0.085	2.387005	8.264435	114.131332	12694636	3.521001	-1.488588	1.492943	0.992951	0.696219
0.095	2.376403	8.103719	52.041019	13065795	3.640511	-1.547868	1.523785	0.996778	0.720303
0.105	2.386445	8.015146	58.302174	13005439	4.199722	-1.555436	1.514065	0.996391	0.716259

"Fast RBF" warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.380417	3.423604	125.801170	8814619	3.252999	-1.423853	1.618688	0.992040	0.514278
0.025	2.383680	1.006130	123.173553	8825411	4.191911	-1.437921	1.617567	0.992207	0.514228
0.035	2.383836	0.709868	118.279404	8838262	3.534575	-1.387174	1.595130	0.992505	0.514561
0.045	2.404492	0.643678	121.024307	8874266	4.644534	-1.451949	1.595444	0.992336	0.514567
0.055	2.400049	0.528564	95.907524	9108379	3.338026	-1.491821	1.543882	0.993942	0.514499
0.065	2.373239	0.494759	157.433228	9525613	3.821955	-1.375023	1.426205	0.990083	0.524735
0.075	2.451583	0.485487	239.962158	9250055	2.973526	-1.326314	1.414505	0.984807	0.525993
0.085	2.368906	0.547733	140.142471	9466030	2.885578	-1.460139	1.461692	0.991169	0.527771
0.095	2.362838	0.516206	138.284271	9429308	2.642907	-1.489552	1.481499	0.991289	0.527127
0.105	2.370828	0.481717	155.461655	9284591	3.139961	-1.433097	1.468878	0.990191	0.523800

B.2.2 Irregular linear gradient synthetic test data

Brute force warp evaluation - software implementation

The gold standard software results :

Interp.	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
TRI	2.355531	789.893372	45.692234	13777435	2.626005	-1.346634	1.550476	0.997703	0.803691
NNI	2.358341	786.234314	53.402924	13117000	2.089005	-1.443807	1.495663	0.997268	0.723957

Brute force warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
2.384521	12.903515	310.845947	8944236	2.405372	-1.338776	1.492722	0.983869	0.517436

Grid based warp evaluation

The following tests used tri-linear interpolation :

Grid	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
138	2.402756	457.606125	142.395981	10928963	3.038791	-1.390588	1.488394	0.992683	0.545244
125	2.380720	375.784742	142.317261	10930298	2.610419	-1.405167	1.487991	0.992685	0.545359
113	2.380672	307.525478	142.424744	10929716	1.914928	-1.325991	1.488277	0.992678	0.545302
100	2.380142	241.184311	142.315277	10930314	2.714800	-1.313660	1.488147	0.992685	0.545352
88	2.381356	187.272332	142.371811	10931257	2.409353	-1.342560	1.488310	0.992681	0.545403
75	2.397676	136.692338	142.368469	10930533	2.071585	-1.363222	1.488199	0.992683	0.545391
63	2.380746	96.993226	142.305054	10931757	2.572620	-1.344039	1.488184	0.992685	0.545464
50	2.351237	62.068908	142.402878	10933192	2.586279	-1.330451	1.487765	0.992680	0.545635
38	2.368118	36.841610	142.455215	10935145	2.414255	-1.387982	1.488091	0.992677	0.545760
25	2.411717	17.407113	142.655441	10944219	1.728329	-1.371382	1.486881	0.992671	0.546658
13	2.376342	6.550922	146.636505	10996925	1.581688	-1.302394	1.476686	0.992471	0.553101

"Fast RBF" warp evaluation - software implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.372944	602.092896	45.952698	13759268	3.562844	-1.343403	1.548132	0.997689	0.802157
0.025	2.382415	69.004898	46.501122	13754542	4.950953	-1.388868	1.546156	0.997662	0.800947
0.035	2.371496	31.984194	53.544243	13369396	3.408283	-1.415595	1.498819	0.997305	0.762078
0.045	2.371935	20.215055	52.073071	13440078	3.788849	-1.326007	1.506174	0.997377	0.769939
0.055	2.365602	14.931692	74.670746	12367490	4.363686	-1.338922	1.442399	0.996215	0.667740
0.065	2.375176	12.844070	206.675400	10832885	3.274747	-1.312763	1.340852	0.989327	0.579054
0.075	2.375260	11.967552	159.174759	11016116	2.280112	-1.310388	1.358444	0.991816	0.587586
0.085	2.383873	11.697344	193.669571	11363091	2.664755	-1.296355	1.368963	0.990040	0.608147
0.095	2.396173	11.536537	109.794098	12217028	3.302600	-1.330764	1.417796	0.994397	0.676216
0.105	2.391925	11.464852	119.313675	12153497	2.836535	-1.319870	1.407698	0.993903	0.670277

The following tests used nearest neighbour interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.359793	597.491638	53.458408	13114209	3.895327	-1.427114	1.495406	0.997268	0.723852
0.025	2.360430	65.499092	53.610725	13112414	3.248831	-1.414110	1.495298	0.997260	0.723674
0.035	2.355102	28.491638	59.822994	13061346	4.186475	-1.413529	1.487615	0.996940	0.721180
0.045	2.356236	16.707981	58.182007	13073527	3.876413	-1.445953	1.489451	0.997025	0.721747
0.055	2.355965	11.592660	79.993469	12830623	2.272686	-1.414155	1.466887	0.995892	0.708103
0.065	2.381390	9.327442	255.120575	11364139	2.667909	-1.258650	1.363840	0.986821	0.630065
0.075	2.388515	8.489008	178.657303	12068893	3.793486	-1.341367	1.399129	0.990784	0.666830
0.085	2.376465	8.258410	237.624268	12024714	2.804293	-1.339037	1.397066	0.987760	0.664005
0.095	2.388066	8.096834	120.683792	12617763	3.407633	-1.406785	1.442868	0.993795	0.697341
0.105	2.390454	8.005185	133.953827	12583690	3.407861	-1.361663	1.435953	0.993109	0.694760

"Fast RBF" warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.394883	2.731065	308.801575	8952572	2.772176	-1.283044	1.490138	0.983978	0.517605
0.025	2.387085	0.951575	302.944275	8967318	3.276702	-1.331702	1.491939	0.984293	0.517849
0.035	2.385026	0.673497	302.031616	9008616	3.348840	-1.312894	1.468353	0.984324	0.518463
0.045	2.355674	0.569640	305.163147	9011390	2.760578	-1.356686	1.472340	0.984164	0.518287
0.055	2.355906	0.507616	261.079224	9225745	2.815356	-1.409003	1.436125	0.986478	0.520362
0.065	2.358438	0.489217	395.764648	9729024	3.807795	-1.280962	1.332126	0.979414	0.538945
0.075	2.360978	0.484206	519.147888	9243836	2.889727	-1.249901	1.335641	0.972920	0.528697
0.085	2.355271	0.459316	356.893982	9594755	2.177010	-1.339064	1.355478	0.981464	0.534249
0.095	2.354124	0.466322	327.832733	9370132	3.064533	-1.362177	1.390857	0.982985	0.527565
0.105	2.359066	0.458180	370.897705	9257209	3.170925	-1.288077	1.381543	0.980723	0.524702

B.2.3 Checkered cube with random noise synthetic test data

Brute force warp evaluation - software implementation

The gold standard software results :

Interp.	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
TRI	2.377059	791.614441	126.434341	9288238	2.528866	-1.326850	1.265181	0.991938	0.532105
NNI	2.378087	786.240051	107.826675	9728415	3.369844	-1.327237	1.278012	0.993087	0.550281

Brute force warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
2.367751	12.844133	404.496155	8838082	2.391382	-1.181130	1.199271	0.973575	0.518398

Grid based warp evaluation

The following tests used tri-linear interpolation :

Grid	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
138	2.385476	457.588845	253.534576	8972895	2.782491	-1.208384	1.217566	0.983585	0.524219
125	2.367999	375.772021	253.471634	8972484	2.839912	-1.239212	1.217586	0.983591	0.524189
113	2.393205	307.538011	253.549088	8972289	2.275991	-1.178750	1.217568	0.983585	0.524177
100	2.356431	241.160599	253.470413	8972449	2.037857	-1.244752	1.217579	0.983591	0.524188
88	2.372878	187.263854	253.510391	8973014	3.186147	-1.186131	1.217566	0.983588	0.524224
75	2.365125	136.659787	253.515289	8973226	2.666205	-1.172888	1.217554	0.983589	0.524237
63	2.383653	96.996133	253.461533	8972769	2.782495	-1.196638	1.217573	0.983588	0.524209
50	2.350951	62.068622	253.524673	8973575	1.778622	-1.165453	1.217547	0.983589	0.524260
38	2.379196	36.852687	253.580750	8973686	2.158936	-1.207407	1.217545	0.983584	0.524265
25	2.411153	17.406549	253.763977	8974741	2.604774	-1.214940	1.217436	0.983573	0.524338
13	2.367457	6.542038	257.349487	8977051	2.623657	-1.183350	1.216137	0.983337	0.524577

"Fast RBF" warp evaluation - software implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.367152	600.830017	126.779449	9270397	4.425428	-1.294679	1.264923	0.991914	0.531147
0.025	2.373552	69.159927	127.306480	9279830	5.190026	-1.335783	1.264482	0.991880	0.531653
0.035	2.369162	32.016685	136.052109	9186411	3.183922	-1.370325	1.258406	0.991312	0.528443
0.045	2.366116	20.233414	133.598053	9219924	4.916340	-1.336546	1.260348	0.991470	0.529360
0.055	2.375819	14.933147	160.831818	9037576	4.230587	-1.254398	1.246714	0.989673	0.523619
0.065	2.375437	12.839722	308.762238	8908914	4.050317	-1.203561	1.204539	0.979905	0.520051
0.075	2.373691	11.967468	254.547470	8971173	2.901217	-1.169971	1.218434	0.983491	0.523113
0.085	2.380296	11.708549	293.614288	8985366	2.218701	-1.179145	1.209372	0.980951	0.523708
0.095	2.386113	11.549995	199.253464	9137660	4.462887	-1.216400	1.231864	0.987126	0.528064
0.105	2.393954	11.648142	209.400269	9072345	3.054528	-1.232129	1.228652	0.986454	0.525674

The following tests used nearest neighbour interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.387748	595.990723	107.968285	9728805	4.950707	-1.369074	1.277874	0.993074	0.550312
0.025	2.383124	65.668137	108.268532	9728050	3.346894	-1.441579	1.277686	0.993055	0.550267
0.035	2.391459	28.501915	117.032486	9728187	3.673051	-1.365876	1.273033	0.992476	0.550465
0.045	2.361383	16.729244	114.618584	9727557	4.222414	-1.340421	1.274299	0.992636	0.550381
0.055	2.376700	11.417331	142.698914	9696539	4.150001	-1.405132	1.262387	0.990783	0.549089
0.065	2.371470	9.340123	365.093292	9422435	3.104573	-1.261215	1.204987	0.976190	0.540702
0.075	2.367363	8.507390	260.519226	9624095	3.442962	-1.281171	1.229735	0.983046	0.548281
0.085	2.380511	8.266939	333.322540	9560037	4.714874	-1.232900	1.216185	0.978310	0.546207
0.095	2.391832	8.108346	197.874512	9676351	4.989373	-1.323776	1.243858	0.987152	0.549615
0.105	2.362009	7.999118	211.764587	9661980	3.624596	-1.319114	1.240164	0.986234	0.549291

"Fast RBF" warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.391128	2.726993	402.790222	8839664	3.355862	-1.153191	1.199294	0.973688	0.518509
0.025	2.376399	0.957887	398.112732	8840230	4.164253	-1.181282	1.199444	0.974000	0.518501
0.035	2.395262	0.683706	396.761536	8847312	3.687699	-1.155597	1.198391	0.974088	0.518885
0.045	2.358572	0.566860	399.429382	8843707	3.913337	-1.131970	1.198170	0.973920	0.518699
0.055	2.386940	0.510334	360.008362	8841291	2.821343	-1.209982	1.199373	0.976533	0.518192
0.065	2.362258	0.485753	470.442993	8836981	4.184361	-1.151628	1.184447	0.969143	0.518475
0.075	2.381778	0.482194	565.435059	8853377	2.753368	-1.103441	1.181561	0.962806	0.519779
0.085	2.383088	0.482324	446.404907	8881201	3.736889	-1.132724	1.188455	0.970755	0.520979
0.095	2.384876	0.467512	415.788971	8867654	2.443402	-1.111952	1.194313	0.972798	0.519938
0.105	2.377853	0.461260	451.672974	8864043	4.159781	-1.144300	1.189919	0.970430	0.519942

B.2.4 Non-linear gradient synthetic test data

Brute force warp evaluation - software implementation

The gold standard software results :

Interp.	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
TRI	2.655427	790.550395	18.480581	15204015	3.637929	-1.607983	1.688309	0.998689	0.892309
NNI	2.662549	786.884338	22.613308	13681778	3.252796	-1.628734	1.593237	0.998357	0.757362

Brute force warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
2.668583	12.150575	127.198395	10278898	3.382539	-1.607205	1.526939	0.990671	0.539471

Grid based warp evaluation

The following tests used tri-linear interpolation :

Grid	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
138	2.391126	457.594495	63.263046	14380895	3.073471	-1.675809	1.614877	0.995434	0.816229
125	2.398175	375.802197	63.206333	14378475	3.850818	-1.663173	1.614749	0.995437	0.816015
113	2.387243	307.532049	63.288033	14379032	3.710397	-1.717055	1.614788	0.995433	0.816070
100	2.379908	241.184076	63.208324	14378643	3.778827	-1.738901	1.614764	0.995437	0.816021
88	2.376288	187.267264	63.254997	14378927	3.344915	-1.666718	1.614749	0.995435	0.816030
75	2.380960	136.675623	63.256752	14378686	2.042067	-1.601609	1.614692	0.995434	0.816029
63	2.393656	97.006136	63.200871	14377591	3.784342	-1.730612	1.614699	0.995436	0.815918
50	2.351083	62.068753	63.271751	14371084	3.665046	-1.730035	1.614087	0.995432	0.815423
38	2.402667	36.876159	63.296906	14369833	2.837882	-1.662981	1.613937	0.995433	0.815290
25	2.391129	17.386524	63.386692	14341094	2.794712	-1.615735	1.611184	0.995420	0.812869
13	2.366035	6.540615	64.980927	14072572	2.647657	-1.728304	1.586158	0.995307	0.790168

"Fast RBF" warp evaluation - software implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.674665	630.982268	18.628507	15283682	3.006503	-1.619225	1.683217	0.998680	0.888832
0.025	2.650567	77.914455	18.796854	15213791	4.410259	-1.624485	1.672714	0.998668	0.880734
0.035	2.645845	36.364490	21.923120	15099935	3.870007	-1.632417	1.651298	0.998443	0.868420
0.045	2.626389	22.874748	20.955948	14943515	4.723029	-1.675763	1.634978	0.998513	0.850366
0.055	2.612950	16.806778	30.616223	14028938	4.328679	-1.657945	1.567590	0.997819	0.750275
0.065	2.624126	14.400051	88.262660	13518032	4.000382	-1.650926	1.501356	0.993629	0.715923
0.075	2.628585	13.401670	68.003870	12864898	4.444007	-1.665076	1.472087	0.995104	0.661370
0.085	2.616597	13.158685	88.087507	13464396	3.627984	-1.728970	1.495436	0.993643	0.709426
0.095	2.614994	12.966087	43.822963	13265953	5.173137	-1.622350	1.498652	0.996857	0.700565
0.105	2.619249	12.825166	48.440777	13281555	4.977075	-1.615368	1.491866	0.996524	0.695686

The following tests used nearest neighbour interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.655410	620.840210	22.604454	13678321	3.838090	-1.637274	1.592884	0.998357	0.757203
0.025	2.657108	72.928955	22.649372	13673856	4.999471	-1.694171	1.592541	0.998353	0.756909
0.035	2.665363	31.928062	25.346563	13660332	4.251327	-1.699968	1.588909	0.998155	0.755972
0.045	2.669980	18.696148	24.377157	13645878	4.518627	-1.683085	1.586968	0.998227	0.755317
0.055	2.672976	12.723988	34.102936	13499502	4.658998	-1.705496	1.575404	0.997522	0.746197
0.065	2.664312	10.374887	112.581268	12996150	3.676647	-1.774933	1.518199	0.991826	0.714805
0.075	2.662237	9.435470	77.829178	12878555	4.310036	-1.663417	1.506778	0.994317	0.711591
0.085	2.686661	9.166762	114.838463	13056970	4.929541	-1.643502	1.516912	0.991656	0.720379
0.095	2.663806	8.987809	48.981800	13007993	3.445803	-1.739342	1.524079	0.996419	0.720857
0.105	2.661408	8.875854	55.305874	12983459	3.334520	-1.717871	1.518424	0.995959	0.717337

"Fast RBF" warp evaluation - hardware implementation

The following tests used tri-linear interpolation :

H	TPS Time	Warp Time	MS	MRSD	MI	MMI	NMI	CC	CM
0.015	2.608479	2.507260	128.459153	10804707	3.369455	-1.660599	1.524744	0.990707	0.539358
0.025	2.612484	0.892530	126.011066	10877276	3.623257	-1.635933	1.525948	0.990886	0.541858
0.035	2.618628	0.629972	121.610433	10879980	3.850842	-1.601913	1.516394	0.991204	0.543363
0.045	2.615780	0.523501	124.729210	10879285	4.138794	-1.550483	1.509231	0.990979	0.542199
0.055	2.606009	0.472499	102.827263	11401094	4.441890	-1.658598	1.510325	0.992576	0.559310
0.065	2.611820	0.446739	162.741138	11050393	3.636330	-1.621917	1.447692	0.988211	0.562589
0.075	2.609573	0.435745	234.645180	10630001	4.079852	-1.513772	1.409508	0.982981	0.550770
0.085	2.609526	0.429452	144.783579	11226074	3.518717	-1.541242	1.437367	0.989517	0.569523
0.095	2.615183	0.426413	140.001258	11698360	4.380117	-1.630889	1.441866	0.989865	0.594191
0.105	2.615454	0.423736	156.866795	11259770	3.584064	-1.503846	1.426140	0.988637	0.576302

Appendix C

Programmable shaders - source code

This appendix contains source code listings for the hardware brute force warp evaluation technique and the hardware "Fast RBF" warp evaluation technique.

The source code is in Shader Model 3.0 (SM3) format and is supplied to the experimental software in DirectX "effect files" (*.fx).

C.1 Brute force hardware evaluation

The following SM3 shader source code implements the brute force warp evaluation on the GPU.

C.1.1 Source code for $\phi(r) = r$

```
// textures
texture gLandmarks;                    // tps landmarks as a texture
texture gWeights;                      // tps weights as a texture
texture gSourceTexture;                // source 3d texture

// sampler states
sampler Landmarks = sampler_state      // landmark texture - nni
{
```

```
  Texture = <gLandmarks>;
  MinFilter = POINT;
  MagFilter = POINT;
};

sampler Weights = sampler_state        // tps weights texture - nni
{
  Texture = <gWeights>;
  MinFilter = POINT;
  MagFilter = POINT;
};

sampler SourceTexture = sampler_state  // 3d source texture - tri
{
  Texture = <gSourceTexture>;
  MinFilter = LINEAR;
  MagFilter = LINEAR;
};

// global parameters
float4x4 gWVP;                         // matrix for w/v/p
float2 gBottomLeft;                    // bottom left quad coord
float3 gDimensions;                    // volume dimensions
float3 gRecipDimensions;               // reciprocal volume dimensions
float2 gTpsTextureSize;                // size of tps landmark and weights textures
float3x3 gAffineXform;                 // tps affine transformation
float3 gAffineXlate;                   // tps affine translation
float3 gTextureAdjust;                 // 3d texture limits
float4 gConstants;                     // various constants

technique TransformTech {
  pass P0
  {
    // set state
    Lighting = FALSE;
    FillMode = Solid;

    // bind samplers
    Sampler[0] = <Landmarks>;
    Sampler[1] = <Weights>;
```

```
  Sampler[2] = <SourceTexture>;

  // set vertex shader constants
  VertexShaderConstant4[0] = <gWVP>;
  VertexShaderConstant1[4] = <gBottomLeft>;
  VertexShaderConstant1[5] = <gRecipDimensions>;

  // set pixel shader constants
  PixelShaderConstant3[0] = <gAffineXform>;
  PixelShaderConstant1[3] = <gTpsTextureSize>;
  PixelShaderConstant1[4] = <gAffineXlate>;
  PixelShaderConstant1[5] = <gTextureAdjust>;
  PixelShaderConstant1[6] = <gConstants>;
  PixelShaderConstant1[7] = <gDimensions>;

  // define vertex shader
  VertexShader = asm
  {
    vs_3_0
    dcl_position o0                 // output vertex position
    dcl_texcoord0 o1                // output texcoord (actual position)
    dcl_position v0                 // input vertex position
    def c6, 0.0f, 0.0f, 0.0f, 0.0f  // various constants

    // set z = 0 on each vertex to ensure ortho
    mov r0, v0                      // copy incoming vertex
    mov r0.z, c6.x                  // zeroise incoming z coordinate
    m4x4 o0, r0, c0                 // transform adjusted vertex by wvp matrix

    // preserve original coordinates
    add r0, v0, c4                  // offset to bottom left
    mul o1, r0, c5                  // normalise
};

  // define pixel shader
  PixelShader = asm
  {
    ps_3_0
    dcl_texcoord0 v0                // target voxel position - from vertex shader
    dcl_2d s0                       // landmarks - coded in a 2d texture
```

```
dcl_2d s1                       // weights - coded in a 2d texture
dcl_volume s2                   // source volume as a 3d texture
// various constants 1/255, 1/log2(e)
def c8, 0.003921568f, 0.693147180559945f, 0.0f, 0.0f
defi i0, 255, 0, 0, 0           // loop control 1 - null decrement
defi i1, 255, 0, 0, 0           // loop control 2 - null decrement

// recover original voxel coordinates
mul r5.xyz, v0.xyz, c7.xyz      // multiply 0..1 by volume dimensions
add r5.y, c7.y, -r5.y           // invert y for dx9 texture

// initialise variables
mov r0.x, c6.x                  // float fIndex = 0.0f;
mov r1.xyz, c6.xxx              // float3 fWeighted = {0.0f, 0.0f, 0.0f};
mov r2.xyzw, c6.xxxx            // float4 TpsCoord = {0.0f, 0.0f, 0.0f, 0.0f};

// loop for all landmarks
loop aL, i0                     // while (true)
  break_ge r0.x, c6.w           // if (fIndex >= nLandmarks) break;
  loop aL, i1                   // while (true) - nested loop needed for > 255
    break_ge r0.x, c6.w         // if (fIndex >= nLandmarks) break;

    // TpsCoord.x = fIndex * (1.0f / gTpsTextureSize.x);
    mul r2.x, r0.x, c3.x

    // get landmark position
    texldl r3, r2.xy, s0        // fLandmark = tex2Dlod(Landmarks, TpsCoord);

    // calculate r for this landmark
    add r4.xyz, r5.xyz, -r3.xyz  // r4 = posL - fLandmark;
    dp3 r4.w, r4, r4            // RSquared = (r4.x)^2 + (r4.y)^2 + (r4.z)^2;
    rsq r3.w, r4.w              // R = 1.0f / sqrt(RSquared);
    rcp r3.w, r3.w              // R = 1.0f / R;

    // calculate rbf from r (r3.w) and/or r^2 (r4.w) and store in r4.x
    //
    mov r4.x, r3.w              // fRbf = R;

    // fetch coefficients for this landmark
    texldl r3, r2.xy, s1        // fWeight = tex2Dlod(Weights, TpsCoord);
```

```
          // accumulate weight
          mul r3.xyz, r3.xyz, r4.x
          add r1.xyz, r1.xyz, r3.xyz

          // increment index
          add r0.x, r0.x, c6.z          // fIndex += 1.0f;
        endloop                         // endwhile
      endloop                           // endwhile

      // apply affine transformation
      dp3 r3.x, r5, c0                  // fWarped = mul(gAffineXform, posL);
      dp3 r3.y, r5, c1                  //    "       "       "      "
      dp3 r3.z, r5, c2                  //    "       "       "      "
      add r2.xyz, r3.xyz, c4            // fWarped += gAffineXlate;
      add r2.xyz, r2.xyz, r1.xyz        // fWarped += fWeighted;

      // convert to texture coordinates and look up
      mul_sat r3.xyz, r2.xyz, c5.xyz    // fWarped *= gTextureAdjust;
      add r3.y, c6.z, -r3.y             // tex.y = 1.0 - tex.y for DX9
      texld r0, r3.xyz, s2              // fResult = tex3D(SourceTexture, fWarped);
      mov oC0, r0
  };
 }
}
```

C.1.2 Source code for $\phi(r) = r^2 \log r$

```
// textures
texture gLandmarks;                     // tps landmarks as a texture
texture gWeights;                       // tps weights as a texture
texture gSourceTexture;                 // source 3d texture

// sampler states
sampler Landmarks = sampler_state       // landmark texture - nearest neighbour
{
  Texture = <gLandmarks>;
  MinFilter = POINT;
  MagFilter = POINT;
};
```

```
sampler Weights = sampler_state        // tps weights texture - nearest neighbour
{
  Texture = <gWeights>;
  MinFilter = POINT;
  MagFilter = POINT;
};

sampler SourceTexture = sampler_state  // 3d source texture - tri-linear interpolation
{
  Texture = <gSourceTexture>;
  MinFilter = LINEAR;
  MagFilter = LINEAR;
};

// global parameters
float4x4 gWVP;                         // external matrix for world/view/projection
float2 gBottomLeft;                    // bottom left quad coordinate
float3 gDimensions;                    // volume dimensions
float3 gRecipDimensions;               // reciprocal volume dimensions
float2 gTpsTextureSize;                // size of tps landmark and weights textures
float3x3 gAffineXform;                 // tps affine transformation
float3 gAffineXlate;                   // tps affine translation
float3 gTextureAdjust;                 // 3d texture limits
float4 gConstants;                     // various constants

technique TransformTech
{
  pass P0
  {
    // set state
    Lighting = FALSE;
    FillMode = Solid;

    // bind samplers
    Sampler[0] = <Landmarks>;
    Sampler[1] = <Weights>;
    Sampler[2] = <SourceTexture>;

    // set vertex shader constants
```

```
VertexShaderConstant4[0] = <gWVP>;
VertexShaderConstant1[4] = <gBottomLeft>;
VertexShaderConstant1[5] = <gRecipDimensions>;

// set pixel shader constants
PixelShaderConstant3[0] = <gAffineXform>;
PixelShaderConstant1[3] = <gTpsTextureSize>;
PixelShaderConstant1[4] = <gAffineXlate>;
PixelShaderConstant1[5] = <gTextureAdjust>;
PixelShaderConstant1[6] = <gConstants>;
PixelShaderConstant1[7] = <gDimensions>;

// define vertex shader
VertexShader = asm
{
  vs_3_0
  dcl_position o0                 // output vertex position
  dcl_texcoord0 o1                // output texcoord (actual position)
  dcl_position v0                 // input vertex position
  def c6, 0.0f, 0.0f, 0.0f, 0.0f  // various constants

  // set z = 0 on each vertex to ensure ortho
  mov r0, v0                      // copy incoming vertex
  mov r0.z, c6.x                  // zeroise incoming z coordinate
  m4x4 o0, r0, c0                 // transform adjusted vertex by wvp matrix

  // preserve original coordinates
  add r0, v0, c4
  mul o1, r0, c5                  // normalise
};

// define pixel shader
PixelShader = asm
{
  ps_3_0
  dcl_texcoord0 v0                // target voxel position - from vertex shader
  dcl_2d s0                       // landmarks - coded in a 2d texture
  dcl_2d s1                       // weights - coded in a 2d texture
  dcl_volume s2                   // source volume as a 3d texture
  def c8, 0.693147180559945f, 0.0f, 0.0f, 0.0f    // various constants 1/log2(e)
```

```
defi i0, 255, 0, 0, 0            // loop control 1 - null decrement
defi i1, 255, 0, 0, 0            // loop control 2 - null decrement

// recover original voxel coordinates
mul r5.xyz, v0.xyz, c7.xyz       // multiply 0..1 by volume dimensions
add r5.y, c7.y, -r5.y            // invert y for dx9 texture

// initialise variables
mov r0.x, c6.x                   // float fIndex = 0.0f;
mov r1.xyz, c6.xxx               // float3 fWeighted = {0.0f, 0.0f, 0.0f};
mov r2.xyzw, c6.xxxx             // float4 TpsCoord = {0.0f, 0.0f, 0.0f, 0.0f};

// loop for all landmarks
loop aL, i0                      // while (true)
  break_ge r0.x, c6.w            // if (fIndex >= nLandmarks) break;
  loop aL, i1                    // while (true) - nested loop needed for > 255
    break_ge r0.x, c6.w          // if (fIndex >= nLandmarks) break;

    // TpsCoord.x = fIndex * (1.0f / gTpsTextureSize.x);
    mul r2.x, r0.x, c3.x

    // get landmark position
    texldl r3, r2.xy, s0         // fLandmark = tex2Dlod(Landmarks, TpsCoord);

    // calculate r for this landmark
    add r4.xyz, r5.xyz, -r3.xyz  // r4 = posL - fLandmark;
    dp3 r4.w, r4, r4             // RSquared = (r4.x)^2 + (r4.y)^2 + (r4.z)^2;
    rsq r3.w, r4.w               // R = 1.0f / sqrt(RSquared);
    rcp r3.w, r3.w               // R = 1.0f / R;

    // calculate rbf from r (r3.w) and/or r^2 (r4.w) and store in r4.x
    //
    log r6.y, r3.w               // r6 = log2(R);
    mul r6.y, r6.y, c8.x         // r6 *= 1 / log2(e);  so r6.y = loge(R)
    mul r4.x, r4.w, r6.y         // fRbf = R^2logR;

    // fetch coefficients for this landmark
    texldl r3, r2.xy, s1         // fWeight = tex2Dlod(Weights, TpsCoord);

    // accumulate weight
```

```
            mul r3.xyz, r3.xyz, r4.x
            add r1.xyz, r1.xyz, r3.xyz

            // increment index
            add r0.x, r0.x, c6.z          // fIndex += 1.0f;
          endloop                         // endwhile
        endloop                           // endwhile

        // apply affine transformation
        dp3 r3.x, r5, c0                  // fWarped = mul(gAffineXform, posL);
        dp3 r3.y, r5, c1                  //    "       "       "      "
        dp3 r3.z, r5, c2                  //    "       "       "      "
        add r2.xyz, r3.xyz, c4            // fWarped += gAffineXlate;
        add r2.xyz, r2.xyz, r1.xyz        // fWarped += fWeighted;

        // convert to texture coordinates and look up
        mul_sat r3.xyz, r2.xyz, c5.xyz    // fWarped *= gTextureAdjust;
        add r3.y, c6.z, -r3.y             // tex.y = 1.0 - tex.y for DX9
        texld r0, r3.xyz, s2              // fResult = tex3D(SourceTexture, fWarped);
        mov oC0, r0

    };
  }
}
```

C.2 "Fast RBF" hardware evaluation

The following SM3 shader source code implements the "Fast RBF" warp evaluation on the GPU.

C.2.1 Source code for $\phi(r) = r$

```
// textures
texture gLandmarks;                   // landmarks as a texture
texture gWeights;                     // weights as a texture
texture gXGrid;                       // X grid as a texture
texture gSourceTexture;               // source 3d texture

// sampler states
```

```
sampler Landmarks = sampler_state      // landmark texture - nearest neighbour
{
  Texture = <gLandmarks>;
  MinFilter = POINT;
  MagFilter = POINT;
};

sampler Weights = sampler_state        // weights texture - nearest neighbour
{
  Texture = <gWeights>;
  MinFilter = POINT;
  MagFilter = POINT;
};

sampler XGrid = sampler_state          // X grid texture - nearest neighbour
{
  Texture = <gXGrid>;
  MinFilter = POINT;
  MagFilter = POINT;
};

sampler SourceTexture = sampler_state  // 3d source texture - tri-linear interpolation
{
  Texture = <gSourceTexture>;
  MinFilter = LINEAR;
  MagFilter = LINEAR;
};

// global parameters
float2 gTpsTextureSize;                // size of landmark and weights textures
float2 gRecipTpsTextureSize;           // reciprocal size of landmark and weights textures
float4 gBottomLeft;                    // bottom left quad coordinate
float3 gOriginX;                       // origin of X grid
float3 gDimensions;                    // dimensions
float4 gConstants;                     // various constants
float4 gFlat3D;                        // flat 3d constants
float3 gRecipDimensions;               // reciprocal dimensions
float3x3 gAffineXform;                 // tps affine transformation
float3 gAffineXlate;                   // tps affine translation
float3 gTextureAdjust;                 // 3d texture limits
```

```
// the following technique performs the coarse
// summation to create grid X from grid Y, which
// is supplied as a set of landmarks and weights
// in two 2d textures. The X grid nodes are
// supplied as vertices and actual positions are
// derived in the pixel shader using the X grid
// start position and the grid size, H.
technique SummationTech
{
  pass P0
  {
    // set state
    Lighting = FALSE;
    FillMode = Solid;

    // bind samplers
    Sampler[0] = <Landmarks>;
    Sampler[1] = <Weights>;

    // set pixel shader constants
    PixelShaderConstant1[0] = <gTpsTextureSize>;
    PixelShaderConstant1[1] = <gOriginX>;
    PixelShaderConstant1[2] = <gConstants>;           // [0.0f, H, 1.0f, nLandmarks]
    PixelShaderConstant1[3] = <gRecipTpsTextureSize>; // reciprocal for easy mul
    PixelShaderConstant1[4] = <gDimensions>;
    PixelShaderConstant1[5] = <gBottomLeft>;

    // define pixel shader
    PixelShader = asm
    {
      ps_3_0
      dcl vPos.xy                     // target screen position
      dcl_2d s0                       // landmarks - coded in a 2d texture
      dcl_2d s1                       // weights - coded in a 2d texture
      defi i0, 255, 0, 0, 0           // loop control 1 - null decrement
      defi i1, 255, 0, 0, 0           // loop control 2 - null decrement

      // recover original X node coordinates
      add r5.xy, vPos.xy, c5.xy       // recover offset for this slice
```

```
mov r5.z, c5.z

// calculate actual position
mov r0.x, c2.y
mad r5.xyz, r5.xyz, r0.x, c1     // multiply by H and add origin

// initialise variables
mov r0.x, c2.x                   // float fIndex = 0.0f;
mov r1.xyz, c2.xxx               // float3 fWeighted = {0.0f, 0.0f, 0.0f};
mov r2.xyzw, c2.xxxx             // float4 TpsCoord = {0.0f, 0.0f, 0.0f, 0.0f};

// loop for all landmarks
loop aL, i0                      // while (true)
  break_ge r0.x, c2.w            // if (fIndex >= nLandmarks) break;
  loop aL, i1                    // while (true) - nested loop needed for > 255
    break_ge r0.x, c2.w          // if (fIndex >= nLandmarks) break;

    // TpsCoord.y = floor(fIndex / gTpsTextureSize.x);
    mul r2.y, r0.x, c3.x         // TpsCoord.y = fIndex * gRecipTpsTextureSize.x;
    frc r3.z, r2.y               // extract fractional part of TpsCoord.y
    add r2.y, r2.y, -r3.z        // TpsCoord.y = floor(TpsCoord.y)

    // TpsCoord.x = fIndex - (gTpsTextureSize.x * TpsCoord.y);
    mad r2.x, -r2.y, c0.x, r0.x  // TpsCoord.x = remainder

    // TpsCoord.xy /= gTpsTextureSize;
    mul r2.xy, r2.xy, c3.xy      // normalise TpsCoord.xy

    // get landmark position
    texldl r3, r2.xy, s0         // fLandmark = tex2Dlod(Landmarks, TpsCoord);

    // calculate r for this landmark
    add r4.xyz, r5.xyz, -r3.xyz  // r4 = posL - fLandmark;
    dp3 r4.w, r4, r4             // RSquared = (r4.x)^2 + (r4.y)^2 + (r4.z)^2;
    rsq r3.w, r4.w               // R = 1.0f / sqrt(RSquared);
    rcp r3.w, r3.w               // R = 1.0f / R;

    // calculate rbf from r (r3.w) and/or r^2 (r4.w) and store in r4.x
    //
    mov r4.x, r3.w               // fRbf = R;
```

```
          // fetch coefficients for this landmark
          texldl r3, r2.xy, s1         // fWeight = tex2Dlod(Weights, TpsCoord);

          // accumulate weight
          mad r1.xyz, r3.xyz, r4.x, r1.xyz // fWeighted += fWeight * fRbf;

          // increment index
          add r0.x, r0.x, c2.z         // fIndex += 1.0f;
        endloop                        // endwhile
      endloop                          // endwhile

      // output weights
      mov oC0, r1
    };
  }
}

// the following technique applies the tensor
// product interpolation for each voxel and
// returns the value of the warped voxel position
// from the 3d source texture.
technique TransformTech
{
  pass P0
  {
    // set state
    Lighting = FALSE;
    FillMode = Solid;

    // bind samplers
    Sampler[0] = <XGrid>;
    Sampler[1] = <SourceTexture>;

    // set pixel shader constants
    PixelShaderConstant3[0]  = <gAffineXform>;
    PixelShaderConstant1[3]  = <gAffineXlate>;
    PixelShaderConstant1[4]  = <gTextureAdjust>;
    PixelShaderConstant1[5]  = <gConstants>;        // [H, 1/H, Columns, 1/Columns]
    PixelShaderConstant1[6]  = <gBottomLeft>;
```

```
PixelShaderConstant1[7]  = <gDimensions>;
PixelShaderConstant1[8]  = <gRecipDimensions>;
PixelShaderConstant1[9]  = <gOriginX>;
PixelShaderConstant1[10] = <gFlat3D>; // [CellWidth, CellHeight, RecipTexW, RecipTexH]

// define pixel shader
PixelShader = asm
{
  ps_3_0
  dcl vPos.xy                     // target screen position
  dcl_2d s0                       // X grid - coded in a 2d texture
  dcl_volume s1                   // source volume as a 3d texture
  def c11, 1.0f, 1.0f, 1.0f, 0.0f  // various constants
  def c12, 0.0f, 1.0f, 0.0f, 1.0f  // various constants
  def c13, 0.000001f, 0.0f, 0.0f, 0.0f

  // get original voxel coordinates
  mov r0, vPos.xy                 // recover z for this slice
  mov r0.z, c6.z

  // normalise voxel position (keep in r5)
  mul r5, r0, c8                  // xyz /= volume dimensions

  // adjust to X grid origin
  add r0, r5, -c9                 // xyz -= xorigin

  // find the integral x node coordinates (r1) and
  // the city-block distances to these (r2) as
  // a fraction of H (easier for interpolation weights)
  mul r1.xyz, r0.xyz, c5.yyy      // r1 = xyz / H;
  frc r3, r1                      // r3 = frac(r1);   note == dW0
  add r1, r1, -r3                 // r1 = floor(r1);

  // calculate interpolation weights dW0 (r2) and dW1 (r3)
  add r2, c11, -r3                // dW1 = 1 - dW0

  // initialise tensor product
  mov r10.xyz, c11.w              // fWeighted = (0, 0, 0);

  // interpolate from X grid - node 1 of 8
```

```
mul r4.y, r2.x, r2.y
mul r4.x, r4.y, r2.z             // dWeight = dWX0 * dWY0 * dWZ0;
mov r6.x, c13.x                  // add epsilon to avoid frc = 1
mad r6.y, r1.z, c5.w, r6.x       // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x            // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r1.z      // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r1.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw         // get flat xy texture coordinates
texld r8, r7, s0                 // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz // fWeighted += (dWeight * X grid weights);

// node 2 of 8
add r9.xyz, r1.xyz, c12.xxy      // node = x, y, z+1
mul r4.x, r4.y, r3.z             // dWeight = dWX0 * dWY0 * dWZ1;
mov r6.x, c13.x                  // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x       // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x            // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z      // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw         // get flat xy texture coordinates
texld r8, r7, s0                 // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz // fWeighted += (dWeight * X grid weights);

// node 3 of 8
add r9.xyz, r1.xyz, c12.xyz      // node = x, y+1, z
mul r4.y, r2.x, r3.y
mul r4.x, r4.y, r2.z             // dWeight = dWX0 * dWY1 * dWZ0;
mov r6.x, c13.x                  // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x       // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x            // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z      // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw         // get flat xy texture coordinates
texld r8, r7, s0                 // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz // fWeighted += (dWeight * X grid weights);

// node 4 of 8
```

```
add r9.xyz, r1.xyz, c12.xyy      // node = x, y+1, z+1
mul r4.x, r4.y, r3.z             // dWeight = dWX0 * dWY1 * dWZ1;
mov r6.x, c13.x                  // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x       // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x            // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z      // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw         // get flat xy texture coordinates
texld r8, r7, s0                 // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz  // fWeighted += (dWeight * X grid weights);

// node 5 of 8
add r9.xyz, r1.xyz, c12.yzz      // node = x+1, y, z
mul r4.y, r3.x, r2.y
mul r4.x, r4.y, r2.z             // dWeight = dWX1 * dWY0 * dWZ0;
mov r6.x, c13.x                  // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x       // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x            // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z      // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw         // get flat xy texture coordinates
texld r8, r7, s0                 // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz  // fWeighted += (dWeight * X grid weights);

// node 6 of 8
add r9.xyz, r1.xyz, c12.yzw      // node = x+1, y, z+1
mul r4.x, r4.y, r3.z             // dWeight = dWX1 * dWY0 * dWZ0;
mov r6.x, c13.x                  // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x       // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x            // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z      // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw         // get flat xy texture coordinates
texld r8, r7, s0                 // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz  // fWeighted += (dWeight * X grid weights);

// node 7 of 8
```

```
add r9.xyz, r1.xyz, c12.yyz      // node = x+1, y+1, z
mul r4.y, r3.x, r3.y
mul r4.x, r4.y, r2.z             // dWeight = dWX1 * dWY1 * dWZ0;
mov r6.x, c13.x                  // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x       // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x            // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z      // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw         // get flat xy texture coordinates
texld r8, r7, s0                 // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz  // fWeighted += (dWeight * X grid weights);

// node 8 of 8
add r9.xyz, r1.xyz, c12.yyy      // node = x+1, y+1, z+1
mul r4.x, r4.y, r3.z             // dWeight = dWX1 * dWY1 * dWZ1;
mov r6.x, c13.x                  // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x       // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x            // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z      // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw         // get flat xy texture coordinates
texld r8, r7, s0                 // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz  // fWeighted += (dWeight * X grid weights);

// r10.xyz finally holds the tensor product
// so apply the affine transformation
dp3 r3.x, r5, c0                 // fWarped = mul(gAffineXform, posL);
dp3 r3.y, r5, c1                 //    "       "       "      "
dp3 r3.z, r5, c2                 //    "       "       "      "
add r2.xyz, r3.xyz, c3           // fWarped += gAffineXlate;
add r2, r2, r10                  // fWarped += fWeighted;

// still in normalised space so convert back to voxels
mul r2, r2, c7                   // fWarped *= volume dimensions

// convert to texture coordinates and look up
mul_sat r3.xyz, r2.xyz, c4.xyz   // fWarped *= gTextureAdjust;
add r3.y, c11.y, -r3.y           // tex.y = 1.0 - tex.y for DX9
```

```
      texld r0, r3.xyz, s1            // fResult = tex3D(SourceTexture, fWarped);
      mov oC0, r0
    };
  }
}
```

C.2.2 Source code for $\phi(r) = r^2 \log r$

```
// textures
texture gLandmarks;                   // landmarks as a texture
texture gWeights;                     // weights as a texture
texture gXGrid;                       // X grid as a texture
texture gSourceTexture;               // source 3d texture

// sampler states
sampler Landmarks = sampler_state     // landmark texture - nearest neighbour
{
  Texture = <gLandmarks>;
  MinFilter = POINT;
  MagFilter = POINT;
};

sampler Weights = sampler_state       // weights texture - nearest neighbour
{
  Texture = <gWeights>;
  MinFilter = POINT;
  MagFilter = POINT;
};

sampler XGrid = sampler_state         // X grid texture - nearest neighbour
{
  Texture = <gXGrid>;
  MinFilter = POINT;
  MagFilter = POINT;
};

sampler SourceTexture = sampler_state // 3d source texture - tri-linear interpolation
{
  Texture = <gSourceTexture>;
  MinFilter = LINEAR;
```

```
  MagFilter = LINEAR;
};

// global parameters
float2 gTpsTextureSize;               // size of landmark and weights textures
float2 gRecipTpsTextureSize;          // reciprocal size of landmark and weights textures
float4 gBottomLeft;                   // bottom left quad coordinate
float3 gOriginX;                      // origin of X grid
float3 gDimensions;                   // dimensions
float4 gConstants;                    // various constants
float4 gFlat3D;                       // flat 3d constants
float3 gRecipDimensions;              // reciprocal dimensions
float3x3 gAffineXform;                // tps affine transformation
float3 gAffineXlate;                  // tps affine translation
float3 gTextureAdjust;                // 3d texture limits

// the following technique performs the coarse
// summation to create grid X from grid Y, which
// is supplied as a set of landmarks and weights
// in two 2d textures. The X grid nodes are
// supplied as vertices and actual positions are
// derived in the pixel shader using the X grid
// start position and the grid size, H.
technique SummationTech
{
  pass P0
  {
    // set state
    Lighting = FALSE;
    FillMode = Solid;

    // bind samplers
    Sampler[0] = <Landmarks>;
    Sampler[1] = <Weights>;

    // set pixel shader constants
    PixelShaderConstant1[0] = <gTpsTextureSize>;
    PixelShaderConstant1[1] = <gOriginX>;
    PixelShaderConstant1[2] = <gConstants>;          // [0.0f, H, 1.0f, nLandmarks]
    PixelShaderConstant1[3] = <gRecipTpsTextureSize>; // reciprocal for easy mul
```

```
PixelShaderConstant1[4] = <gDimensions>;
PixelShaderConstant1[5] = <gBottomLeft>;

// define pixel shader
PixelShader = asm
{
  ps_3_0
  dcl vPos.xy                     // target screen position
  dcl_2d s0                       // landmarks - coded in a 2d texture
  dcl_2d s1                       // weights - coded in a 2d texture
  defi i0, 255, 0, 0, 0           // loop control 1 - null decrement
  defi i1, 255, 0, 0, 0           // loop control 2 - null decrement

  // recover original X node coordinates
  add r5.xy, vPos.xy, c5.xy       // recover offset for this slice
  mov r5.z, c5.z

  // calculate actual position
  mov r0.x, c2.y
  mad r5.xyz, r5.xyz, r0.x, c1    // multiply by H and add origin

  // initialise variables
  mov r0.x, c2.x                  // float fIndex = 0.0f;
  mov r1.xyz, c2.xxx              // float3 fWeighted = {0.0f, 0.0f, 0.0f};
  mov r2.xyzw, c2.xxxx            // float4 TpsCoord = {0.0f, 0.0f, 0.0f, 0.0f};

  // loop for all landmarks
  loop aL, i0                     // while (true)
    break_ge r0.x, c2.w           // if (fIndex >= nLandmarks) break;
    loop aL, i1                   // while (true) - nested loop needed for > 255
      break_ge r0.x, c2.w         // if (fIndex >= nLandmarks) break;

      // TpsCoord.y = floor(fIndex / gTpsTextureSize.x);
      mul r2.y, r0.x, c3.x        // TpsCoord.y = fIndex * gRecipTpsTextureSize.x;
      frc r3.z, r2.y              // extract fractional part of TpsCoord.y
      add r2.y, r2.y, -r3.z       // TpsCoord.y = floor(TpsCoord.y)

      // TpsCoord.x = fIndex - (gTpsTextureSize.x * TpsCoord.y);
      mad r2.x, -r2.y, c0.x, r0.x  // TpsCoord.x = remainder
```

```
            // TpsCoord.xy /= gTpsTextureSize;
            mul r2.xy, r2.xy, c3.xy      // normalise TpsCoord.xy

            // get landmark position
            texldl r3, r2.xy, s0         // fLandmark = tex2Dlod(Landmarks, TpsCoord);

            // calculate r for this landmark
            add r4.xyz, r5.xyz, -r3.xyz  // r4 = posL - fLandmark;
            dp3 r4.w, r4, r4             // RSquared = (r4.x)^2 + (r4.y)^2 + (r4.z)^2;
            rsq r3.w, r4.w               // R = 1.0f / sqrt(RSquared);
            rcp r3.w, r3.w               // R = 1.0f / R;

            // calculate rbf from r (r3.w) and/or r^2 (r4.w) and store in r4.x
            //
            log r6.y, r3.w               // r6 = log2(R);
            mul r6.y, r6.y, c6.x         // r6 *= 1 / log2(e);  so r6.y = loge(R)
            mul r4.x, r4.w, r6.y         // fRbf = R^2logR;

            // fetch coefficients for this landmark
            texldl r3, r2.xy, s1         // fWeight = tex2Dlod(Weights, TpsCoord);

            // accumulate weight
            mad r1.xyz, r3.xyz, r4.x, r1.xyz // fWeighted += fWeight * fRbf;

            // increment index
            add r0.x, r0.x, c2.z         // fIndex += 1.0f;
          endloop                        // endwhile
        endloop                          // endwhile

        // output weights
        mov oC0, r1
      };
    }
}

// the following technique applies the tensor
// product interpolation for each voxel and
// returns the value of the warped voxel position
// from the 3d source texture.
technique TransformTech
```

```
{
  pass P0
  {
    // set state
    Lighting = FALSE;
    FillMode = Solid;

    // bind samplers
    Sampler[0] = <XGrid>;
    Sampler[1] = <SourceTexture>;

    // set pixel shader constants
    PixelShaderConstant3[0]  = <gAffineXform>;
    PixelShaderConstant1[3]  = <gAffineXlate>;
    PixelShaderConstant1[4]  = <gTextureAdjust>;
    PixelShaderConstant1[5]  = <gConstants>;        // [H, 1/H, Columns, 1/Columns]
    PixelShaderConstant1[6]  = <gBottomLeft>;
    PixelShaderConstant1[7]  = <gDimensions>;
    PixelShaderConstant1[8]  = <gRecipDimensions>;
    PixelShaderConstant1[9]  = <gOriginX>;
    PixelShaderConstant1[10] = <gFlat3D>; // [CellWidth, CellHeight, RecipTexW, RecipTexH]

    // define pixel shader
    PixelShader = asm
    {
      ps_3_0
      dcl vPos.xy                     // target screen position
      dcl_2d s0                       // X grid - coded in a 2d texture
      dcl_volume s1                   // source volume as a 3d texture
      def c11, 1.0f, 1.0f, 1.0f, 0.0f  // various constants
      def c12, 0.0f, 1.0f, 0.0f, 1.0f  // various constants
      def c13, 0.000001f, 0.0f, 0.0f, 0.0f

      // get original voxel coordinates
      mov r0, vPos.xy                 // recover z for this slice
      mov r0.z, c6.z

      // normalise voxel position (keep in r5)
      mul r5, r0, c8                  // xyz /= volume dimensions
```

```
// adjust to X grid origin
add r0, r5, -c9                 // xyz -= xorigin

// find the integral x node coordinates (r1) and
// the city-block distances to these (r2) as
// a fraction of H (easier for interpolation weights)
mul r1.xyz, r0.xyz, c5.yyy      // r1 = xyz / H;
frc r3, r1                      // r3 = frac(r1);   note == dW0
add r1, r1, -r3                 // r1 = floor(r1);

// calculate interpolation weights dW0 (r2) and dW1 (r3)
add r2, c11, -r3                // dW1 = 1 - dW0

// initialise tensor product
mov r10.xyz, c11.w              // fWeighted = (0, 0, 0);

// interpolate from X grid - node 1 of 8
mul r4.y, r2.x, r2.y
mul r4.x, r4.y, r2.z            // dWeight = dWX0 * dWY0 * dWZ0;
mov r6.x, c13.x                 // add epsilon to avoid frc = 1
mad r6.y, r1.z, c5.w, r6.x      // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x           // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r1.z     // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r1.xy // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw        // get flat xy texture coordinates
texld r8, r7, s0                // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz // fWeighted += (dWeight * X grid weights);

// node 2 of 8
add r9.xyz, r1.xyz, c12.xxy     // node = x, y, z+1
mul r4.x, r4.y, r3.z            // dWeight = dWX0 * dWY0 * dWZ1;
mov r6.x, c13.x                 // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x      // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x           // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z     // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw        // get flat xy texture coordinates
texld r8, r7, s0                // get X grid weights
```

```
mad r10.xyz, r8.xyz, r4.x, r10.xyz // fWeighted += (dWeight * X grid weights);

// node 3 of 8
add r9.xyz, r1.xyz, c12.xyz      // node = x, y+1, z
mul r4.y, r2.x, r3.y
mul r4.x, r4.y, r2.z             // dWeight = dWX0 * dWY1 * dWZ0;
mov r6.x, c13.x                  // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x       // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x            // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z      // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw         // get flat xy texture coordinates
texld r8, r7, s0                 // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz // fWeighted += (dWeight * X grid weights);

// node 4 of 8
add r9.xyz, r1.xyz, c12.xyy      // node = x, y+1, z+1
mul r4.x, r4.y, r3.z             // dWeight = dWX0 * dWY1 * dWZ1;
mov r6.x, c13.x                  // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x       // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x            // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z      // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw         // get flat xy texture coordinates
texld r8, r7, s0                 // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz  // fWeighted += (dWeight * X grid weights);

// node 5 of 8
add r9.xyz, r1.xyz, c12.yzz      // node = x+1, y, z
mul r4.y, r3.x, r2.y
mul r4.x, r4.y, r2.z             // dWeight = dWX1 * dWY0 * dWZ0;
mov r6.x, c13.x                  // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x       // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x            // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z      // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw         // get flat xy texture coordinates
```

```
texld r8, r7, s0                // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz  // fWeighted += (dWeight * X grid weights);

// node 6 of 8
add r9.xyz, r1.xyz, c12.yzw     // node = x+1, y, z+1
mul r4.x, r4.y, r3.z            // dWeight = dWX1 * dWY0 * dWZ0;
mov r6.x, c13.x                 // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x      // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x           // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z     // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw        // get flat xy texture coordinates
texld r8, r7, s0                // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz  // fWeighted += (dWeight * X grid weights);

// node 7 of 8
add r9.xyz, r1.xyz, c12.yyz     // node = x+1, y+1, z
mul r4.y, r3.x, r3.y
mul r4.x, r4.y, r2.z            // dWeight = dWX1 * dWY1 * dWZ0;
mov r6.x, c13.x                 // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x      // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x           // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z     // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw        // get flat xy texture coordinates
texld r8, r7, s0                // get X grid weights
mad r10.xyz, r8.xyz, r4.x, r10.xyz  // fWeighted += (dWeight * X grid weights);

// node 8 of 8
add r9.xyz, r1.xyz, c12.yyy     // node = x+1, y+1, z+1
mul r4.x, r4.y, r3.z            // dWeight = dWX1 * dWY1 * dWZ1;
mov r6.x, c13.x                 // add epsilon to avoid frc = 1
mad r6.y, r9.z, c5.w, r6.x      // fRow = z * (1 / nColumns);
frc r6.x, r6.y
add r6.y, r6.y, -r6.x           // fRow = floor(fRow);
mad r6.x, -r6.y, c5.z, r9.z     // fCol = z - (fRow * nColumns);
mad r7.xy, r6.xy, c10.xy, r9.xy  // FlatXY = xy + (RowCol * Cellsize);
mul r7.xy, r7.xy, c10.zw        // get flat xy texture coordinates
```

```
      texld r8, r7, s0                // get X grid weights
      mad r10.xyz, r8.xyz, r4.x, r10.xyz  // fWeighted += (dWeight * X grid weights);

      // r10.xyz finally holds the tensor product
      // so apply the affine transformation
      dp3 r3.x, r5, c0                // fWarped = mul(gAffineXform, posL);
      dp3 r3.y, r5, c1                //    "       "       "      "
      dp3 r3.z, r5, c2                //    "       "       "      "
      add r2.xyz, r3.xyz, c3          // fWarped += gAffineXlate;
      add r2, r2, r10                 // fWarped += fWeighted;

      // still in normalised space so convert back to voxels
      mul r2, r2, c7                  // fWarped *= volume dimensions

      // convert to texture coordinates and look up
      mul_sat r3.xyz, r2.xyz, c4.xyz  // fWarped *= gTextureAdjust;
      add r3.y, c11.y, -r3.y          // tex.y = 1.0 - tex.y for DX9
      texld r0, r3.xyz, s1            // fResult = tex3D(SourceTexture, fWarped);
      mov oC0, r0
    };
  }
}
```

Bibliography

[1] Ackermann, M. J. The Visible Human Project: National Library of Medicine – http://www.nlm.nih.gov. Web Site, 1995.

[2] Aljabar, P., Bhatia, K., Hajnal, J., Boardman, J., Srinivasan, L., Rutherford, M., Dyet, L., Edwards, A., and Rueckert, D. Analysis of growth in the developing brain using non-rigid registration. In *Biomedical Imaging: Macro to Nano, 2006. 3rd IEEE International Symposium on* (6-9 April 2006), pp. 201–204.

[3] Alvarez, N. A., and Sanchiz, J. M. Image registration from mutual information of edge correspondences. *Progress in Pattern Recognition, Image Analysis and Applications, Proceedings 3773* (2005), 528–539.

[4] Andronache, A. S. *Multi-Modal Non-Rigid Registration of Volumetric Medical Images*. PhD thesis, University Politehnica of Bucharest, 2006.

[5] Audette, M. A., Ferrie, F. P., and Peters, T. M. An algorithmic overview of surface registration techniques for medical imaging. *Medical Image Analysis 4*, 3 (2000), 201–217.

[6] Auer, M., Regitnig, P., and Holzapfel, G. A. An automatic non-rigid registration for stained histological sections. *IEEE Trans Image Process 14*, 4 (Apr 2005), 475–486.

[7] Bajcsy, R., and Kovačič, S. Multiresolution elastic matching. *Comput. Vision Graph. Image Process. 46*, 1 (1989), 1–21.

[8] Bale, R. J., Vogele, M., Freysinger, W., Gunkel, A. R., Martin, A., Bumm, K., and Thumfart, W. F. Minimally invasive head holder to improve the performance of frameless stereotactic surgery. *Laryngoscope 107*, 3 (Mar 1997), 373–377.

[9] Bale, R. J., Vogele, M., Martin, A., Auer, T., Hensler, E., Eichberger, P., Freysinger, W., Sweeney, R., Gunkel, A. R., and Lukas, P. H. VBH head holder to improve frameless stereotactic brachytherapy of cranial tumors. *Comput Aided Surg 2*, 5 (1997), 286–291.

[10] Beatson, R. K., and Newsam, G. N. Fast evaluation of radial basis functions : I. *Computers Math. Applic. 24* (1992), 7–19.

[11] Besl, P., and McKay, H. A method for registration of 3-d shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 14*, 2 (Feb. 1992), 239–256.

[12] Betke, M., Hong, H., Thomas, D., Prince, C., and Ko, J. P. Landmark detection in the chest and registration of lung surfaces with an application to nodule registration. *Med Image Anal 7*, 3 (Sep 2003), 265–281.

[13] Bookstein, F. L. Principal warps - thin-plate splines and the decomposition of deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11*, 6 (1989), 567–585.

[14] Bro-Nielsen, M. *Medical Image Registration and Surgery Simulation. PhD thesis.* PhD thesis, Institut for Matematisk Modellering, Danmarks Tekniske Universitet, Lyngby, Denmark, 1996.

[15] Bro-Nielsen, M. Fast finite elements for surgery simulation. *Stud Health Technol Inform 39* (1997), 395–400.

[16] Bro-Nielsen, M., Helfrick, D., Glass, B., Zeng, X., and Connacher, H. Vr simulation of abdominal trauma surgery. *Stud Health Technol Inform 50* (1998), 117–123.

[17] Bro-Nielsen, M., Tasto, J. L., Cunningham, R., and Merril, G. L. PreOp endoscopic simulator: a pc-based immersive training system for bronchoscopy. *Stud Health Technol Inform 62* (1999), 76–82.

[18] Brown, L. G. A survey of image registration techniques. *ACM Computing Surveys 24*, 4 (1992), 325–376.

[19] Buhmann, M. D. *Radial Basis Functions: Theory and Implementations*. Cambridge University Press, 2004.

[20] Bushberg, J., Seibert, J., Leidholdt, E., and Boone, J. *The Essential Physics of Medical Imaging, Second Edition*. Lippincott Williams & Wilkins, 2002. ISBN 0-683-30118-7.

[21] Castro-Pareja, C. R., and Shekhar, R. Hardware acceleration of mutual information-based 3D image registration. *Journal of Imaging Science and Technology 49*, 2 (2005), 105–113.

[22] Chandrashekara, R., Mohiaddin, R., and Rueckert, D. Cardiac motion tracking in tagged MR images using a 4D B-spline motion model and nonrigid image registration. In *Biomedical Imaging: Macro to Nano, 2004. IEEE International Symposium on* (15-18 April 2004), pp. 468–471Vol.1.

[23] Chandrashekara, R., Mohiaddin, R. H., and Rueckert, D. Analysis of 3-D myocardial motion in tagged MR images using nonrigid image

registration. *IEEE Trans Med Imaging 23*, 10 (2004), 1245–50. IEEE transactions on medical imaging.

[24] Chen, M. S., Gonzalez, G., and Lapeer, R. Intra-operative registration for image enhanced endoscopic sinus surgery using photo-consistency. *Stud Health Technol Inform 125* (2007), 67–72.

[25] Chen, M. S., Lapeer, R. J., and Rowland, R. S. Real-time rendering of radially distorted virtual scenes for endoscopic image augmentation. *Stud Health Technol Inform 111* (2005), 87–89.

[26] Christensen, G., Rabbitt, R., and Miller, M. Deformable templates using large deformation kinematics. *Image Processing, IEEE Transactions on 5*, 10 (Oct. 1996), 1435–1447.

[27] Chui, H. L., and Rangarajan, A. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding 89*, 2-3 (2003), 114–141.

[28] Clatz, O., Delingette, H., Talos, I. F., Golby, A. J., Kikinis, R., Jolesz, F. A., Ayache, N., and Warfield, S. K. Robust non-rigid registration to capture brain shift from intraoperative MRI. *IEEE Transactions on Medical Imaging 24*, 11 (2005), 1417–1427.

[29] Collignon, A. *Multi-modality medical image registration by maximization of mutual information, PhD Thesis*. PhD thesis, K.U. Leuven, 1998.

[30] Cooper, J. Optical flow for validating medical image registration. In *in 9th IASTED International Conference on Signal and Image Processing, pp. 502-506, Honolulu, Hawaii, USA, 13-15 August 2003. ACTA Press / IASTED* (2003).

[31] Crum, W., Hartkens, T., and Hill, D. Non-rigid image registration: theory and practice. *The British Journal of Radiology 77* (2004), S140–S153.

[32] Crum, W. R., Hill, D. L. G., and Hawkes, D. J. Information theoretic similarity measures in non-rigid registration. *Inf Process Med Imaging 18* (Jul 2003), 378–387.

[33] D'Agostino, E., Maes, F., Vandermeulen, D., and Suetens, P. A viscous fluid model for multimodal non-rigid image registration using mutual information. In *Lecture Notes in Computer Science: Medical Image Computing and Computer-Assisted Intervention - MICCAI 2002: 5th International Conference, Tokyo, Japan, September 25-28, 2002, Proceedings, Part II* (2002).

[34] Denton, E. R. E., Sonoda, L. I., Rueckert, D., Rankin, S. C., Hayes, C., Leach, M. O., Hill, D. L. G., and Hawkes, D. J. Comparison and evaluation of rigid, affine, and nonrigid registration of breast MR images. *Journal of Computer Assisted Tomography 23*, 5 (1999), 800–805.

[35] Duchon, J. Splines minimizing rotation-invariant semi-norms in Sobolev spaces. *Constructive Theory of Functions of Several Variables, Proc. of Conf., Math. Res. Inst., Oberwolfach (1976), Lecture Notes in Mathematics 571* (1977), 85–100.

[36] Duchon, J. Sur l'erreur d'interpolation des fonctions de plusieurs variables par les d m splines. *Rev. Francaise Automat. Informat. Rech. Oper. Analyse Numerique 12*, 4 (1978), 325–334.

[37] Eggers, G., Muhling, J., and Marmulla, R. Image-to-patient registration techniques in head surgery. *Int J Oral Maxillofac Surg 35*, 12 (Dec 2006), 1081–1095.

[38] Ehrhardt, J., Handels, H., Plotz, W., and Poppl, S. J. Atlas-based recognition of anatomical structures and landmarks and the automatic computation of orthopedic parameters. *Methods Inf Med 43*, 4 (2004), 391–397.

[39] Eldeib, A., Yamany, S., and Farag, A. Multi-modal medical volumes fusion by surface matching. In *Signal Processing and Its Applications, 1999. ISSPA '99. Proceedings of the Fifth International Symposium on* (22-25 Aug. 1999), vol. 1, pp. 439–442vol.1.

[40] Fei, B. W., Duerk, J. L., Sodee, D. B., and Wilson, D. L. Semi-automatic nonrigid registration for the prostate and pelvic MR volumes. *Academic Radiology 12*, 7 (2005), 815–824.

[41] Fei, B. W., Kemper, C., and Wilson, D. L. A comparative study of warping and rigid body registration for the prostate and pelvic MR volumes. *Computerized Medical Imaging and Graphics 27*, 4 (2003), 267–281.

[42] Ferrant, M., Nabavi, A., Macq, B., Jolesz, F. A., Kikinis, R., and Warfield, S. K. Registration of 3-D intraoperative MR images of the brain using a finite-element biomechanical model. *IEEE Trans Med Imaging 20*, 12 (Dec 2001), 1384–1397.

[43] Fischer, B., and Modersitzki, J. Intensity-based image registration with a guaranteed one-to-one point match. *Methods Inf Med 43*, 4 (2004), 327–330.

[44] Fitzpatrick, J. M., West, J. B., and Maurer, C. R. Derivation of expected registration error for point-based rigid-body registration. In *Medical Imaging : Image Processing* (June 1998), K. M. Hanson, Ed., vol. 3338 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pp. 16–27.

[45] Fornefett, M., Rohr, K., and Stiehl, H. Elastic registration of medical images using radial basis functions with compact support. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.* (23-25 June 1999), vol. 1.

[46] Frantz, S., Rohr, K., Stiehl, H., Kim, S., and Weese, J. Validating point-based MR/CT registration based on semi-automatic landmark extraction. In *Proc. Computer Assisted Radiology and Surgery 1999 (CARS'99), Paris, France 23-26 June* (1999), Elsevier Science, Amsterdam, pp. 233–237.

[47] Frantz, S., Rohr, K., and Stiehl, H. S. Development and validation of a multi-step approach to improved detection of 3D point landmarks in tomographic images. *Image and Vision Computing 23* (2005), 956–971.

[48] Gasson, P. D., and Lapeer, R. J. In vitro skin-tissue experiment for increased realism in open surgery simulations. *Stud Health Technol Inform 125* (2007), 143–145.

[49] Gee, J. C., Reivich, M., and Bajcsy, R. Elastically deforming 3d atlas to match anatomical brain images. *J Comput Assist Tomogr 17*, 2 (1993), 225–236.

[50] Gholipour, A., Kehtarnavaz, N., Briggs, R., Devous, M., and Gopinath, K. Brain functional localization: a survey of image registration techniques. *IEEE Trans Med Imaging 26*, 4 (Apr 2007), 427–451.

[51] Gladilin, E., Pekar, V., Rohr, K., and Stiehl, H. A comparison between BEM and FEM for elastic registration of medical images. *Image and Vision Computing 24*, 4 (April 2006), 375–379.

[52] Glozman, D., Shoham, M., and Fischer, A. A surface-matching technique for robot-assisted registration. *Comput Aided Surg 6*, 5 (2001), 259–269.

[53] Guimond, A., Roche, A., Ayache, N., and Meunier, J. Three-dimensional multimodal brain warping using the demons algorithm and adaptive intensity corrections. *IEEE Trans Med Imaging 20*, 1 (Jan 2001), 58–69.

[54] Hajnal, J. V., Hill, D. L. G., and Hawkes, D. J. *Medical Image Registration.* CRC Press, 2001.

[55] Han, Y., and Park, H. Automatic registration of brain magnetic resonance images based on Talairach reference system. *J Magn Reson Imaging 20*, 4 (Oct 2004), 572–580.

[56] Hardy, R. L. Multiquadric equations of topography and other irregular surfaces. *J. Geophys. Res. 76* (1971), 1905–1915.

[57] Hardy, R. L. Theory and applications of the multiquadric-biharmonic method. *Comput. Math. Appl. 19* (1990), 163–208.

[58] Hardy, S. M., Melroy, C., White, D. R., Dubin, M., and Senior, B. A comparison of computer-aided surgery registration methods for endoscopic sinus surgery. *Am J Rhinol 20*, 1 (2006), 48–52.

[59] Harris, M. J., Baxter, W. V., Scheuermann, T., and Lastra, A. Simulation of cloud dynamics on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on graphics hardware, Eurographics association* (2003).

[60] Herline, A. J., Herring, J. L., Stefansic, J. D., Chapman, W. C., Galloway, R. L., and Dawant, B. M. Surface registration for use in interactive, image-guided liver surgery. *Comput Aided Surg 5*, 1 (2000), 11–17.

[61] Hermosillo, G., Chefd'Hotel, C., and Faugeras, O. Variational methods for multimodal image matching. *International Journal of Computer Vision 50*, 3 (2002), 329–343.

[62] Herring, J. L., and Dawant, B. M. Automatic lumbar vertebral identification using surface-based registration. *J Biomed Inform 34*, 2 (Apr 2001), 74–84.

[63] Hill, D. L., Hawkes, D. J., Crossman, J. E., Gleeson, M. J., Cox, T. C., Bracey, E. E., Strong, A. J., and Graves, P. Registration of MR and CT images for skull base surgery using point-like anatomical features. *Br J Radiol 64*, 767 (Nov 1991), 1030–1035.

[64] Hill, D. L., Hawkes, D. J., Gleeson, M. J., Cox, T. C., Strong, A. J., Wong, W. L., Ruff, C. F., Kitchen, N. D., Thomas, D. G., and Sofat, A. Accurate frameless registration of MR and CT images of the head: applications in planning surgery and radiation therapy. *Radiology 191*, 2 (May 1994), 447–454.

[65] Hill, D. L., Hawkes, D. J., Hussain, Z., Green, S. E., Ruff, C. F., and Robinson, G. P. Accurate combination of CT and MR data of the head: validation and applications in surgical and therapy planning. *Comput Med Imaging Graph 17*, 4-5 (1993), 357–363.

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

[67] Holly, L. T., Bloch, O., and Johnson, J. P. Evaluation of registration techniques for spinal image guidance. *J Neurosurg Spine 4*, 4 (Apr 2006), 323–328.

[68] Huang, X. L., Paragios, N., and Metaxas, D. N. Shape registration in implicit spaces using information theory and free form deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 8 (2006), 1303–1318.

[69] Huesman, R. H., Klein, G. J., Kimdon, J. A., Kuo, C., and Majumdar, S. Deformable registration of multimodal data including rigid structures. *IEEE Transactions on Nuclear Science 50*, 3 (2003), 389–392.

[70] Hufner, T., Geerling, J., Kfuri, M., Gansslen, A., Citak, M., Kirchhoff, T., Sott, A. H., and Krettek, C. Computer assisted pelvic surgery: registration based on a modified external fixator. *Comput Aided Surg 8*, 4 (2003), 192–197.

[71] Ibanez, L., Schroeder, W., Ng, L., Cates, J., and The Insight Software Consortium. The ITK software guide, 2003.

[72] Ino, F., Ooyama, K., Takeuchi, A., and Hagihara, K. Design and implementation of parallel nonrigid image registration using off-the-shelf supercomputers. *Medical Image Computing and Computer-Assisted Intervention - Miccai 2003, Pt 1 2878* (2003), 327–334.

[73] Jacobs, M. A., Windham, J. P., Soltanian-Zadeh, H., Peck, D. J., and Knight, R. A. Registration and warping of magnetic resonance images to histological sections. *Med Phys 26*, 8 (Aug 1999), 1568–1578.

[74] Jian, B., Vemuri, B. C., and Marroquin, J. L. Robust nonrigid multimodal image registration using local frequency maps. *Information Processing in Medical Imaging, Proceedings 3565* (2005), 504–515.

[75] Kagadis, G. C., Delibasis, K. K., Matsopoulos, G. K., Mouravliansky, N. A., Asvestas, P. A., and Nikiforidis, G. C. A comparative study of surface- and volume-based techniques for the automatic registration between CT and SPECT brain images. *Med Phys 29*, 2 (Feb 2002), 201–213.

[76] Keeling, S., and Ring, W. Medical image registration and interpolation by optical flow with maximal rigidity. *Journal of Mathematical Imaging and Vision 23*, 1 (July 2005), 47–65.

[77] Khamene, A., Bloch, P., Wein, W., Svatos, M., and Sauer, F. Automatic registration of portal images and volumetric CT for patient positioning in radiation therapy. *Medical Image Analysis 10*, 1 (2006), 96–112.

[78] Klein, S., Staring, M., and Pluim, J. A comparison of acceleration techniques for nonrigid medical image registration. In *International Workshop on Biomedical Image Registration* (2006), vol. 4057 of *Lecture Notes in Computer Science*, pp. 151–159.

[79] Knops, Z. F., Maintz, J. B. A., Viergever, M. A., and Pluim, J. P. W. Normalized mutual information based registration using k-means clustering and shading correction. *Medical Image Analysis 10*, 3 (2006), 432–439.

[80] Kohlrausch, J., Rohr, K., and Stiehl, H. A new class of elastic body splines for nonrigid registration of medical images. *Journal of Mathematical Imaging and Vision 23*, 3 (2005), 253–280.

[81] Krucker, J. F., LeCarpentier, G. L., Fowlkes, J. B., and Carson, P. L. Rapid elastic image registration for 3-D ultrasound. *IEEE Transactions on Medical Imaging 21*, 11 (2002), 1384–1394.

[82] Kybic, J., and Unser, M. Fast parametric elastic image registration. *Image Processing, IEEE Transactions on 12*, 11 (Nov. 2003), 1427–1442.

[83] Lange, T., Eulenstein, S., Hunerbein, M., and Schlag, P.-M. Vessel-based non-rigid registration of MR/CT and 3D ultrasound for navigation in liver surgery. *Comput Aided Surg 8*, 5 (2003), 228–240.

[84] Lapeer, R. J. A mechanical contact model for the simulation of obstetric forceps delivery in a virtual/augmented environment. *Stud Health Technol Inform 111* (2005), 284–289.

[85] Lapeer, R. J., Gasson, P., Florens, J.-L., Laycock, S., and Karri, V. Introducing a novel haptic interface for the planning and simulation of open surgery. *Stud Health Technol Inform 98* (2004), 197–199.

[86] Lapeer, R. J., and Prager, R. W. 3D shape recovery of a newborn skull using thin-plate splines. *Comput Med Imaging Graph 24*, 3 (2000), 193–204.

[87] Lapeer, R. J., and Prager, R. W. Fetal head moulding: finite element analysis of a fetal skull subjected to uterine pressures during the first stage of labour. *J Biomech 34*, 9 (Sep 2001), 1125–1133.

[88] Lapeer, R. J. A. *A biomechanical model of foetal head moulding - PhD Thesis*. PhD thesis, Queens' College Cambridge and Department of Engineering, University of Cambridge, 1999.

[89] Lazebnik, R. S., Lancaster, T. L., Breen, M. S., Lewin, J. S., and Wilson, D. L. Volume registration using needle paths and point

landmarks for evaluation of interventional MRI treatments. *IEEE Trans Med Imaging 22*, 5 (May 2003), 653–660.

[90] Lee, S., Wolberg, G., and Shin, S. Y. Scattered data interpolation with multilevel B-splines. *IEEE Transactions on Visualization and Computer Graphics 3*, 3 (1997), 228–244.

[91] Lee, W.-C. C., Tublin, M. E., and Chapman, B. E. Registration of MR and CT images of the liver: comparison of voxel similarity and surface based registration algorithms. *Comput Methods Programs Biomed 78*, 2 (May 2005), 101–114.

[92] Lefebure, M., and Cohen, L. D. Image registration, optical flow and local rigidity. *Journal of Mathematical Imaging and Vision 14*, 2 (March 2001), 131–147.

[93] Leroy, A., Mozer, P., Payan, Y., and Troccaz, J. Intensity-based registration of freehand 3D ultrasound and CT-scan images of the kidney. *Int J CARS 2* (2007), 31–41.

[94] Levesley, J. Where there's a Will there's a way - the research of Will Light. *Numerical Algorithms 39* (2005), 7–34.

[95] Levin, D., Dey, D., and Slomka, P. Efficient 3D nonlinear warping of computed tomography: two high-performance implementations using OpenGL. In *Medical Imaging 2005: Visualization, Image-Guided Procedures, and Display. Edited by Galloway, Robert L., Jr.; Cleary, Kevin R. Proceedings of the SPIE, Volume 5744, pp. 34-42 (2005).* (April 2005), R. Galloway, Jr. and K. Cleary, Eds., vol. 5744 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pp. 34–42.

[96] Levin, D., Dey, D., and Slomka, P. J. Acceleration of 3D, nonlinear warping using standard video graphics hardware: implementation and initial validation. *Comput Med Imaging Graph 28*, 8 (Dec 2004), 471–483.

[97] Likar, B., and Pernus, F. Automatic extraction of corresponding points for the registration of medical images. *Med Phys 26*, 8 (Aug 1999), 1678–1686.

[98] Likar, B., and Pernus, F. A hierarchical approach to elastic registration based on mutual information. *Image and Vision Computing 19*, 1 (January 2001), 33–44.

[99] Little, J. A., Hill, D. L. G., and Hawkes, D. J. Deformations incorporating rigid structures. In *MMBIA '96: Proceedings of the 1996 Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA '96)* (Washington, DC, USA, 1996), IEEE Computer Society, p. 104.

[100] Livne, O. E., and Wright, G. B. Fast multilevel evaluation of 1-D piecewise smooth radial basis function expansions. *SIAM Proceedings Geometric Design and Computing To appear (2007)* (2005), 0. To appear (2007).

[101] Livne, O. E., and Wright, G. B. Fast multilevel evaluation of smooth radial basis function expansions. *Electronic Transactions on Numerical Analysis 23* (2006), 263–287.

[102] Loeckx, D., Maes, F., Vandermeulen, D., and Suetens, P. Non-rigid image registration using free-form deformations with a local rigidity constraint. *Medical Image Computing and Computer-Assisted Intervention - Miccai 2004, Pt 1, Proceedings 3216* (2004), 639–646.

[103] Maes, F., Collignon, A., Vandermeulen, D., Marchal, G., and Suetens, P. Multimodality image registration by maximization of mutual information. *Medical Imaging, IEEE Transactions on 16*, 2 (April 1997), 187–198.

[104] Maes, F., Vandermeulen, D., and Suetens, P. Medical image registration using mutual information. *Proceedings of the IEEE 91*, 10 (2003), 1699–1722.

[105] Maintz, J., Meijering, E., and Viergever, M. General multimodal elastic registration based on mutual information. In *In: Proc. SPIE Medical Imaging 1998: Image Processing. Bellingham* (1998).

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

[107] Malthan, D., Ehrlich, G., Stallkamp, J., Dammann, F., Schwaderer, E., and Maassen, M. M. Automated registration of partially defective surfaces by local landmark identification. *Comput Aided Surg 8*, 6 (2003), 300–309.

[108] Matsopoulos, G. K., Delibasis, K. K., Mouravliansky, N. A., Asvestas, P. A., Nikita, K. S., Kouloulias, V. E., and Uzunoglu, N. K. CT-MRI automatic surface-based registration schemes combining global and local optimization techniques. *Technol Health Care 11*, 4 (2003), 219–232.

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

[110] Mattes, D., Haynor, D. R., Vesselle, H., Lewellyn, T. K., and Eubank, W. Nonrigid multimodality image registration. In *Proc. SPIE*

Vol. 4322, p. 1609-1620, Medical Imaging 2001: Image Processing, Milan Sonka; Kenneth M. Hanson; Eds. (July 2001), M. Sonka and K. M. Hanson, Eds., vol. 4322 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pp. 1609–1620.

[111] Matuszewski, B. J., Shen, J.-K., Shark, L.-K., and Moore, C. J. Estimation of internal body deformations using an elastic registration technique. *International Conference on Medical Information Visualisation–BioMedical Visualisation (MedVis'06) 0* (2006), 15–20.

[112] Maurer, C. R., Fitzpatrick, J. M., Wang, M. Y., Galloway, R. L., Maciunas, R. J., and Allen, G. S. Registration of head volume images using implantable fiducial markers. *IEEE Trans Med Imaging 16*, 4 (Aug 1997), 447–462.

[113] Maurer, C. R., Maciunas, R. J., and Fitzpatrick, J. M. Registration of head CT images to physical space using a weighted combination of points and surfaces. *IEEE Trans Med Imaging 17*, 5 (Oct 1998), 753–761.

[114] Maurer, C. R., McCrory, J. J., and Fitzpatrick, J. M. Estimation of accuracy in localizing externally attached markers in multimodal volume head images. In *Medical imaging: image processing* (Bellingham, WA., 1993), M. H. Loew, Ed., vol. 1898, SPIE Press, pp. 43–54.

[115] McRobbie, D. W., Moore, E. A., Graves, M. J., and Prince, M. R. *MRI From Picture to Proton*. Cambridge University Press, 2003.

[116] Meinguet, J. Multivariate interpolation at arbitrary points made simple. *Zeitschrift fur Angewandte Mathematik und Physik (ZAMP) 30*, 2 (March 1979), 292–304.

[117] Meyer, C. R., Boes, J. L., Kim, B., Bland, P. H., Zasadny, K. R., Kison, P. V., Koral, K., Frey, K. A., and Wahl, R. L. Demon-

stration of accuracy and clinical versatility of mutual information for automatic multimodality image fusion using affine and thin-plate spline warped geometric deformations. *Med Image Anal 1*, 3 (Apr 1997), 195–206.

[118] Muratore, D. M., Russ, J. H., Dawant, B. M., and Jr., R. L. G. Three-dimensional image registration of phantom vertebrae for image-guided surgery: A preliminary study. *Computer Aided Surgery 7*, 6 (2002), 342–352.

[119] Ourselin, S., Roche, A., Prima, S., and Ayache, N. Block matching: A general framework to improve robustness of rigid registration of medical images. In *MICCAI '00: Proceedings of the Third International Conference on Medical Image Computing and Computer-Assisted Intervention* (2000), Springer-Verlag, pp. 557–566.

[120] Ourselin, S., Stefanescu, R., and Pennec, X. Robust registration of multi-modal images: Towards real-time clinical applications. In *MICCAI '02: Proceedings of the 5th International Conference on Medical Image Computing and Computer-Assisted Intervention-Part II* (2002), Springer-Varlag, pp. 140–147.

[121] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., and Purcell, T. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports* (September 2005), pp. 21–51.

[122] Papademetris, X., Jackowski, A. P., Schultz, R. T., Staib, L. H., and Duncan, J. S. Integrated intensity and point-feature nonrigid registration. *Medical Image Computing and Computer-Assisted Intervention - Miccai 2004, Pt 1, Proceedings 3216* (2004), 763–770.

[123] Peckar, W., Schnorr, C., Rohr, K., and Stiehl, H. Non-rigid image registration using a parameter-free elastic model. In *BMVC98* (1998).

[124] Petrovic, V. S., Cootes, T. F., Twining, C. J., and Taylor, C. Automatic framework for medical image registration, segmentation and modeling. In *Proc. Medical Image Understanding and Analysis MIUA* (2006), vol. 2, pp. 141–145.

[125] Pitiot, A., Malandain, G., Bardinet, E., and Thompson, P. *Piecewise Affine Registration of Biological Images*, vol. 2717 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003.

[126] Pluim, J. P., Maintz, J. B., and Viergever, M. A. Image registration by maximization of combined mutual information and gradient information. *IEEE Trans Med Imaging 19*, 8 (Aug 2000), 809–814.

[127] Pluim, J. P. W., Maintz, J. B. A., and Viergever, M. A. Interpolation artefacts in mutual information-based image registration. *Computer Vision and Image Understanding 77* (2000), 211–232.

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

[129] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes in C, The Art of Scientific Computing, Second Edition*. Cambridge University Press, 1999.

[130] Rietzel, E., and Chen, G. T. Y. Deformable registration of 4D computed tomography data. *Med Phys 33*, 11 (Nov 2006), 4423–4430.

[131] Rivera-Rovelo, J., and Bayro-Corrochano, E. Non-rigid registration and geometric approach for tracking in neurosurgery. *17th International Conference on Pattern Recognition (ICPR'04) 04* (2004), 717–720.

[132] Rohde, G., Healy, D., Berenstein, C., and Aldroubi, A. Interpolation artifacts in biomedical image registration. In *Biomedical Imaging: From Nano to Macro, 2007. ISBI 2007. 4th IEEE International Symposium on* (12-15 April 2007), pp. 648–651.

[133] Rohde, G. K., Aldroubi, A., and Dawant, B. M. The adaptive bases algorithm for intensity-based nonrigid image registration. *IEEE Transactions on Medical Imaging 22*, 11 (2003), 1470–1479.

[134] Rohde, G. K., Berenstein, C. A., and Healy Jr., D. M. Measuring image similarity in the presence of noise. In *Proceedings of the SPIE, vol. 5747, Medical Imaging* (2005), pp. Image Processing; pp. 132–143.

[135] Rohde, G. K., Healy Jr., D. M., Berenstein, C. A., and Aldroubi, A. Measuring image similarity to sub-pixel accuracy. In *Biomedical Imaging: Macro to Nano, 2006. 3rd IEEE International Symposium on* (6-9 April 2006), pp. 638–641.

[136] Rohlfing, T., and Maurer, C. R. Nonrigid image registration in shared-memory multiprocessor environments with application to brains, breasts, and bees. *IEEE Transactions on Information Technology in Biomedicine 7*, 1 (2003), 16–25.

[137] Rohlfing, T., Maurer, C. R., Bluemke, D. A., and Jacobs, M. A. Volume-preserving nonrigid registration of MR breast images using free-form deformation with an incompressibility constraint. *IEEE Transactions on Medical Imaging 22*, 6 (2003), 730–741.

[138] Rohr, K. *Landmark-Based Image Analysis Using Geometric and Intensity Models*. Kluwer Academic Publishers, 2001.

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

[140] Roussos, G., and Baxter, B. J. C. Rapid evaluation of radial basis functions. *Journal of Computational and Applied Mathematics 180* (2005), 51–70.

[141] Rowland, R. S., and Lapeer, R. J. A. 3DView - volumetric visualisation software – http://www.rmrsystems.co.uk/volume_rendering.htm, 2004.

[142] Rueckert, D., Aljabar, P., Heckemann, R., Hajnal, J., and Hammers, A. Diffeomorphic registration using b-splines. In *In Ninth Int. Conf. on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2006)* (2006), Lecture Notes in Computer Science, pp. 702–709.

[143] Rueckert, D., Burger, P., Forbat, S., Mohiaddin, R., and Yang, G. Automatic tracking of the aorta in cardiovascular MR images using deformable models. *Medical Imaging, IEEE Transactions on 16*, 5 (Oct. 1997), 581–590.

[144] Rueckert, D., Chandrashekara, R., Aljabar, P., Bhatia, K., Boardman, J., Srinivasan, L., Rutherford, M., Dyet, L., Edwards, A., Hajnal, J., and Mohiaddin, R. Quantification of growth and motion using non-rigid registration. In *In 2nd International Workshop on Computer Vision Approaches to Medical Image Analysis (CVAMIA 2006)* (2006), Lecture Notes in Computer Science, pp. 49–60.

[145] Rueckert, D., Frangi, A. F., and Schnabel, J. A. Automatic construction of 3-D statistical deformation models of the brain using nonrigid registration. *IEEE Trans Med Imaging 22*, 8 (2003), 1014–25. IEEE transactions on medical imaging.

[146] Rueckert, D., Hayes, C., Studholme, C., Summers, P., Leach, M., and Hawkes, D. J. Non-rigid registration of breast MR images using mutual information. *Medical Image Computing and Computer-Assisted Intervention - Miccai'98 1496* (1998), 1144–1152.

[147] Rueckert, D., Lorenzo-Valdes, M., Chandrashekara, R., Sanchez-Ortiz, G., and Mohiaddin, R. Non-rigid registration of cardiac MR: application to motion modelling and atlas-based segmentation. In *Biomedical Imaging, 2002. Proceedings. 2002 IEEE International Symposium on* (7-10 July 2002), pp. 481–484.

[148] Rueckert, D., Sonoda, L. I., Denton, E., Rankin, S., Hayes, C., Hill, D. L. G., Leach, M., and Hawkes, D. J. Comparison and evaluation of rigid and non-rigid registration of breast MR images. In *In Proc. SPIE Medical Imaging 1999: Image Processing, San Diego, CA* (1999), pp. 78–88.

[149] Rueckert, D., Sonoda, L. I., Hayes, C., Hill, D. L. G., Leach, M. O., and Hawkes, D. J. Nonrigid registration using free-form deformations: Application to breast MR images. *IEEE Transactions on Medical Imaging 18*, 8 (1999), 712–721.

[150] Russakoff, D. B., Rohlfing, T., Adler, J. R., and Maurer, C. R. Intensity-based 2D-3D spine image registration incorporating a single fiducial marker. *Acad Radiol 12*, 1 (Jan 2005), 37–50.

[151] Sabisch, T., Ferguson, A., and Bolouri, H. Automatic landmark extraction using self-organising maps. In *IEE/BMVA Proceedings of the 1st annual conference on medical image understanding and analysis (MIUA'97), Oxford UK* (1997), pp. 157–160.

[152] Schestowitz, R., Twining, C., Cootes, T., Petrovic, V., Taylor, C., and Crum, W. Assessing the accuracy of non-rigid registration with and without ground truth. In *3rd IEEE International Symposium on Biomedical Imaging: Macro to Nano* (2006), pp. 836–839.

[153] Schestowitz, R. S., Twining, C. J., Petrovic, V. S., Cootes, T., Crum, B., and Taylor, C. Non-rigid registration assessment without ground truth. In *Proc. Medical Image Understanding and Analysis MIUA* (2006), vol. 2, pp. 151–155.

[154] Schnabel, J. A., Rueckert, D., Quist, M., Blackall, J. M., Castellano-Smith, A. D., Hartkens, T., Penney, G. P., Hall, W. A., Liu, H., Truwit, C. L., Gerritsen, F. A., Hill, D. L. G., and Hawkes, D. J. A generic framework for non-rigid registration based on non-uniform multi-level free-form deformations. In *MICCAI '01: Proceedings of the 4th International Conference on Medical Image Computing and Computer-Assisted Intervention* (2001), Springer-Verlag, pp. 573–581.

[155] Schnabel, J. A., Tanner, C., Castellano-Smith, A. D., Degenhard, A., Leach, M. O., Hose, D. R., Hill, D. L. G., and Hawkes, D. J. Validation of nonrigid image registration using finite-element methods: Application to breast MR images. *IEEE Transactions on Medical Imaging 22*, 2 (2003), 238–247.

[156] Schreibmann, E., and Xing, L. Image registration with auto-mapped control volumes. *Medical Physics 33*, 4 (2006), 1165–1179.

[157] Shekhar, R., Walimbe, V., Raja, S., Zagrodsky, V., Kanvinde, M., Wu, G., and Bybel, B. Automated 3-dimensional elastic registration of whole-body PET and CT from separate or combined scanners. *J Nucl Med 46*, 9 (Sep 2005), 1488–1496.

[158] Shekhar, R., and Zagrodsky, V. Mutual information-based rigid and nonrigid registration of ultrasound volumes. *IEEE Transactions on Medical Imaging 21*, 1 (2002), 9–22.

[159] Shekhar, R., Zagrodsky, V., Castro-Pareja, C. R., Walimbe, V., and Jagadeesh, J. M. High-Speed Registration of Three- and Four-dimensional Medical Images by Using Voxel Similarity. *Radiographics 23*, 6 (2003), 1673–1681.

[160] Shuhaiber, J. H. Augmented reality in surgery. *Arch Surg 139*, 2 (Feb 2004), 170–174.

[161] Sivaramakrishna, R., Krucker, J. F., Meyer, C. R., LeCarpentier, G. L., Fowlkes, J. B., and Carson, P. L. 3D spatial compounding of ultrasound images using image-based nonrigid registration. *Technology in Cancer Research & Treatment 26*, 9 (2000), 1475–1488.

[162] Slagmolen, P., Loeckx, D., Roels, S., Geets, X., Maes, F., Haustermans, K., and Suetens, P. Nonrigid registration of multitemporal CT and MR images for radiotherapy treatment planning. *Biomedical Image Registration, Proceedings 4057* (2006), 297–305.

[163] Slomka, P. J., Dey, D., Przetak, C., Aladl, U. E., and Baum, R. P. Automated 3-dimensional registration of stand-alone (18)F-FDG whole-body PET with CT. *J Nucl Med 44*, 7 (Jul 2003), 1156–1167.

[164] Stefanescu, R., Pennec, X., and Ayache, N. Grid powered nonlinear image registration with locally adaptive regularization. *Medical Image Analysis 8*, 3 (2004), 325–342.

[165] Studholme, C., Hill, D. L. G., and Hawkes, D. J. An overlap invariant entropy measure of 3D medical image alignment. *Pattern Recognition 32* (1999), 71–86.

[166] Sturm, B., Powell, K. A., Stillman, A. E., and White, R. D. Registration of 3D CT angiography and cardiac MR images in coronary artery disease patients. *Int J Cardiovasc Imaging 19*, 4 (Aug 2003), 281–293.

[167] Tang, S. Y., Chen, Y. W., Xu, R., Wang, Y., Morikawa, S., and Kurumi, Y. MR-CT image registration in liver cancer treatment with an open configuration MR scanner. *Biomedical Image Registration, Proceedings 4057* (2006), 289–296.

[168] Tang, S. Y., and Jiang, T. Z. Nonrigid registration of medical image by linear singular blending techniques. *Pattern Recognition Letters 25*, 4 (2004), 399–405.

[169] Thirion, J. P. Image matching as a diffusion process: an analogy with maxwell's demons. *Med Image Anal 2*, 3 (Sep 1998), 243–260.

[170] Trancoso, P., and Charalambous, M. Exploring graphics processor performance for general purpose applications. In *In Proceedings of the Euromicro Symposium on Digital System Design, Architectures, Methods and Tools (DSD 2005), IEEE Computer Society* (2005), pp. 306–313.

[171] Tsao, J. Interpolation artifacts in multimodality image registration based on maximization of mutual information. *IEEE Trans Med Imaging 22*, 7 (July 2003), 854–864.

[172] Tustison, N. J., Avants, B. B., Sundaram, T. A., Duda, J. T., and Gee, J. C. A generalization of Free-Form Deformation image registration within the ITK finite element framework. *Biomedical Image Registration, Proceedings 4057* (2006), 238–246.

[173] Tustison, N. J., and Gee, J. C. N-D c^k B-spline scattered data approximation. The Insight Journal - http://hdl.handle.net/1926/140, November 2005.

[174] Tustison, N. J., and Gee, J. C. Generalized n-D c^k B-spline scattered data approximation with confidence values. In *MIAR* (2006), G.-Z. Yang, T. Jiang, D. Shen, L. Gu, and J. Yang, Eds., vol. 4091 of *Lecture Notes in Computer Science*, Springer, pp. 76–83.

[175] Urschler, M., Zach, C., Ditt, H., and Bischof, H. Automatic point landmark matching for regularizing nonlinear intensity registration: application to thoracic CT images. *Med Image Comput Comput Assist Interv Int Conf Med Image Comput Comput Assist Interv 9*, Pt 2 (2006), 710–717.

[176] Viola, P. *Alignment by maximization of mutual information - PhD Thesis*. PhD thesis, M.I.T. Artificial Intelligence Laboratory, 1995.

[177] Viola, P., and Wells, W. M. Alignment by maximization of mutual information. *International Journal of Computer Vision 24*, 2 (1997), 137–154.

[178] Wahba, G. *Spline models for observational data*. Society for Industrial and Applied Mathematics, 1990. ISBN 0-89871-244-0.

[179] Wang, D., Strugnell, W., Cowin, G., Doddrell, D. M., and Slaughter, R. Geometric distortion in clinical MRI systems Part I:

evaluation using a 3D phantom. *Magnetic Resonance Imaging 22* (2004), 1211–1221.

[180] Wang, F., Vemuri, B. C., and Eisenschenk, S. J. Joint registration and segmentation of neuroanatomic structures from brain MRI. *Academic Radiology 13*, 9 (2006), 1104–1111.

[181] Wang, X., and Feng, D. D. Automatic hybrid registration for 2-dimensional CT abdominal images. *Third International Conference on Image and Graphics (ICIG'04) 00* (2004), 208–211.

[182] Warfield, S. K., Nabavi, A., Butz, T., Tuncali, K., Silverman, S. G., Black, P. M., Jolesz, F. A., and Kikinis, R. Intraoperative segmentation and nonrigid registration for image guided therapy. *Medical Image Computing and Computer-Assisted Intervention - Miccai 2000 1935* (2000), 176–185.

[183] Wells III, W. M., Viola, P., Atsumi, H., Nakajima, S., and Kikinis, R. Multi-modal volume registration by maximization of mutual information. *Medical Image Analysis 1*, 1 (1996), 35–51.

[184] West, J., Fitzpatrick, J. M., Wang, M. Y., Dawant, B. M., Maurer, C. R., Kessler, R. M., and Maciunas, R. J. Retrospective intermodality registration techniques for images of the head: surface-based versus volume-based. *IEEE Trans Med Imaging 18*, 2 (Feb 1999), 144–150.

[185] Winter, S., Brendel, B., Rick, A., Stockheim, M., Schmieder, K., and Ermert, H. Registration of bone surfaces, extracted from CT-datasets, with 3D ultrasound. *Biomed Tech (Berl) 47 Suppl 1 Pt 1* (2002), 57–60.

[186] Wolfsberger, S., Rossler, K., Regatschnig, R., and Ungersbock, K. Anatomical landmarks for image registration in frameless stereotactic neuronavigation. *Neurosurg Rev 25*, 1-2 (Mar 2002), 68–72.

[187] Wollny, G., and Kruggel, F. Computional cost of nonrigid registration algorithms based on fluid dynamics. *IEEE Transactions on Medical Imaging 21*, 8 (2002), 946–952.

[188] Worz, S., and Rohr, K. Localization of anatomical point landmarks in 3D medical images by fitting 3D parametric intensity models. *Med Image Anal 10*, 1 (Feb 2006), 41–58.

[189] Worz, S., and Rohr, K. Physics-based elastic image registration using splines and including landmark localization uncertainties. *Med Image Comput Comput Assist Interv Int Conf Med Image Comput Comput Assist Interv 9*, Pt 2 (2006), 678–685.

[190] Wright, G. B. Personal communication. Email correspondence, July 2007.

[191] Xiao, G. F., Brady, J. M., Noble, J. A., Burcher, M., and English, R. Nonrigid registration of 3-D free-hand ultrasound images of the breast. *IEEE Transactions on Medical Imaging 21*, 4 (2002), 405–412.

[192] Xie, Z., and Farin, G. E. Image registration using hierarchical B-splines. *IEEE Trans Vis Comput Graph 10*, 1 (2004), 85–94.

[193] Zagorchev, L., and Goshtasby, A. A comparative study of transformation functions for nonrigid image registration. *IEEE Transactions on Image Processing 15*, 3 (2006), 529–538.

[194] Zhan, Y., Feldman, M., Tomaszeweski, J., Davatzikos, C., and Shen, D. Registering histological and MR images of prostate for image-

based cancer detection. *Med Image Comput Comput Assist Interv Int Conf Med Image Comput Comput Assist Interv 9*, Pt 2 (2006), 620–628.

[195] Zhang, J., Levesque, M. F., Wilson, C. L., Harper, R. M., Engel, J., Lufkin, R., and Behnke, E. J. Multimodality imaging of brain structures for stereotactic surgery. *Radiology 175*, 2 (May 1990), 435–441.

[196] Zitova, B., and Flusser, J. Image registration methods: a survey. *Image and Vision Computing 21*, 11 (2003), 977–1000.