NTNU

Norwegian University of
Science and Technology

Fast Non-Rigid Registration Using Polynomial Expansion with Diffeomorphic Constraints Applied to Cardiac Ultrasound Volumes

Marta Havåg Ranestad

Master of Science in Electronics
Submission date: June 2017
Supervisor: Ilangko Balasingham, IES
Co-supervisor: Gabriel Kiss, ISB

Norwegian University of Science and Technology
Department of Electronic Systems

1 | Abstract

This thesis attempted to develop a fast, non-rigid, diffeomorphic image registration implementation and was written as a continuation of a project that was completed during the fall of 2016 [1] which developed and implemented a rigid image registration code. Non-rigid diffeomorphic image registration is already used in many diagnostic and research related medical situations, however due to the computational demands it is only used as a post-processing tool. This thesis focused mostly on how it could be used with 3D echocardiography to estimate a vector field representation of the movement of the heart, preferably in real-time.

To find these vector fields, Farnebäck's optical flow algorithm was used to decompose the images and represent them as polynomials that could be used to compute the local deformation for each voxel. Both a linear and a quadratic polynomial were used simultaneously, but it was found that using only the quadratic polynomial provided better results.

An iterative scheme was proposed to allow one deformation estimate to be used as a priori information for the estimate in the following iteration, and three different accumulation methods were implemented to make this iteration possible; additive, composite and exponential composite. The additive method simply added together the new estimate to the previously accumulated estimate while the composite method combined the new estimate and the accumulated estimate through linear interpolation. The exponential composite method found a recursive accumulation method by solving a linear differential with flow constraints. While it was found that the exponential composite provided more accurate and diffeomorphic results, it was almost twice as computationally expensive as the other two methods due to its recursive implementation.

The image decomposition and the image registration required a large number of matrix multiplications and inversions which made the algorithm a prime candidate for parallel programming, so the implementation was completed entirely on the GPU using OpenCL.

Theoretical diffeomorphic vector fields were created to test the code for accuracy and computational expense and an optimal set-up was found that was tested on real cases as well. The real cases showed deformation fields that were mostly diffeo-

morphic which indicated that the code had calculated realistic deformation fields. A comparison between running the code on a laptop computer and a desktop computer showed very different timing and indicated that real-time applicability of the method developed in this thesis will be highly dependent on the hardware of the device it is run on. However, both the real and theoretical cases indicated a high level of accuracy even with large deformations, so it is therefore reasonable to assume that in high frame rate cases, some frames in the 3D echocardiographic image sets can be skipped to achieve accurate, real-time results.

2 | Sammendrag

Denne oppgaven forsøkte å utvikle en rask, elastisk og diffeomorfisk bilderegistreringsimplementasjon og ble skrevet som en fortsettelse av et prosjekt som ble gjennomført i høsten, 2016 [1]. Dette prosjektet utviklet og implementerte en rigid bilderegistreringskode. Elastisk, diffeomorfisk bilderegistrering har allerede blitt tatt i bruk i mange diagnostiske og forskningsrelaterte medisinske situasjoner, men blir brukt som et post-prosesseringsverktøy på grunn av den høye tidsbruken. Denne masteroppgaven fokuserte mest på hvordan elastisk, diffeomorfisk bilderegistrering kunne bli brukt med 3D ekkokardiografi for å estimere en vektorfeltrepresentasjon av hjertebevegelse, helst i sanntid.

For å finne disse vektorfeltene, ble Farnebäcks optiske flytalgoritme brukt for å dekomponere bildene og representere de som polynomer. Disse polynomene ble brukt til å kalkulere den lokale deformasjonen for hver enkelt voksel. Både et lineært og kvadratisk polynom ble testet i bruk samtidig og alene, men resultatene viste seg å bli bedre når kun et kvadratisk polynom ble brukt.

For å bedre resultatene, ble deformasjonen estimert iterativt ved at et estimat ble matet inn i neste iterasjon som a priori informasjon. Tre forskjellige akkumuleringsmetoder gjorde iterering mulig; additive, 'composite', og eksponensiell 'composite'. Den additive metoden la kun sammen det nye estimated med det akkumulerte estimatet fra forrige iterasjon mens 'composite' metoden kombinerte det nye og det akkumulerte estimatet ved hjelp av lineær interpolering. Den eksponensielle 'composite' metoden fant en rekursiv akkumuleringsmetode ved å løse en lineær differensiallikning med initialbetingelser for flyt. Selv om resultatene antydet at den eksponensielle 'composite' metoden viste mer nøyaktige og diffeomorfiske resultater, brukte den nesten dobbelt så lang tid som de to andre akkumuleringsmetodene på grunn av den rekursive implementeringen.

Bildedekomponeringen og bilderegistreringen krevde mye matrisemultiplisering og invertering. Dette gjorde algoritmen godt egnet til parallelprogrammering, så implementasjonen ble skrevet fullstending på GPUen ved hjelp av OpenCL med gode resultater for tidsbruk.

Teoretiske diffeomorfiske vektorfelt ble generert for å teste kodens nøyaktighet og tidsbruk, og et optimalt oppsett ble funnet og brukt til testing av reelle 3D

ekkokardiografiske bilder. De reelle bildene viste estimerte deformasjonsfelt som var diffeomorfiske over nesten hele feltet, noe som indikerte at koden kalkulerte realistiske deformasjonsfelt. Ved å sammenligne tidsbruken til koden når den ble kjørt på en laptop og en stasjonær datamaskin, ble det klart at sanntidsapplikasjon av denne metoden ville være veldig avhengig av hardwaren den ble kjørt på. Til tross for dette, viste både de reelle og teoretiske testresultatene at nøyaktigheten var høy selv for store deformasjoner. Det er derfor rimelig å anta at i situasjoner med høy bilderate vil det være mulig å hoppe over noen bilder for å oppnå et nøyaktig estimat i sanntid.

3 | Acknowledgements

I would like to thank my advisor, Gabriel Kiss, without whose patience, enthusiasm and knowledge this thesis never would have been completed.

Contents

4 | Introduction

Ultrasound imaging is the safest and least damaging medical imaging modality and is therefore very useful in medical diagnostics and research. By using non-rigid image registration in conjunction with ultrasound imaging one can develop multi-modal imaging (e.g. fusion of ultrasound and magnetic resonance imaging (MRI)), study organ development over time, study anatomical differences between patients, diagnose patients and many other applications [2]. To properly diagnose and treat a patient, real-time information about the movement of a certain organ can be elemental and non-rigid image registration can achieve this. Treating liver lesions is an example of this, where the movement of the liver due to the respiratory system will affect the precision of the lesion targeting [3]. Non-rigid image registration can also be used for strain imaging in the heart to determine the active and passive movements of the heart muscles which, in turn, can be used to assess regional function and to perform quantitative deformation measurements which relate to muscle contractility [4]. While real-time implementation of image registration is unneccessary if used in post-processing, it will widen the possible areas of use such as diagnostics or treatment if it is fast as well as accurate.

In this thesis, the focus is finding a deformation field of the heart using 3D echocardiographic images. To find the movement of a region or point of interest, two images of the same heart at different times of the heart cycle must be aligned and the vector field that describes this will represent the deformation between the two frames. Rigid image registration can prove helpful before the non-rigid image registration because it can estimate the large transformations between the images and it can be used as a priori information for a non-rigid algorithm which can lead to a reduction in computational expense. In the following both rigid and non-rigid registration methods present in the literature are summarised.

Multiple rigid image registration solutions to finding a deformation field have been proposed and one is the method in [5] which finds the alignment between two images using normalised cross correlation (NCC). This is a voxel based method that is independent of contrast change between the images and finds the rotation and translation between the two images that maximises the normalised cross correlation. The method uses image pyramids to lower the computational expense by first com-

puting the NCC at higher pyramid levels and then passing the result on to lower levels for refinement purposes. The aim of this method in this article is to expand the field of view of 3D echocardiographic volumes, so it is applied to temporally synchronised images along the heart cycle, but the method could also be used to find the deformation between two images that are taken at different times in the heart cycle.

Another rigid solution to expand the field of view of 3D echocardiographic images was proposed in [6] and [7], and this method used the Farnebäck optical flow method to represent the images as a polynomial. Before the Farnebäck algorithm was performed on the images, the images were thresholded in order to reduce the influence of background noise, and Gaussian image pyramids were used to lower computational expense as in [5]. A polynomial expansion was then found on all image pyramid levels using the Farnebäck algorithm. The assumption was made that one image of the heart was a rigid, affine transformation of another image of the same heart, and this made it possible to find the displacement between images using the coefficients of the polynomial expansions of the images. The displacement was calculated by finding the local transformations in the overlapping parts of the two images and then averaging them. The entire code was written in MATLAB. The aim of the method in this article was also to expand the field of view by first finding the deformation and then fusing the two images together. The method provided accurate results, but its speed made it unsuitable for use in a real-time scenario.

I completed a project [1] during the fall of 2016 built on the method used in [6] and [7] and attempted to make it faster. The project used two different image pyramid types created on the CPU, a quadratic polynomial image decomposition performed entirely on the GPU using OpenCL, and a linear and affine rigid image registration performed mostly on the GPU using OpenCL. The results showed that the pyramid scheme was essential to pick up large deformations as well as to reduce computational expense, and while the affine registration provided more accurate results than the linear registration the computational expense was much higher. The project also determined that the graphics card used for implementation was elemental in reaching the real-time goal of the code.

A solution for non-rigid image registration is presented in [8]. This method uses the Farnebäck optical flow method like [6], [7] and [1] but with a linear polynomial instead of quadratic. Its image registration method is also similar, but in this article, the registration is non-rigid and thus requires a different accumulation method than the previously mentioned methods. [8] presents three different accumulation methods; additive, composite and exponential composite. The additive method adds the estimated deformation field to the previous estimation while the composite finds the linear interpolation between the two. The exponential composite method finds a diffeomorphic vector field, which is a smooth and invertible field and will be described

in more detail in Section 5.3. This was achieved by using two flow constraints to accumulate the field recursively. The study focused on the exponential composite field accumulation and found that it led to mostly diffeomorphic fields at the cost of high computational expense.

A well known method for strain estimation is speckle tracking. This method searches for the best match of a template in a search window using a sum of squared differences or normalised cross correlation. The downside to speckle tracking is that it typically results in a very noisy raw deformation field that requires heavy smoothing. Additionally, the maximum deformation it can pick up is limited by the size of the search window which will explode in computational expense if it is very large. Speckle tracking somtimes results in a negative Jacobian determinant and therefore a non-diffeomorphic estimation of heart movement, which is not natural because the mass of the myocardium should be constant [9].

Another non-rigid image registration method is completed in [10] based on speckle tracking and the mutual information of voxel intensities. The mutual information of two random variables is a statistical measure that quantifies to what degree one variable can be defined by the information given by the other variable. In this case, the mutual information is based on B-splines and is used as a type of smoothing to determine whether the deformation found by speckle tracking can be trusted or not.

The demons algorithm is another algorithm that calculates non-rigid registration. The demons algorithm works by defining the velocity of each pixel by the intensity differences and gradients of its neighbourhood. The obtained velocity field is smoothed by a Gaussian and then used to iteratively transform the moving image into the reference image. The algorithm works well for computed tomography (CT) and MRI, but remains challenging for ultrasound imaging [11].

This thesis is a continuation of the project in [1]. The rigidity constraint is relaxed and the local deformation between voxel pairs is computed by matching the polynomial coefficients in the reference and moving volumes to determine a deformation field. An iterative scheme is adopted using the three different accumulation methods described in [8]. However, in contrast to prior work both the linear and quadratic terms of the decomposition are considered in order to increase the accuracy of the method. Image pyramids are not used and everything is performed on the GPU using OpenCL to reduce the computational expense as much as possible. The code is tested on simulated diffeomorphic deformation fields to determine its accuracy and timing, and on several real cases as. The tests were all done on 3D images, but would work just the same for 2D images as well.

5 | Materials and Method

Any non-rigid registration algorithm requires three components: a measure to compare the similarity of two voxels (including their neighborhood), a method to find a new voxel in the moving image that matches the voxel in the reference image and a field accumulation method to update the local deformation after each iteration. In this work the similarity measure was based on the Farnebäck coefficients, this similarity measure was used to estimate a translation between two voxels, and the field was accumulated using three different methods; an additive method, a composite method and an exponential composite method.

5.1 Image Decomposition

The image was decomposed by the method of polynomial expansion which is based on normalised convolution [12]. The idea behind polynomial expansion is to fit a basis function to the neighborhood of each pixel and generate a first or second order approximation that is optimal in a least squares sense. The following approximation was chosen for each pixel location:

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{x} + c \tag{5.1}$$

where $\mathbf{x}$ is a position vector, $\mathbf{x} = (x, y, z)$, $\mathbf{A}$ is a matrix, $\mathbf{b}$ is a vector and c is a constant [13].

All of the coefficients in equation 5.1 were estimated based on two predetermined parameters called the applicability and the certainty. The applicability is used to decide how neighbouring pixels will be weighted in the polynomial expansion, while the certainty denotes how accurate a measured value is [14]. As the certainty of each voxel was not known in the images used for this experiment, all voxels had the same certainty and the code was simplified significantly.

5.1.1 Gaussian Kernels

Gaussian kernels were used while creating the image decomposition to smooth out the image. This was necessary so that the extracted information in the decompo-

sition was based on the relevant features and less susceptible to the speckle noise that is typical to ultrasound acquisitions. The Gaussian kernels were also related to the applicability of the decomposition so that a larger kernel would ensure a larger neighbourhood around a voxel was included in the decomposition. The width, n, of the kernel determined the highest translation that the image registration would be able to pick up, and σ determined the standard deviation of the kernel.

5.2 Similarity Metric

To estimate the displacement between a reference image and a moving image, both images needed to be represented by polynomials as follows

$$f_r(\mathbf{x}) = \mathbf{x}^T \mathbf{A}_r \mathbf{x} + \mathbf{b}_r^T \mathbf{x} + c_r \tag{5.2}$$

$$f_m(\mathbf{x}) = \mathbf{x}^T \mathbf{A}_m \mathbf{x} + \mathbf{b}_m^T \mathbf{x} + c_m \tag{5.3}$$

where the subscripts on the functions and the coefficients denote whether the polynomials represent the moving or the reference image. $f_m(\mathbf{x})$ can be represented as a function of $f_r(\mathbf{x})$

$$\begin{aligned} f_m(\mathbf{x}) &= f_r(\mathbf{x} - \mathbf{d}) \\ &= \mathbf{x} - \mathbf{d}^T \mathbf{A}_r (\mathbf{x} - \mathbf{d}) + \mathbf{b}_r^T (\mathbf{x} - \mathbf{d}) + c_r \\ &= \mathbf{x}^T \mathbf{A}_r \mathbf{x} + (\mathbf{b}_r - 2\mathbf{A}_r \mathbf{d})^T \mathbf{x} + \mathbf{d}^T \mathbf{A}_r \mathbf{d} - \mathbf{b}_r^T \mathbf{d} + c_r \\ &= \mathbf{x}^T \mathbf{A}_m \mathbf{x} + \mathbf{b}_m^T \mathbf{x} + c_m \end{aligned} \tag{5.4}$$

This leads to

$$\mathbf{A}_m = \mathbf{A}_r \tag{5.5}$$

$$\mathbf{b}_m = \mathbf{b}_r - 2\mathbf{A}_r \mathbf{d} \tag{5.6}$$

which can be used to find an expression for the displacement, $\mathbf{d}$

$$\mathbf{d} = \frac{1}{2} \mathbf{A}_r^{-1} (\mathbf{b}_r - \mathbf{b}_m) \tag{5.7}$$

While equation 5.7 assumes we have a global polynomial for the entire image, we have a polynomial representation for each *voxel* in the image. Thus, each coefficient is a function of $\mathbf{x}$; $\mathbf{A}(\mathbf{x})$, $\mathbf{b}(\mathbf{x})$ and $c(\mathbf{x})$. Equation 5.5 assumes $\mathbf{A}_m = \mathbf{A}_r$, but this assumption is unlikely to hold when $f_m(\mathbf{x})$ and $f_r(\mathbf{x})$ are sets of polynomials for each voxel instead of a single polynomial for the entire image. Therefore, new variables are introduced

$$\mathbf{A}(\mathbf{x}) = \frac{\mathbf{A}_m(\mathbf{x}) + \mathbf{A}_r(\mathbf{x})}{2} \tag{5.8}$$

$$\Delta\mathbf{b}(\mathbf{x}) = \frac{1}{2}(\mathbf{b}_r(\mathbf{x}) - \mathbf{b}_m(\mathbf{x})) \tag{5.9}$$

which leads to the following constraint

$$\mathbf{A}(\mathbf{x})\mathbf{d} = \Delta\mathbf{b} \tag{5.10}$$

A minimal squared error approach is proposed

$$\epsilon^2 = \sum_x ||\mathbf{A}(\mathbf{x})\mathbf{d} - \Delta\mathbf{b}(\mathbf{x})||^2 \tag{5.11}$$

which results in a least squares solution

$$\mathbf{d} = \left(\sum \mathbf{A}(\mathbf{x})^T\mathbf{A}(\mathbf{x})\right)^{-1} \sum \mathbf{A}(\mathbf{x})^T\Delta\mathbf{b}(\mathbf{x}) \tag{5.12}$$

This displacement estimation method can be extended to include a priori knowledge which allows the estimation to be done iteratively as well. If a priori knowledge is included, equations 5.8 and 5.9 must be modified with a new $\mathbf{x}$ for the moving image

$$\tilde{\mathbf{x}} = \mathbf{x} + \tilde{\mathbf{d}}(\mathbf{x}) \tag{5.13}$$

where $\tilde{\mathbf{d}}(\mathbf{x})$ is the rounded a priori estimate.

Thus, equations 5.8 and 5.9 become

$$\mathbf{A}(\mathbf{x}) = \frac{\mathbf{A}_m(\mathbf{x}) + \mathbf{A}_r(\tilde{\mathbf{x}})}{2} \tag{5.14}$$

$$\Delta\mathbf{b}(\mathbf{x}) = \frac{1}{2}(\mathbf{b}_r(\mathbf{x}) - \mathbf{b}_m(\tilde{\mathbf{x}})) + \mathbf{A}(\mathbf{x})\tilde{\mathbf{d}}(\mathbf{x}) \tag{5.15}$$

and equations 5.11 and 5.12 still hold.

5.2.1 Combining a Quadratic and a Linear Polynomial

The constraint in equation 5.10 loses accuracy with large translations and an attempt is therefore made to weight it to reduce the effect of this inaccuracy as follows

$$\mathbf{A}(\mathbf{x}) = \lambda_2 \frac{\mathbf{A}_m(\mathbf{x}) + \mathbf{A}_r(\tilde{\mathbf{x}})}{2} \tag{5.16}$$

$$\Delta\mathbf{b}(\mathbf{x}) = \lambda_2 \left(\frac{1}{2}(\mathbf{b}_r(\mathbf{x}) - \mathbf{b}_m(\tilde{\mathbf{x}})) + \mathbf{A}(\mathbf{x})\tilde{\mathbf{d}}(\mathbf{x})\right) \tag{5.17}$$

In addition to this weighting, a linear polynomial expansion is proposed in addition to the quadratic polynomial expansion to reduce the errors further. The linear model represents the reference and moving images as follows

$$f_r(\mathbf{x}) = \mathbf{b}_r^T\mathbf{x} + c_r \tag{5.18}$$

$$\begin{aligned} f_m(\mathbf{x}) &= f_r(\mathbf{x} - \mathbf{d}) \\ &= \mathbf{b}_r^T \mathbf{x} + c_r - \mathbf{b}_r^T d \\ &= \mathbf{b}_m^T \mathbf{x} + c_m \end{aligned} \tag{5.19}$$

Similarly to the quadratic model, this leads to

$$\mathbf{b}_m = \mathbf{b}_r \tag{5.20}$$

$$c_m = c_r - \mathbf{b}_r^T d \tag{5.21}$$

and

$$\mathbf{b}(\mathbf{x}) = \lambda_1 \frac{\mathbf{b}_m(\mathbf{x}) + \mathbf{b}_r(\tilde{\mathbf{x}})}{2} \tag{5.22}$$

$$\Delta c(\mathbf{x}) = \lambda_1 (c_r(\mathbf{x}) - c_m(\mathbf{x}) + \mathbf{b}(\mathbf{x})^T \tilde{\mathbf{d}}(\mathbf{x})) \tag{5.23}$$

The linear and quadratic parts are added together to create a minimal squared error approach

$$\epsilon^2 = \sum_x \beta_1 (\mathbf{b}(\mathbf{x})^T \mathbf{d} - \Delta c(\mathbf{x}))^2 + \beta_2 ||\mathbf{A}(\mathbf{x})\mathbf{d} - \mathbf{\Delta b}(\mathbf{x})||^2 \tag{5.24}$$

where β_1 and β_2 are the weights for the linear and quadratic parts respectively. This results in the following least squares solution [12]

$$\mathbf{G} = \sum x \beta_1 \mathbf{b}(\mathbf{x})^T \mathbf{b}(\mathbf{x}) + \beta_2 \mathbf{A}(\mathbf{x})^T \mathbf{A}(\mathbf{x}) \tag{5.25}$$

$$\mathbf{h} = \sum x \beta_1 \mathbf{b}(\mathbf{x}) \Delta c(\mathbf{x}) + \beta_2 \mathbf{A}(\mathbf{x})^T \Delta \mathbf{b}(\mathbf{x}) \tag{5.26}$$

$$\mathbf{d} = \mathbf{G}^{-1} \mathbf{h} \tag{5.27}$$

5.2.2 Non-rigid approach

To change equations 5.25, 5.26 and 5.27 into a non-rigid approach where the displacement of each voxel is found, the following equations are proposed

$$\mathbf{G}_{avg}(\mathbf{x}) = (\mathbf{G} * w)(\mathbf{x}) \tag{5.28}$$

$$\mathbf{h}_{avg}(\mathbf{x}) = (\mathbf{h} * w)(\mathbf{x}) \tag{5.29}$$

$$\mathbf{d}_{avg}(\mathbf{x}) = \mathbf{G}_{avg}(\mathbf{x})^{-1} \mathbf{h}_{avg}(\mathbf{x}) \tag{5.30}$$

where equations 5.25, 5.26 and 5.27 are calculated for a neighbourhood around a voxel. $\mathbf{w}(\mathbf{x})$ is a weighting function for the elements in the neighborhood and in our case a Gaussian kernel was used . The size of this neighbourhood is chosen by the user; a bigger neighborhood will yield smoother results, however if the size of the neighborhood is too big local deformations might be lost [8].

5.2.3 Thresholding

To reduce the interference of noise on the image registration algorithms, a thresholding variable was introduced. In our case, the main focus was the alignment of the wall of the left ventricle. In order to reduce the influence of signal coming from the blood pool inside the ventricle, a threshold was applied to the ultrasound volumes. This variable prevented voxels of a lower value than the threshold from being included in the calculations of equations 5.28, 5.29 and 5.30. This also lead to a reduction in computational expense.

5.3 Diffeomorphism and the Jacobian Determinant

Before presenting the different field accumulation methods, diffeomorphism must be discussed. A vector field is diffeomorphic if it is invertible, and the field and its inverse are differentiable. The Jacobian determinant can be used to determine diffeomorphism as follows

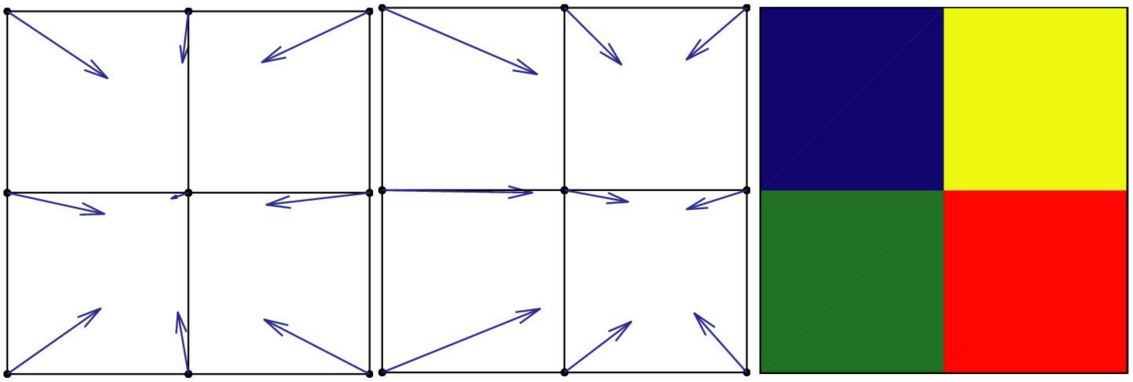
$det(J) < 0$	the voxel has been given a negative volume and "inverted" and is not diffeomorphic
$det(J) = 0$	the voxel has disappeared, so it is not invertible and therefore not diffeomorphic
$0 < det(J) < 1$	the voxel has decreased in volume by a factor equal to $det(J)$
$det(J) = 1$	the voxel has the same volume as before it was deformed
$det(J) > 1$	the voxel has increased in volume by a factor equal to $det(J)$

It is therefore clear that as long as the Jacobian determinant is positive, the vector field is diffeomorphic [15].

When discussing the vector field that represents the movement of the heart, it is necessary to consider whether such a field is diffeomorphic or not because an accurate representation of the deformation of the heart must be diffeomorphic. This is because the heart muscles are incapable of "inverting" as a voxel with a negative Jacobian determinant does and muscle cells of the heart never disappear like a voxel with a zero Jacobian determinant would; the heart is only physically capable of moving in a manner that a diffeomorphic vector field can represent.

5.4 Field Accumulation

To create an iterative scheme for each point in the image, a field accumulation method must be used. Three different methods were implemented and tested; additive, composite and exponential composite. A visual explanation of the different methods is presented using the fields and the reference image in Figure 5.1 [15].



(a) Vector field 1 (b) Vector field 2 (c) Reference image

Figure 5.1: Vector fields and reference image for visual explanation [15]

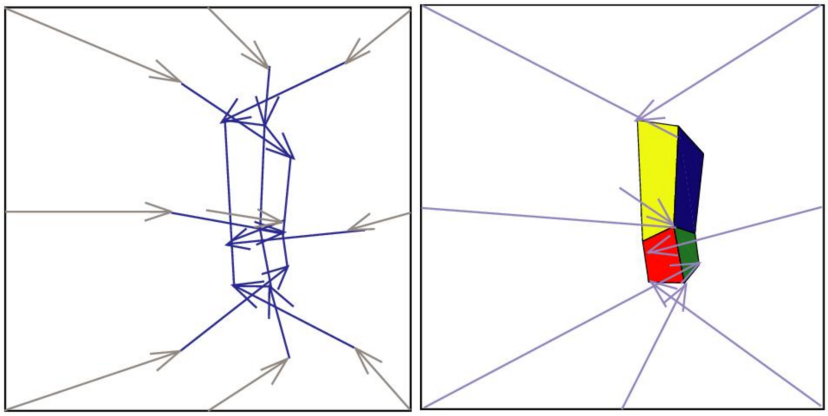
The field accumulation methods could also be used to include a priori information.

5.4.1 Additive

The simplest field accumulation method, the additive method was created by adding the new estimated displacement vector, $\mathbf{d} = (x_d, y_d, z_d)$, to the previously estimated accumulated displacement vector, $\mathbf{v_n} = (x_{v_n}, y_{v_n}, z_{v_n})$ as follows

$$\mathbf{v_{n+1}} = \mathbf{v_n} + \mathbf{d} \tag{5.31}$$

With the vector fields and reference image presented in Figure 5.1, the additive field for vector field 1 and 2 is presented in Figure 5.2 where the blue vector arrows represent vector field 1 and the grey arrows represent vector field 2. The effect of using this additive field to deform the reference image is also presented in the estimated moving image in Figure 5.2



(a) Additive vector field (b) Estimated moving image

Figure 5.2: Effect of additive vector field on reference image [15]

Figure 5.2 shows that in addition to being skewed, the reference image has been flipped. Thus it is clear that an additive method for vector field accumulation does not guarantee diffeomorphism.

5.4.2 Composite

The composite field accumulation method is achieved by first deforming the accumulated deformation, $\mathbf{v_n}$, by the new estimated deformation, $\mathbf{d}$, and then adding the deformed accumulated deformation to the new estimated deformation [8] as follows

$$\mathbf{v_{n+1}} = \mathbf{v_n} \Diamond \mathbf{d} + \mathbf{d} \tag{5.32}$$

where $\mathbf{v_n} \Diamond \mathbf{d}$ means $\mathbf{v_n}$ is linearly interpolated with $\mathbf{d}$ [8].

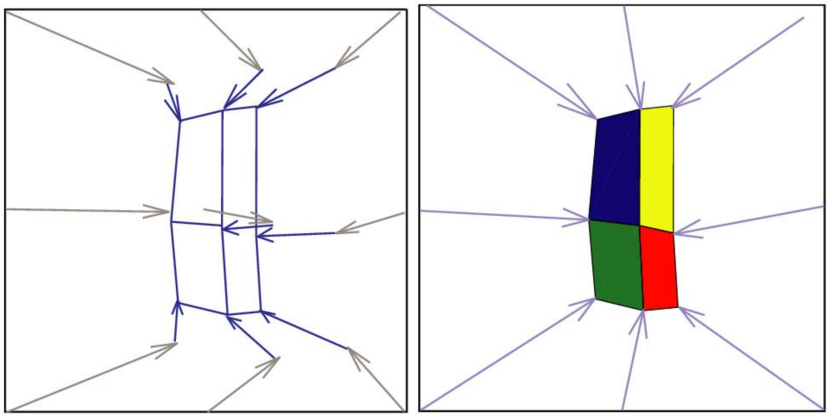
The reference image and vector fields in Figure 5.1 are again used to visually represent the composite field accumulation method. Figure 5.3 shows the composite field for vector field 1 and 2, where the blue vector arrows represent field 1 and the grey arrows represent field 2, and the effect of this composite field on the reference image is also shown in the moving image in the same figure.

With this method, the composite vector field will be diffeomorphic if both the original fields are diffeomorphic [8]

5.4.3 Exponential Composite

In the exponential composite method, it has been found that if $\mathbf{u}(t) = \Phi_d(\mathbf{x}, t)$ is the solution of the linear differential equation

$$\frac{d}{dt}\mathbf{u}(t) = \mathbf{d}(\mathbf{u}) \tag{5.33}$$



(a) Composite vector field (b) Estimated moving image

Figure 5.3: Effect of composite vector field on reference image [15]

$$\mathbf{u}(0) = \mathbf{x} \tag{5.34}$$

where $\mathbf{d}$ is a smooth vector field and $\mathbf{x}$ is a point, $\Phi_d(\mathbf{x}, t)$ represents diffeomorphic flow. This can be used to find the diffeomorphic displacement field because the flow of $\mathbf{d}$ at time $t = 1$ is the exponential of d. This can be approximated as follows

$$exp(\mathbf{d}) = exp(2^{-k}\mathbf{d})^{2^k} \tag{5.35}$$

and implemented in three steps.

1. a factor 2^{-k} must be chosen such that $max(2^{-k}\mathbf{d}) < 0.5$.
2. the composite method from Equation 5.32 is used where $\mathbf{d}$ is replaced with $\mathbf{d} \cdot 2^{-k}$
3. multiplying the exponential by itself 2^k times as shown in Equation 5.35 is done by performing k recursive composite field accumulations of the flow as shown in Step 2

After these three steps are performed, the resulting vector field must be composed in the same manner as the estimated deformation field in Section 5.4.2 [8].

Figure 5.4 shows a theoretical comparison between the exponential composite field accumulation method and the additive field accumulation method.

In Figure 5.4, the arrow labeled $\Delta(x)$ represents the additive accumulation method while the arrows labeled $exp(D)(x)$ represent the exponential composite accumulation method. The diffeomorphic field is represented by the arrows labeled $\Phi_D(x, t)$. It can be seen in Figure 5.4 and it has been shown in [15] that the exponential composite accumulation method more clearly approximates this diffeomorphic field than the additive method.

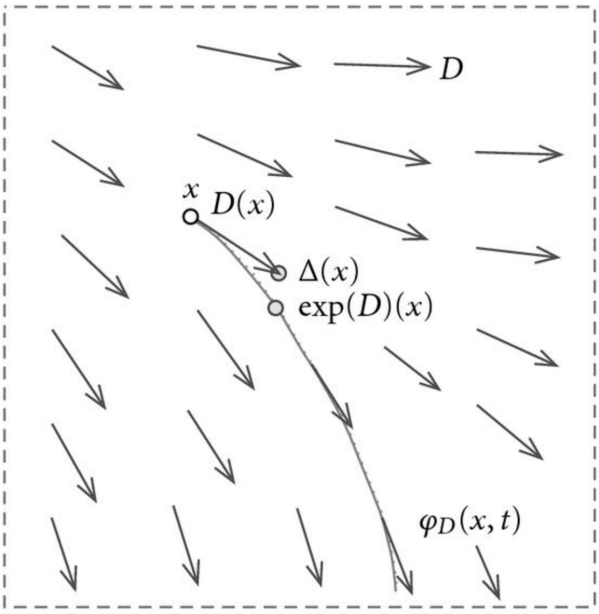


Figure 5.4: Comparison between additive and exponential composite field accumulation methods [15]

5.5 Implementation on the GPU

The computation of the image decomposition and registration requires the calculation of many matrix multiplications and inversions. This type of code lends itself well to parallel programming, so to reduce computational expense, the image decomposition, image registration and the field accumulation were all implemented on the GPU using OpenCL.

Before explaining the OpenCL implementation, some words should be defined:

kernel — A function on the GPU in OpenCL

work-items — The individual neighbourhoods on which the kernels are executed, in this case that is each voxel of the image

work-groups — The groupings of work-items that are executed in parallel. The GPU executes work-groups in sizes that are a multiple of 2 (typically 16, 32 or 64), therefore the number of work-groups should be a multiple of 2 to fill up the executable work-group slots on the GPU [16]

Figure 5.5 shows the typical architecture of a GPU processor when used for parallel computing. The green squares are work-items, and the boxes around the green squares (labeled SMX) are work-groups which are executed in parallel.

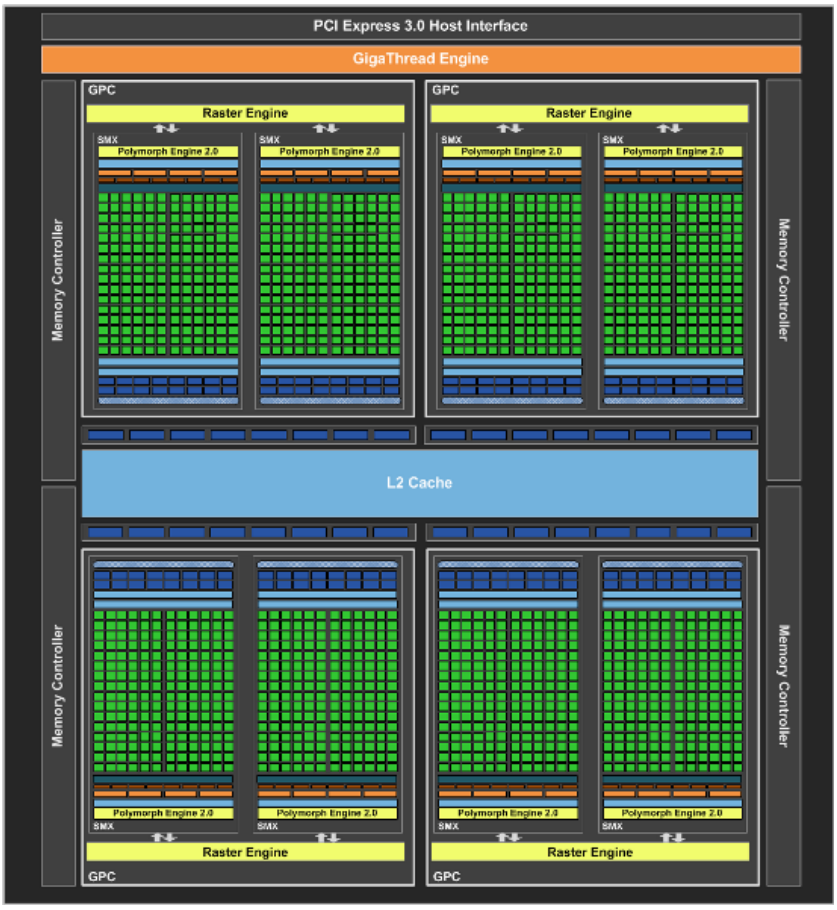


Figure 5.5: Figure representing GPU architecture when used for parallel programming [17]

5.5.1 Image Decomposition

The image decomposition code represents each voxel by a quadratic polynomial like Equation 5.1, so $\mathbf{A}$, $\mathbf{b}$ and c are found for each voxel, and saved in a buffer on the GPU to be used later in the image registration. To complete the image decomposition, multiple things need to be calculated and their dependence on each other makes it impossible to set up just one parallel programming scheme for the entire polynomial expansion method.

First, the Gaussian filter kernels used for image smoothing are created and copied to buffers. Their inverses are also copied to buffers.

Secondly, the polynomial expansion algorithm must be set up on the CPU side before being passed onto the GPU and OpenCL. In this case, the work-groups were divided such that there were 16 work-groups in the x-direction, 4 in the y-direction and 4 in the z-direction. This resulted in $16 * 4 * 4 = 256$ work-groups in total. The

Gaussian filter kernel is separable, so each voxel could be filtered one plane at a time (x-, y- or z-plane) instead of by three nested for-loops. The number of operations necessary to filter the voxels one plane at a time would be $W * H * D * (R + C + L)$ where W, H and D are the width, height and depth of the image, respectively, and R, C and L are the number of rows, columns and layers of the Gaussian filter kernel. If the image was filtered with nested for-loops, the number of operations could be represented by $W * H * D * (R * C * L)$, which would be a significantly higher number of operations for large images and large filter kernels. Therefore, the smoothing was done in three separate passes on the GPU instead of one. As a fourth pass, the polynomial decomposition was calculated of the smoothed image.

5.5.2 Image Registration

For the image registration the OpenCL code was divided into 8 work-groups in the x-, y- and z-direction. In the z-direction, however, each work-item represented more than one voxel so that in the OpenCL kernel, the registration was completed on only one voxel in the x- and y-dimension, but multiple voxels in the z-dimension. Because of this, a for-loop had to be used in the kernel to ensure all the voxels in the z-direction were registered and therefore the computational expense suffered. This had to be done because the laptop used to find the results could not handle the number of parallel threads required to allow each voxel to be a separate work-item. With a more powerful graphics card, it might be possible to represent each voxel by a separate work-item.

5.5.3 Field Accumulation

The field accumulation methods were integrated as kernels into the image registration code. The work-group set-up was therefore determined by the image registration and the methods were written as simple kernels on the GPU-side following the methods explained in Sections 5.4.1, 5.4.2 and 5.4.3. While most of the implementations of these methods was relatively straight forward, the implementation of the linear interpolation could be done in two different ways; using buffers or using built-in functions in the OpenCL texture memory. For simplicity, buffers were used, but using the built-in functions in OpenCL would likely have been less computationally expensive. Using the OpenCL function would have required changing vectors containing the vector field values to three images, one for each vector dimension, and then the OpenCL texture memory could have been used to complete the linear interpolation easily. It is reasonable to assume that this would be faster than the buffer method because the texture memory has been developed for gaming and has therefore been optimised when it comes to computational expense [18].

6 | Results

To determine the optimal set-up for high accuracy and low timing, different parameters were varied and the results on simulated deformations with known ground truth were gathered and compared. After an optimal set of parameters was found the remaining results were gathered with the same parameters and unless otherwise specified, the β values were set to $\beta_1 = \beta_2 = 1.0$, the number of iterations was 5, the maximum theoretical deformation was 3 voxels, the size of the neighbourhood included in the image registration for each voxel was $n = 1$ and the threshold was set to 0.5. The Gaussian kernel used for smoothing the image before image decomposition had a size of $n = 9$. In all the images, the voxel size was 0.7mm. It was also observed that k in Equation 5.35 in Section 5.4.3, which represented the number of recursive composite accumulations completed in the exponential composite method was never higher than 5.

All the results were gathered on a MacBook Pro with an Intel HD Graphics 4000 graphics card with 1536MB of memory. The MacBook could compute a work-group size of maximum 512 work-groups and it had 16 parallel compute units. The OpenCL version that was used was OpenCL 1.2.

The echocardiographic datasets were acquired using the GE Vivid E9 system (GE Vingmed, Horten, Norway) and a 2.5MHz matrix array transducer (GE Vingmed, Horten, Norway), with the subjects in the left lateral decubitus position. Scans were taken from the apical window, in harmonic mode, from 4 up to 6 QRS triggered sub-volumes, during an end-expiratory breath-hold. The depth and angle of the ultrasound sector were adjusted such that the entire LV was visualized. Resulting 3D data sets contained 24 to 32 frames per cardiac cycle and were stored digitally for further processing.

Box plots are used throughout this section to show estimation trends. The lower edge of the box is the first quartile (the maximum of the lowest 25% of the data), the line in the middle of the box is the median and the upper edge is the third quartile (the maximum of the lowest 75% of the data). The red plus signs are outliers in the data.

6.1 β

The β values refer to Equations 5.25, 5.26 and 5.27. Varying the relationship between the β_1 and β_2 varies how heavily the quadratic polynomial of the image decomposition is weighted against the linear polynomial.

6.1.1 Magnitude Error

First the effect of varying the β levels on the magnitude error between the theoretical deformation and the computed deformation was calculated and the results are shown in a box plot in Figure 6.1. The statistical values of the box plot are shown in Table 6.1.

Accumulation method	$\beta_1 and \beta_2 relationship$	Quartile 1	Median	Quartile 3
Additive	$\beta_1 = 0.0, \beta_2 = 1.0$	0.1335	0.1844	0.2815
	$\beta_1 = 0.5, \beta_2 = 1.0$	0.1339	0.1852	0.2899
	$\beta_1 = 1.0, \beta_2 = 1.0$	0.1339	0.1853	0.2917
	$\beta_1 = 1.0, \beta_2 = 0.5$	0.1340	0.1855	0.2935
	$\beta_1 = 1.0, \beta_2 = 0.0$	0.1333	0.1842	0.2851
Composite	$\beta_1 = 0.0, \beta_2 = 1.0$	0.1350	0.1874	0.3066
	$\beta_1 = 0.5, \beta_2 = 1.0$	0.1354	0.1883	0.3237
	$\beta_1 = 1.0, \beta_2 = 1.0$	0.1354	0.1884	0.3280
	$\beta_1 = 1.0, \beta_2 = 0.5$	0.1355	0.1886	0.3321
	$\beta_1 = 1.0, \beta_2 = 0.0$	0.1346	0.1868	0.3125
Exponential Composite	$\beta_1 = 0.0, \beta_2 = 1.0$	0.1350	0.1873	0.3052
	$\beta_1 = 0.5, \beta_2 = 1.0$	0.1352	0.1880	0.3185
	$\beta_1 = 1.0, \beta_2 = 1.0$	0.1353	0.1881	0.3220
	$\beta_1 = 1.0, \beta_2 = 0.5$	0.1353	0.1882	0.3248
	$\beta_1 = 1.0, \beta_2 = 0.0$	0.1341	0.1858	0.2951

Table 6.1: Table showing the statistical values for the box plot of the magnitude error with varying β_1 and β_2 values

6.1.2 Jacobian Determinant

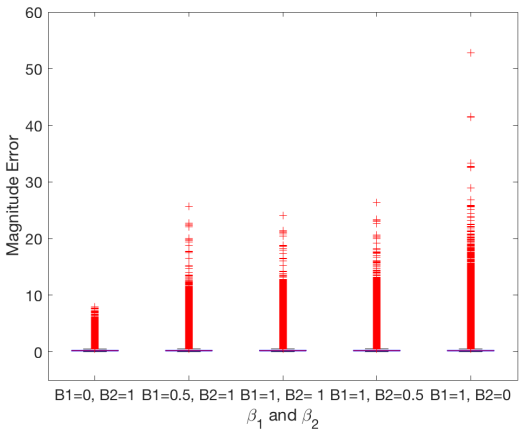
The effect of the beta relationship on the Jacobian determinant, and therefore the diffeomorphism of the image, is shown in Figure 6.2. For all these box plots, quartile 1, the median and quartile 3 are equal to 1.

6.1.3 Timing

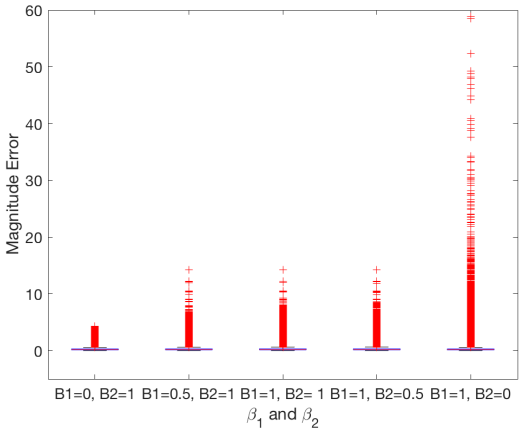
The timing was also estimated for varying beta relationships and the results are shown in Figure 6.2.

β_1 and β_2 relationship	Additive method[ms]	Composite method[ms]	Exponential Composite method[ms]
$\beta_1 = 0.0, \beta_2 = 1.0$	3911.5	4131.1	6472.1
$\beta_1 = 0.5, \beta_2 = 1.0$	3993.9	4173.5	6559.8
$\beta_1 = 1.0, \beta_2 = 1.0$	3916.4	4213.1	6668.9
$\beta_1 = 1.0, \beta_2 = 0.5$	3910.9	4176.1	6608.5
$\beta_1 = 1.0, \beta_2 = 0.0$	3856.2	4141.7	4099.8

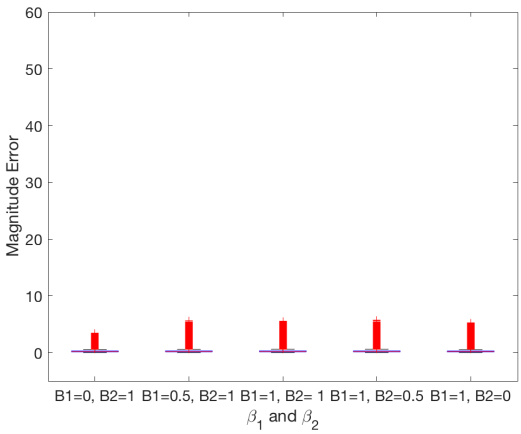
Table 6.2: Table showing the timing of the different cumulative methods with varying β_1 and β_2 values



(a) Additive method

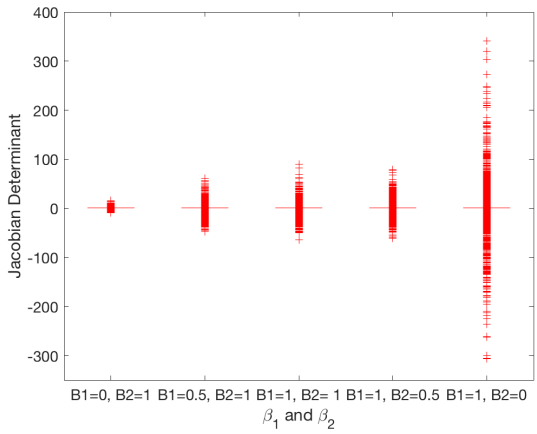


(b) Composite method

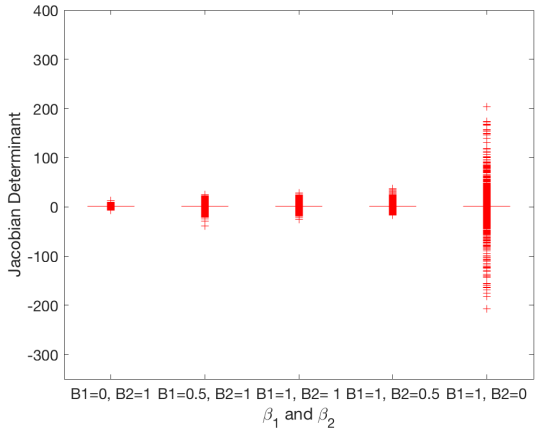


(c) Exponential composite method

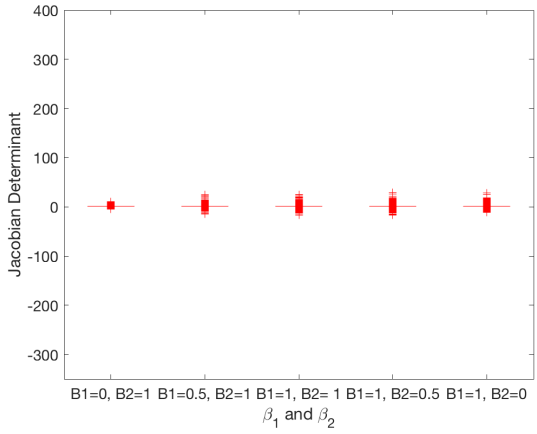
Figure 6.1: Box plot showing the magnitude of error with varying β_1 and β_2 values



(a) Additive method



(b) Composite method



(c) Exponential composite method

Figure 6.2: Box plot showing the Jacobian determinant with varying β_1 and β_2 values

6.2 Number of iterations

The number of iterations the image registration is performed over can affect the maximum deformation the image registration algorithm is able to pick up, so the effect of the number of iterations on accuracy and timing was found. These effects were found for both a threshold equal to 0.5 and 0.

6.2.1 Magnitude Error

First, the magnitude error between a theoretical and the estimated deformation field was found when the threshold was equal to 0.5 and the results were gathered in Figure 6.3 with the statistical values in Table 6.3

Accumulation method	Number of iterations	Quartile 1	Median	Quartile 3
Additive	1	0.1311	0.1796	0.2573
	2	0.1320	0.1813	0.2650
	3	0.1327	0.1829	0.2747
	4	0.1334	0.1841	0.2819
	5	0.1339	0.1853	0.2917
Composite	1	0.1311	0.1796	0.2573
	2	0.1334	0.1843	0.2822
	3	0.1342	0.1860	0.2989
	4	0.1348	0.1872	0.3083
	5	0.1354	0.1884	0.3280
Exponential Composite	1	0.1325	0.1824	0.2705
	2	0.1333	0.1841	0.2807
	3	0.1341	0.1858	0.2959
	4	0.1347	0.1869	0.3041
	5	0.1353	0.1881	0.3220

Table 6.3: Table showing the statistical values for the box plot of the magnitude error with varying number of iterations

With threshold equal to zero, the results became what is shown in Figure 6.4 and Table 6.4.

6.2.2 Jacobian Determinant

The Jacobian determinant was also found for a varying number of iterations and the results for the threshold equal to 0.5 are shown in Figure 6.5. In this case, quartile

Accumulation method	Number of iterations	Quartile 1	Median	Quartile 3
Additive	1	0.1551	0.2604	1.2062
	2	0.1560	0.2657	1.1112
	3	0.1571	0.2770	1.2410
	4	0.1578	0.2819	1.2078
	5	0.1586	0.2940	1.2796
Composite	1	0.1551	0.2604	1.2062
	2	0.1629	0.3673	1.1477
	3	0.1642	0.4265	1.3221
	4	0.1650	0.4267	1.2602
	5	0.1660	0.4762	1.3971
Exponential Composite	1	0.1620	0.3588	1.1974
	2	0.1630	0.3650	1.1085
	3	0.1642	0.4271	1.2919
	4	0.1649	0.4138	1.1941
	5	0.1658	0.4677	1.3497

Table 6.4: Table showing the statistical values for the box plot of the magnitude error with varying number of iterations and threshold equal to zero

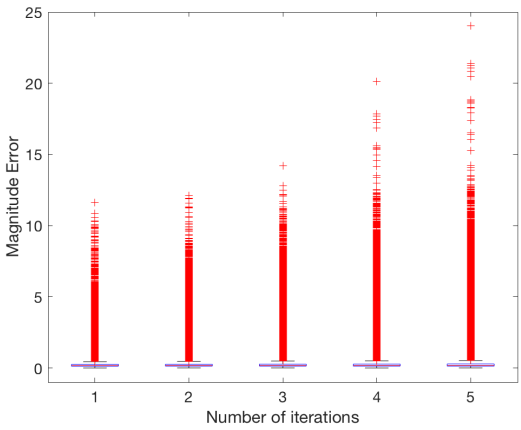
1, the median and quartile 3 were all equal to 1. The results with threshold equal to zero shown in Figure 6.6 and Table 6.5.

6.2.3 Timing

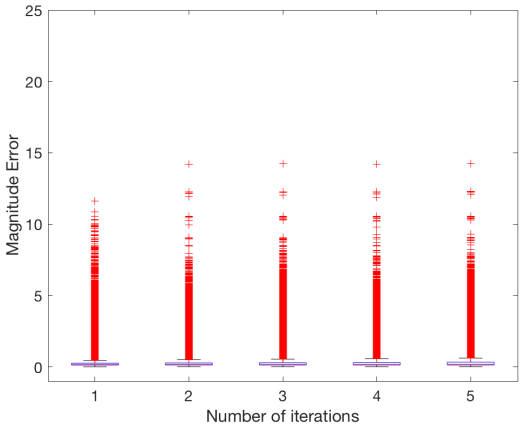
Varying the number of iterations was assumed to affect the timing, but results were gathererd to see how much and they are shown in Figure 6.7 for threshold equal to 0.5 and in Figure 6.8 for threshold equal to 0.

Accumulation method	Number of iterations	Quartile 1	Median	Quartile 3
Additive	1	0.9491	1.0000	1.0000
	2	0.9433	1.0000	1.0002
	3	0.9319	1.0000	1.0011
	4	0.9287	1.0000	1.0024
	5	0.9210	1.0000	1.0037
Composite	1	0.9491	1.0000	1.0000
	2	0.9052	1.0000	1.0075
	3	0.8845	1.0000	1.0084
	4	0.8779	1.0000	1.0089
	5	0.8640	1.0000	1.0105
Exponential Composite	1	0.9169	1.0000	1.0038
	2	0.9111	1.0000	1.0062
	3	0.8938	1.0000	1.0078
	4	0.8941	1.0000	1.0093
	5	0.8807	1.0000	1.0109

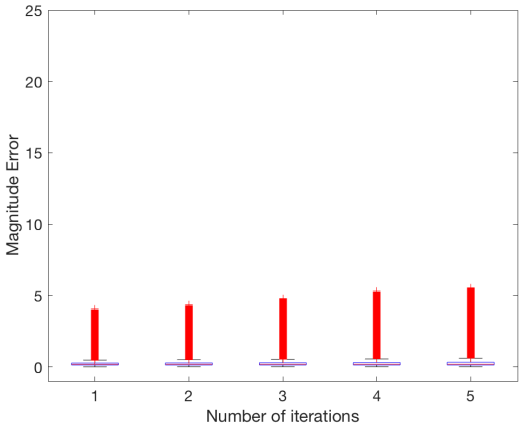
Table 6.5: Table showing the statistical values for the box plot of the Jacobian determinant with varying number of iterations and threshold equal to zero



(a) Additive method

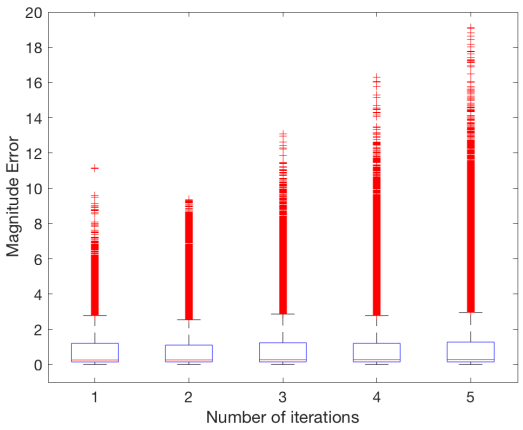


(b) Composite method

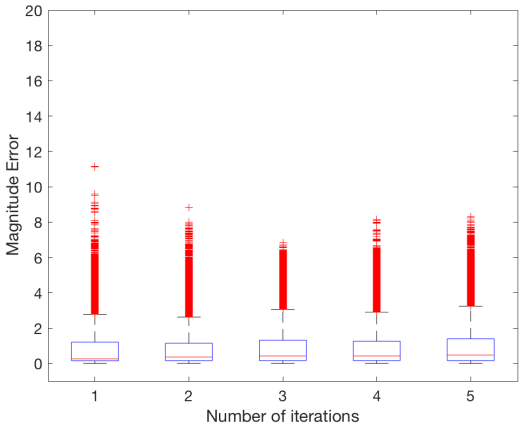


(c) Exponential composite method

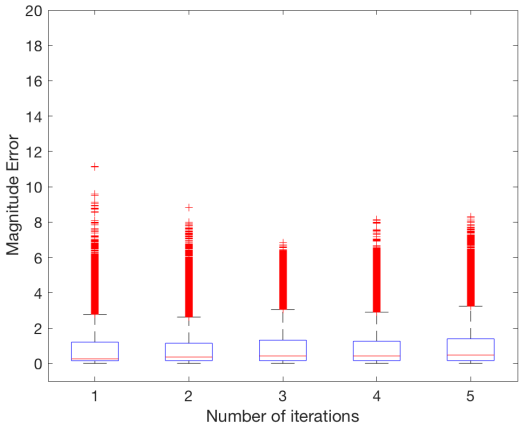
Figure 6.3: Box plot showing the magnitude of error with varying number of iterations



(a) Additive method

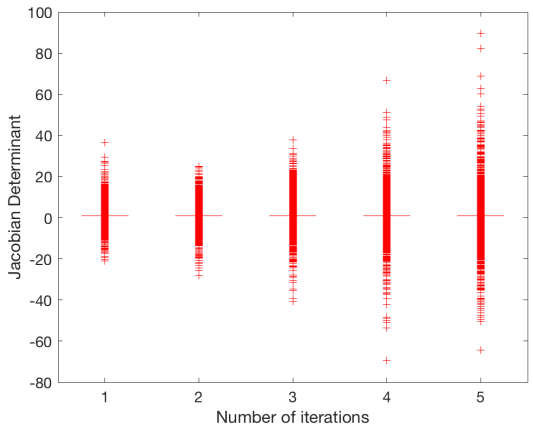


(b) Composite method

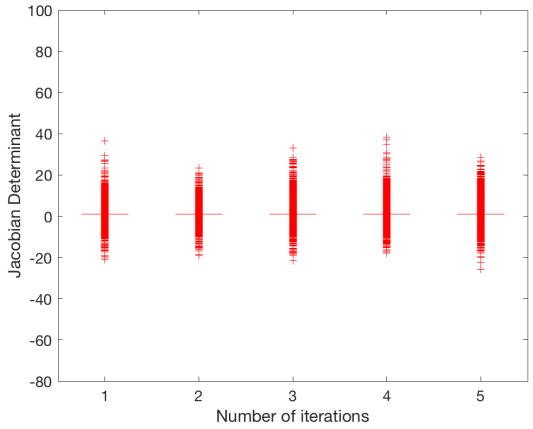


(c) Exponential composite method

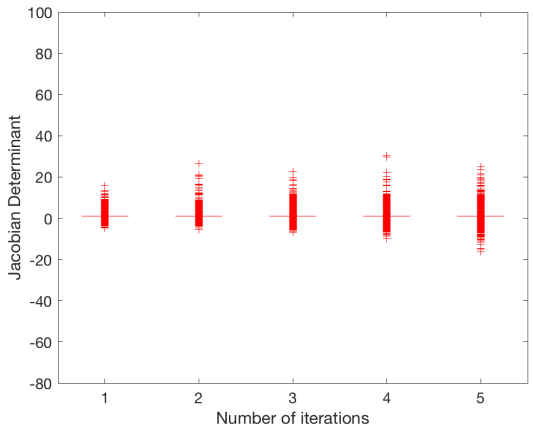
Figure 6.4: Box plot showing the magnitude of error with varying number of iterations and threshold equal to zero



(a) Additive method

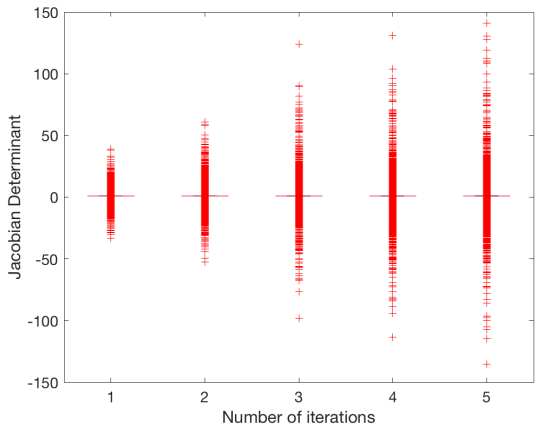


(b) Composite method

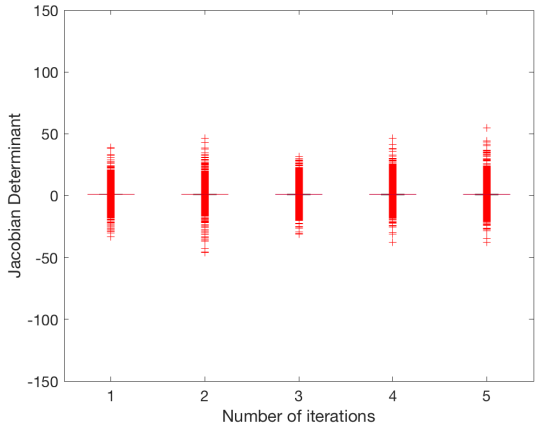


(c) Exponential composite method

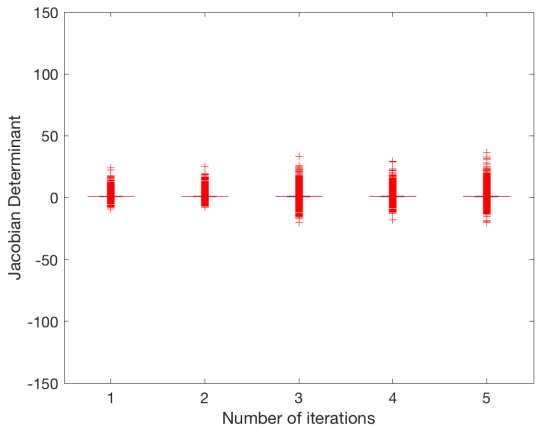
Figure 6.5: Box plot showing the Jacobian determinant with varying number of iterations



(a) Additive method



(b) Composite method



(c) Exponential composite method

Figure 6.6: Box plot showing the Jacobian determinant with varying number of iterations and threshold equal to zero

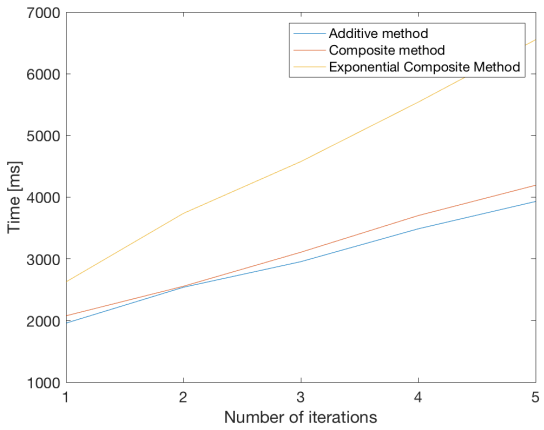


Figure 6.7: Figure showing the timing of the three accumulation methods with varying number of iterations

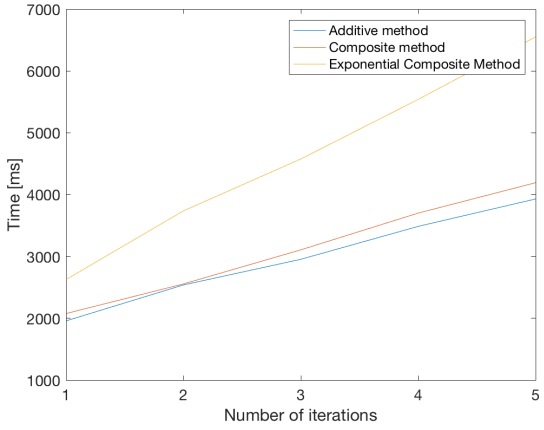


Figure 6.8: Figure showing the timing of the three accumulation methods with varying number of iterations and threshold equal to zero

6.3 Maximum theoretical deformation

A theoretical diffeomorphic vector field was created to test the accuracy of the image registration code and the different accumulation methods' effect on accuracy and timing. For most of the results in this chapter the maximum deformation of this field was set to three voxels, meaning that the deformations in the vector field were randomly distributed up to three voxels. In this section, the maximum theoretical deformation is varied from 1 to 10 to ascertain the image registration code's robustness to large deformations.

6.3.1 Magnitude Error

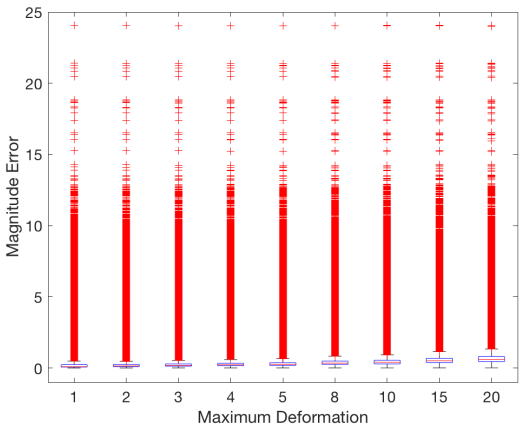
The magnitude error with different levels of maximum deformation is shown in Figure 6.9 and Table 6.6.

6.3.2 Jacobian Determinant

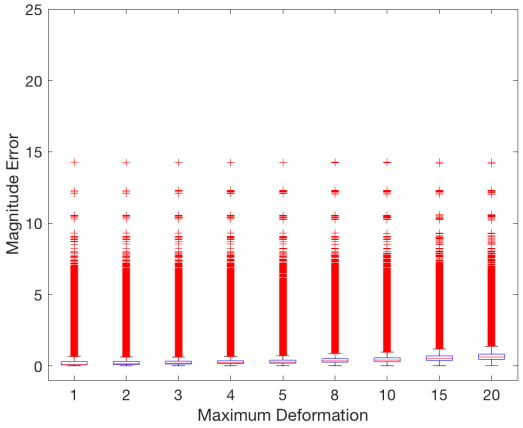
The Jacobian determinant was also found for all the estimated vector deformation fields and the results are shown in Figure 6.10. While the outliers differed, the other statistical values were the same for all the box plot with quartile 1, the median and quartile 3 equal to 1.

6.3.3 Timing

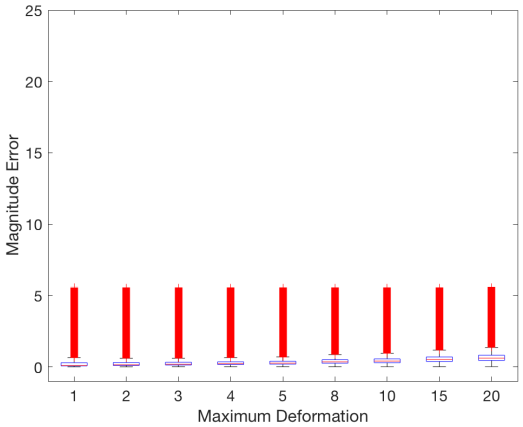
The timing was found for varying maximum deformations as well and the results can be found in Figure 6.11.



(a) Additive method



(b) Composite method

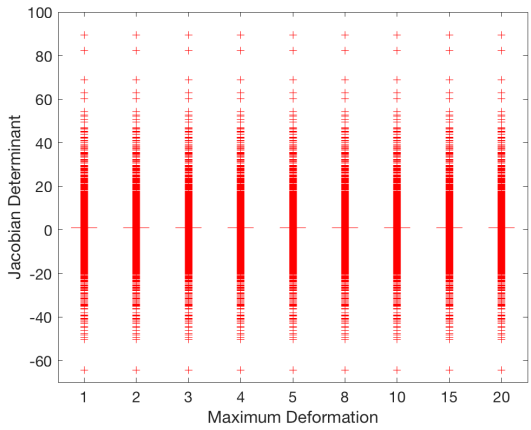


(c) Exponential composite method

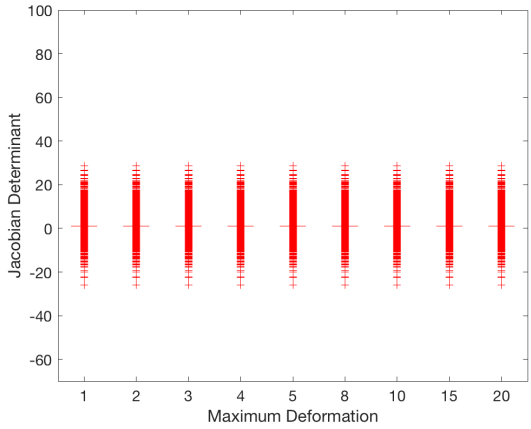
Figure 6.9: Box plot showing the magnitude of error with varying level of maximum theoretical deformation

Accumulation method	Maximum theoretical deformation	Quartile 1	Median	Quartile 3
Additive	1	0.0653	0.0914	0.2336
	2	0.1028	0.1429	0.2499
	3	0.1339	0.1853	0.2917
	4	0.1615	0.2227	0.3351
	5	0.1867	0.2568	0.3759
	8	0.2531	0.3459	0.4841
	10	0.2923	0.3982	0.5475
	15	0.3792	0.5134	0.6870
	20	0.4557	0.6142	0.8090
Composite	1	0.0661	0.0931	0.3067
	2	0.1040	0.1454	0.3125
	3	0.1354	0.1884	0.3280
	4	0.1632	0.2263	0.3607
	5	0.1887	0.2607	0.3974
	8	0.2556	0.3507	0.5014
	10	0.2950	0.4034	0.5638
	15	0.3825	0.5194	0.7022
	20	0.4593	0.6206	0.8235
Exponential Composite	1	0.0661	0.0930	0.2975
	2	0.1039	0.1452	0.3039
	3	0.1353	0.1881	0.3220
	4	0.1631	0.2260	0.3567
	5	0.1885	0.2603	0.3939
	8	0.2553	0.3501	0.4983
	10	0.2947	0.4027	0.5606
	15	0.3820	0.5184	0.6986
	20	0.4587	0.6194	0.8194

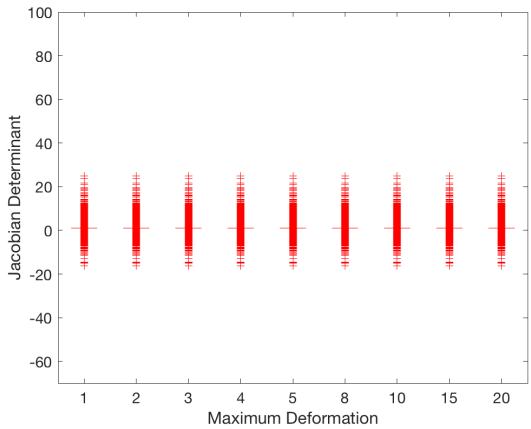
Table 6.6: Table showing the statistical values for the box plot of the magnitude error with varying level of maximum theoretical deformation



(a) Additive method



(b) Composite method



(c) Exponential composite method

Figure 6.10: Box plot showing the Jacobian determinant with varying level of maximum theoretical deformation

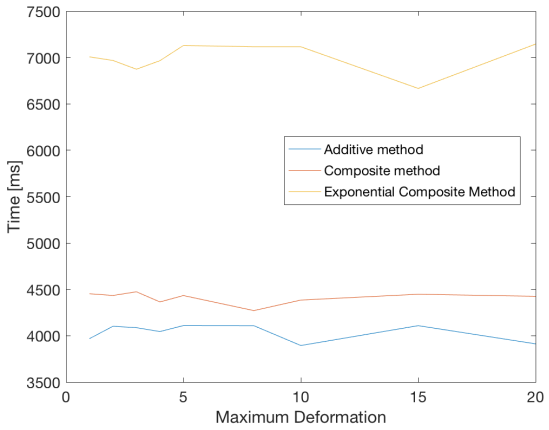


Figure 6.11: Figure showing the timing of the three accumulation methods with varying level of maximum theoretical deformation

6.4 Size of neighbourhood

In this case, what is meant by the size of the neighbourhood is the number of voxels in each direction around a voxel that are included in the image registration of that voxel. This means that if the size of the neighbourhood is equal to n, $(2n + 1)^3$ voxels are included in the image registration estimation of that voxel. This can also be viewed as a representation of the size of $w(\mathbf{x})$ in equations 5.28, 5.29 and 5.30.

6.4.1 Magnitude Error

The effect of the size of the neighbourhood on the magnitude error was found for the threshold equal to 0.5 and is shown in Figure 6.12 and Table 6.7.

Accumulation method	Size of neighbourhood (n)	Quartile 1	Median	Quartile 3
Additive	0	0.1302	0.1776	0.2464
	1	0.1339	0.1853	0.2917
	2	0.1364	0.1908	0.3962
Composite	0	0.1312	0.1795	0.2530
	1	0.1354	0.1884	0.3280
	2	0.1382	0.1948	0.4848
Exponential Composite	0	0.1311	0.1792	0.2518
	1	0.1353	0.1881	0.3220
	2	0.1381	0.1945	0.4730

Table 6.7: Table showing the statistical values for the box plot of the magnitude error with varying size of neighbourhood

The magnitude error was found with the threshold equal to zero as well, but only for n equal to 0 and 1 because the code crashed with a higher n. The results can be seen in Figure 6.13 and Table 6.8.

6.4.2 Jacobian Determinant

The effect of a larger neighbourhood on the diffeomorphism of the estimated image was found by looking at the Jacobian determinant and the results for the threshold equal to 0.5 can be seen in Figure 6.14. All the box plots in Figure 6.14 had a first quartile, median and third quartile equal to 1.

The Jacobian determinant was found for varying sizes of neighbourhoods with the threshold equal to zero as well and the results can be seen in Figure 6.15 and Table 6.9.

Accumulation method	Size of neighbourhood (n)	Quartile 1	Median	Quartile 3
Additive	0	0.1592	0.3241	1.5565
	1	0.1586	0.2940	1.2796
Composite	0	0.1664	0.5842	1.6040
	1	0.1660	0.4762	1.3971
Exponential Composite	0	0.1664	0.5842	1.6040
	1	0.1660	0.4762	1.3971

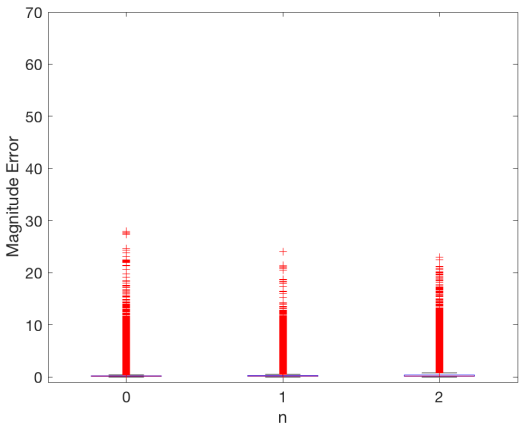
Table 6.8: Table showing the statistical values for the box plot of the magnitude error with varying size of neighbourhood and threshold equal to zero

Accumulation method	Size of neighbourhood (n)	Quartile 1	Median	Quartile 3
Additive	0	0.8510	1.0000	1.0019
	1	0.9210	1.0000	1.0037
Composite	0	0.7806	1.0000	1.0098
	1	0.8640	1.0000	1.0105
Exonential Composite	0	0.8500	1.0000	1.0060
	1	0.8807	1.0000	1.0109

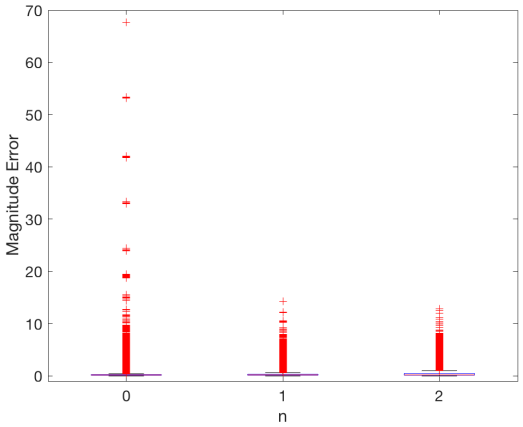
Table 6.9: Table showing the statistical values for the box plot of the Jacobian determinant with varying size of neighbourhood and threshold equal to zero

6.4.3 Timing

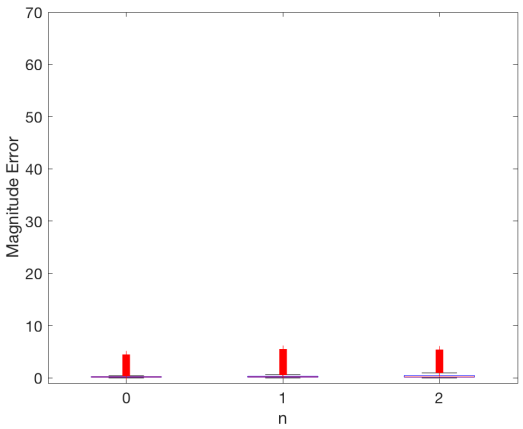
The effect of a varying size of neighbourhood on timing was found and the results are shown in Figure 6.16 with the threshold equal to 0.5 and in Figure 6.17 with threshold equal to zero.



(a) Additive method

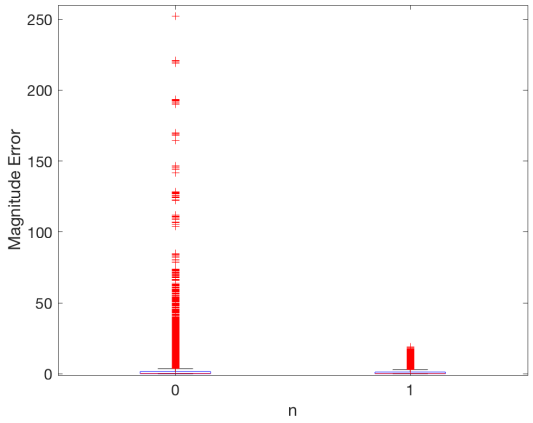


(b) Composite method

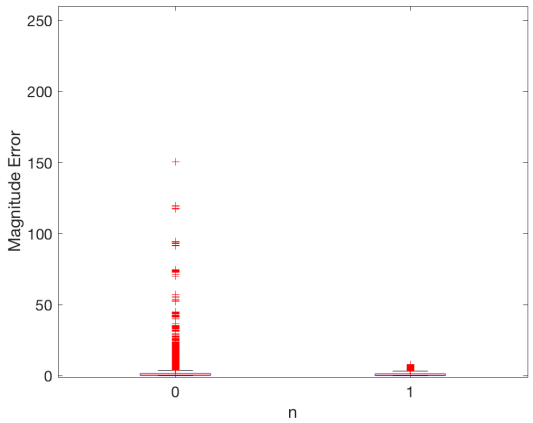


(c) Exponential composite method

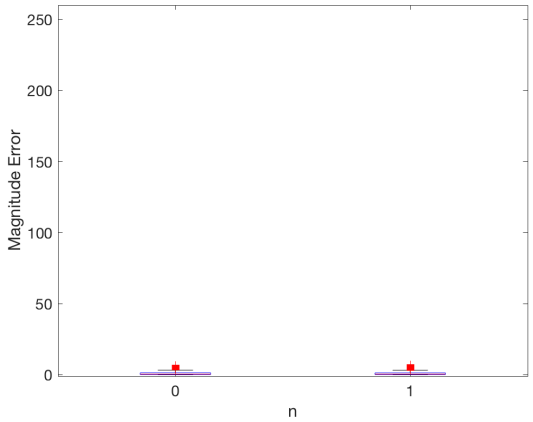
Figure 6.12: Box plot showing the magnitude of error with varying size of neighbourhood



(a) Additive method

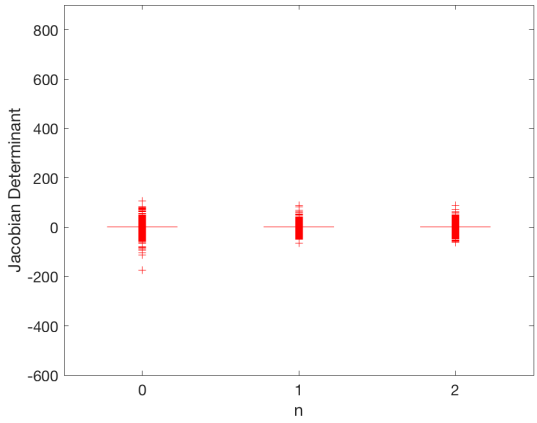


(b) Composite method

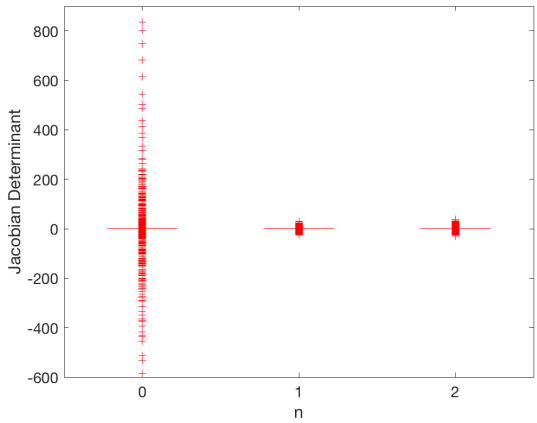


(c) Exponential composite method

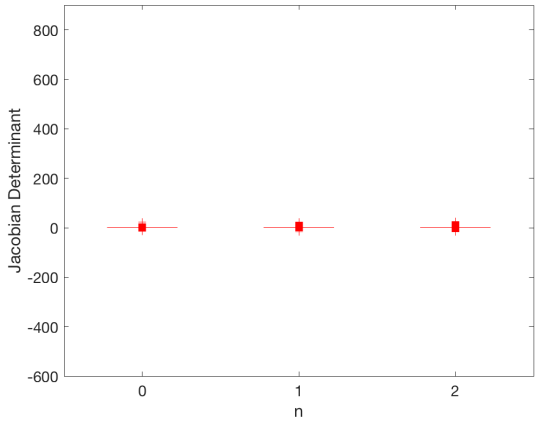
Figure 6.13: Box plot showing the magnitude of error with varying size of neighbourhood and threshold equal to zero



(a) Additive method

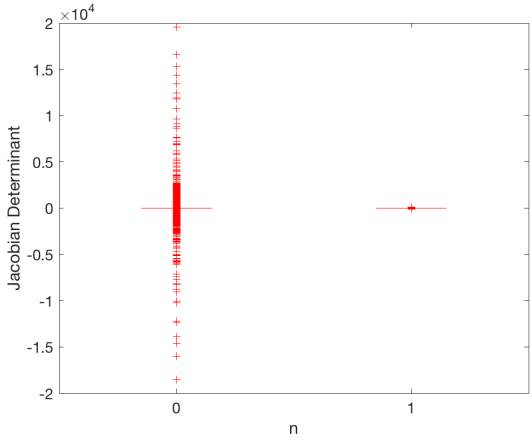


(b) Composite method

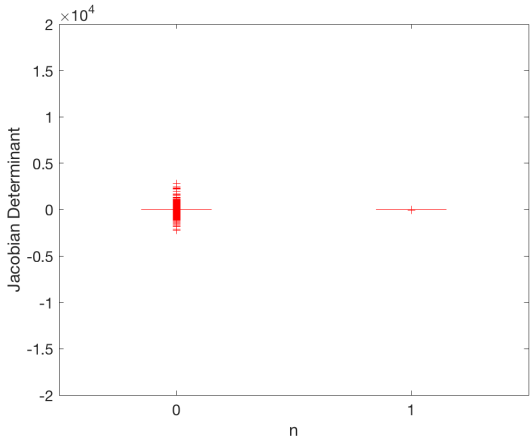


(c) Exponential composite method

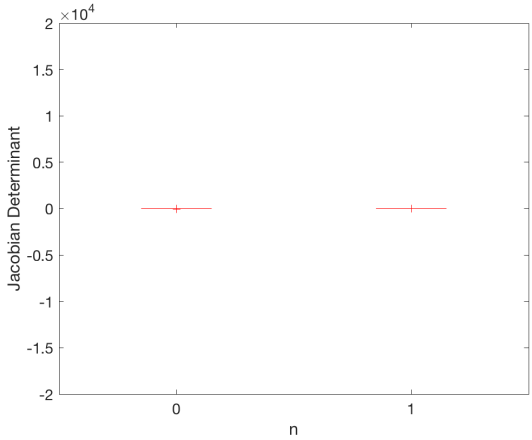
Figure 6.14: Box plot showing the Jacobian determinant with varying size of neighbourhood



(a) Additive method



(b) Composite method



(c) Exponential composite method

Figure 6.15: Box plot showing the Jacobian determinant with varying size of neighbourhood and threshold equal to zero

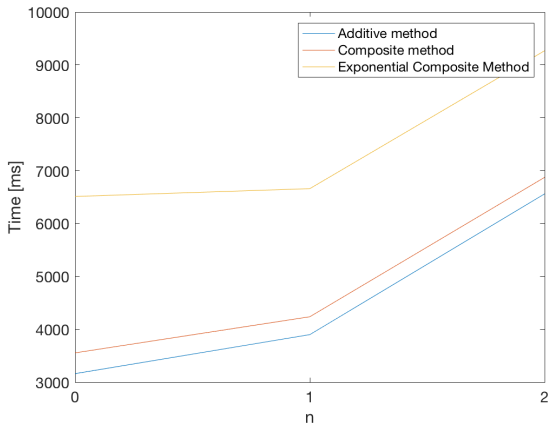


Figure 6.16: Figure showing the timing of the three accumulation methods with varying size of neighbourhood

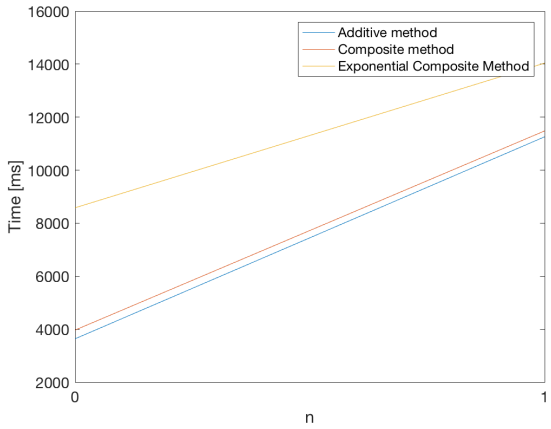


Figure 6.17: Figure showing the timing of the three accumulation methods with varying size of neighbourhood and threshold equal to zero

6.5 Threshold

Varying the threshold can limit the number of voxels whose deformations are estimated. Because this was likely to affect both accuracy and timing, results were gathered to determine how pronounced this effect would be. The threshold values range from 0 to 255, but have been normalised to 1. The thresholding is set up in such a way that the threshold value is included in the image registration, therefore, when the threshold is set to 1 only the values equal to 255 are included in the image registration.

6.5.1 Magnitude Error

The magnitude error for varying the threshold is shown in Figure 6.18 and Table 6.10.

Accumulation method	Threshold	Quartile 1	Median	Quartile 3
Additive	0	0.1586	0.2940	1.2796
	0.2	0.1530	0.2464	1.2166
	0.4	0.1396	0.1981	0.5561
	0.6	0.1298	0.1769	0.2450
	0.8	0.1231	0.1648	0.2123
	1	0.1192	0.1582	0.1992
Composite	0	0.1660	0.4762	1.3971
	0.2	0.1580	0.2809	1.2972
	0.4	0.1417	0.2033	0.6422
	0.6	0.1307	0.1786	0.2512
	0.8	0.1236	0.1657	0.2142
	1	0.1192	0.1583	0.1993
Exponential Composite	0	0.1658	0.4677	1.3497
	0.2	0.1580	0.2799	1.2573
	0.4	0.1416	0.2030	0.6270
	0.6	0.1306	0.1784	0.2502
	0.8	0.1236	0.1656	0.2140
	1	0.1192	0.1583	0.1993

Table 6.10: Table showing the statistical values for the box plot of the magnitude error with varying threshold

6.5.2 Jacobian Determinant

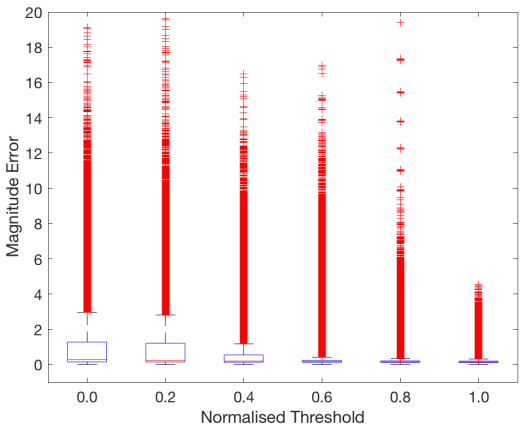
The effect of varying the threshold on the diffeomorphism of the estimated vector deformation field is shown in Figure 6.19 and Table 6.11.

Accumulation method	Threshold	Quartile 1	Median	Quartile 3
Additive	0	0.9210	1.0000	1.0037
	0.2	0.9665	1.0000	1.0001
	> 0.4	1.0000	1.0000	1.0000
Composite	0	0.8640	1.0000	1.0105
	0.2	0.9235	1.0000	1.0004
	> 0.4	1.0000	1.0000	1.0000
Exponential Composite	0	0.8807	1.0000	1.0109
	0.2	0.9318	1.0000	1.0005
	> 0.4	1.0000	1.0000	1.0000

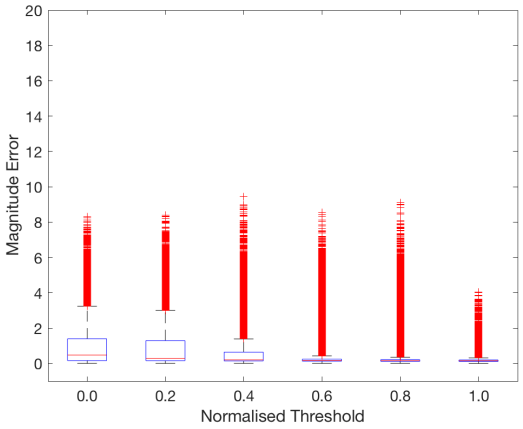
Table 6.11: Table showing the statistical values for the box plot of the Jacobian determinant with varying threshold

6.5.3 Timing

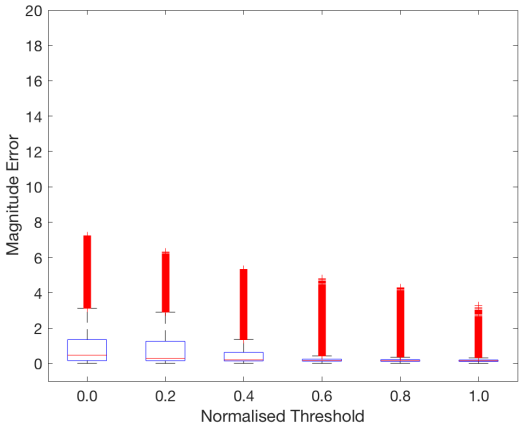
The timing was also likely to be affected by the threshold level, so its effects were found and the results can be seen in Figure 6.20.



(a) Additive method

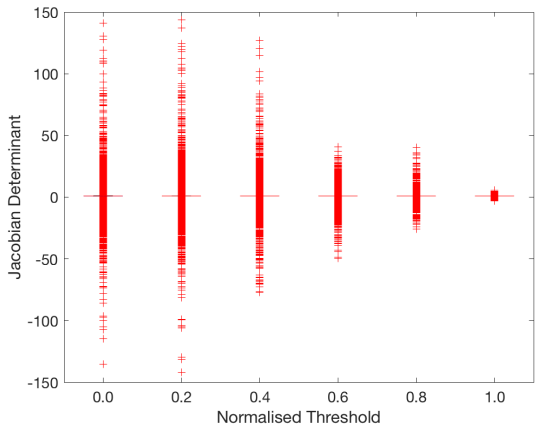


(b) Composite method

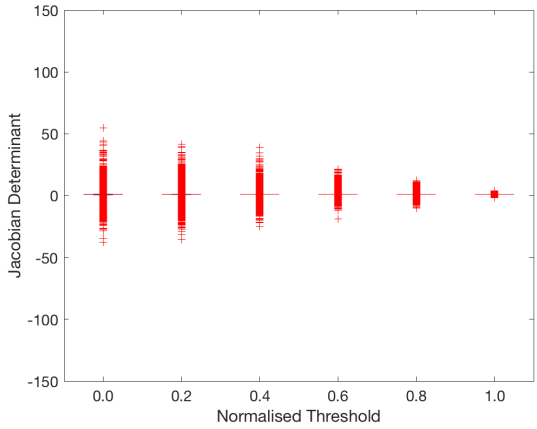


(c) Exponential composite method

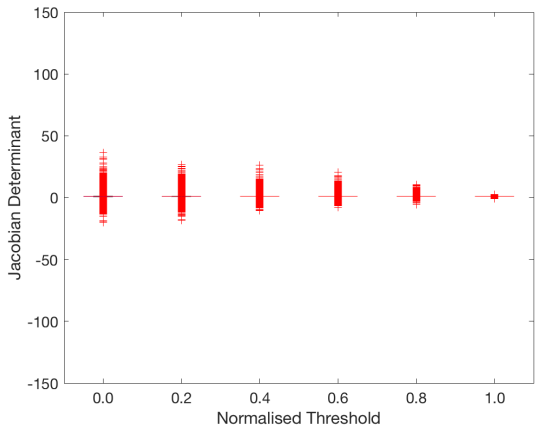
Figure 6.18: Box plot showing the magnitude of error with varying threshold



(a) Additive method



(b) Composite method



(c) Exponential composite method

Figure 6.19: Box plot showing the Jacobian determinant with varying threshold

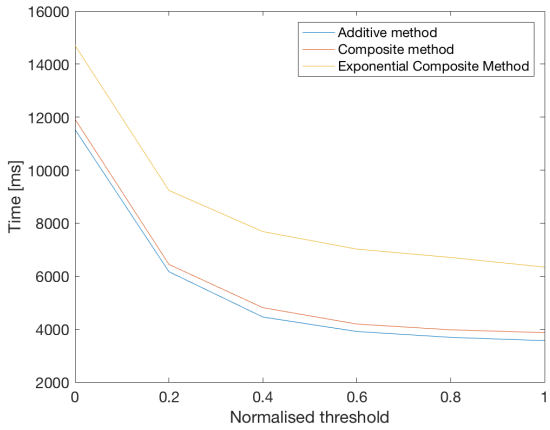


Figure 6.20: Figure showing the timing of the three accumulation methods with varying threshold

6.6 Accumulation Methods

To set up an iterative scheme, three different accumulation methods were tested; additive, composite and exponential composite. After testing the effects of many different parameters on the timing and accuracy, a more visual representation of the results for the three accumulation methods was generated. These representations were shown on both the short axis and one of the long axes. The short axis slice of the reference image can be seen in Figure 6.21 and the long axis slice of the reference image is in Figure 6.22.

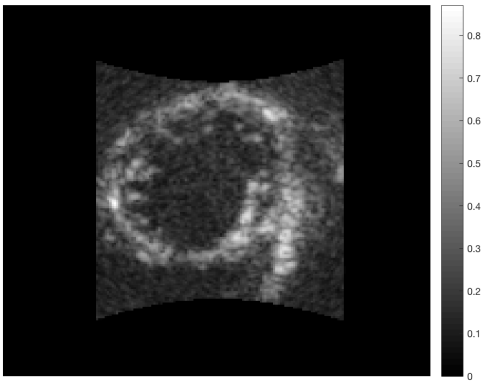


Figure 6.21: Short axis slice of 3D echocardiographic image

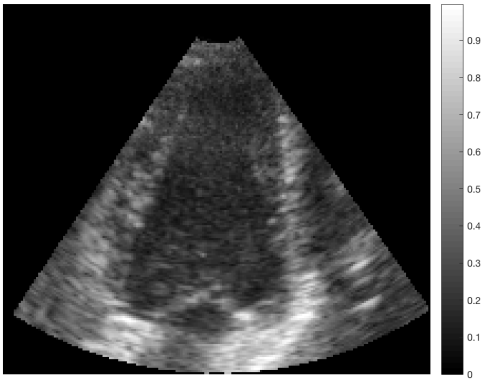


Figure 6.22: Long axis slice of 3D echocardiographic image

6.6.1 Magnitude Error

In Figures 6.23 and 6.24 the theoretical generated vector field is shown alongside the magnitude error for each pixel along that slice for each accumulation method.

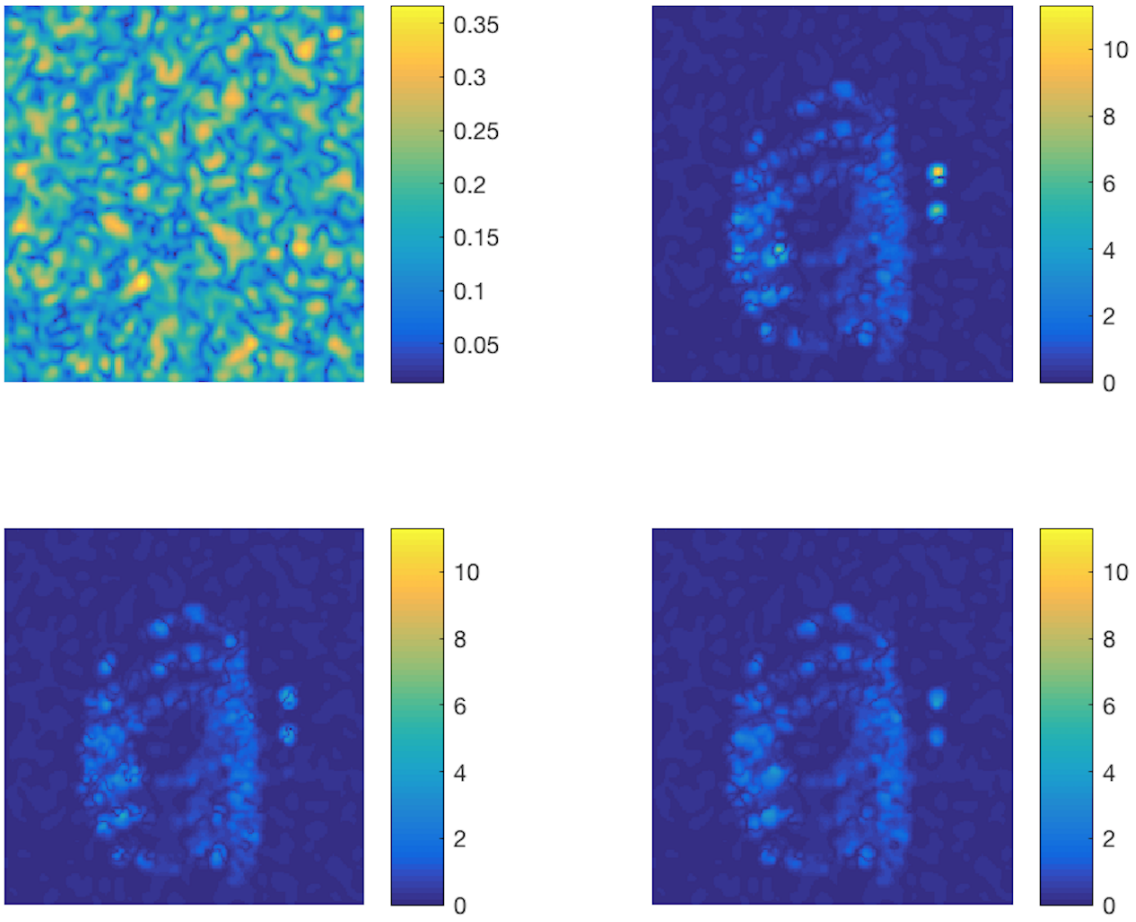


Figure 6.23: Short axis slice magnitude error of each voxel. Top left: deformation field, top right: magnitude error of additive method, bottom left: magnitude error of composite method, bottom right: magnitude error exponential composite method

6.6.2 Jacobian Determinant

Figures 6.25 and 6.26 show the Jacobian determinant of each pixel along the slice for each accumulation method. The colorbars on the sides of Figures 6.25 and 6.26 are normalised to the absolute maximum Jacobian determinant that was found for all three accumulation methods in that particular slice. The negative Jacobian

determinant values are represented by red colour tones while the positive Jacobian determinant values are represented by green colour tones.

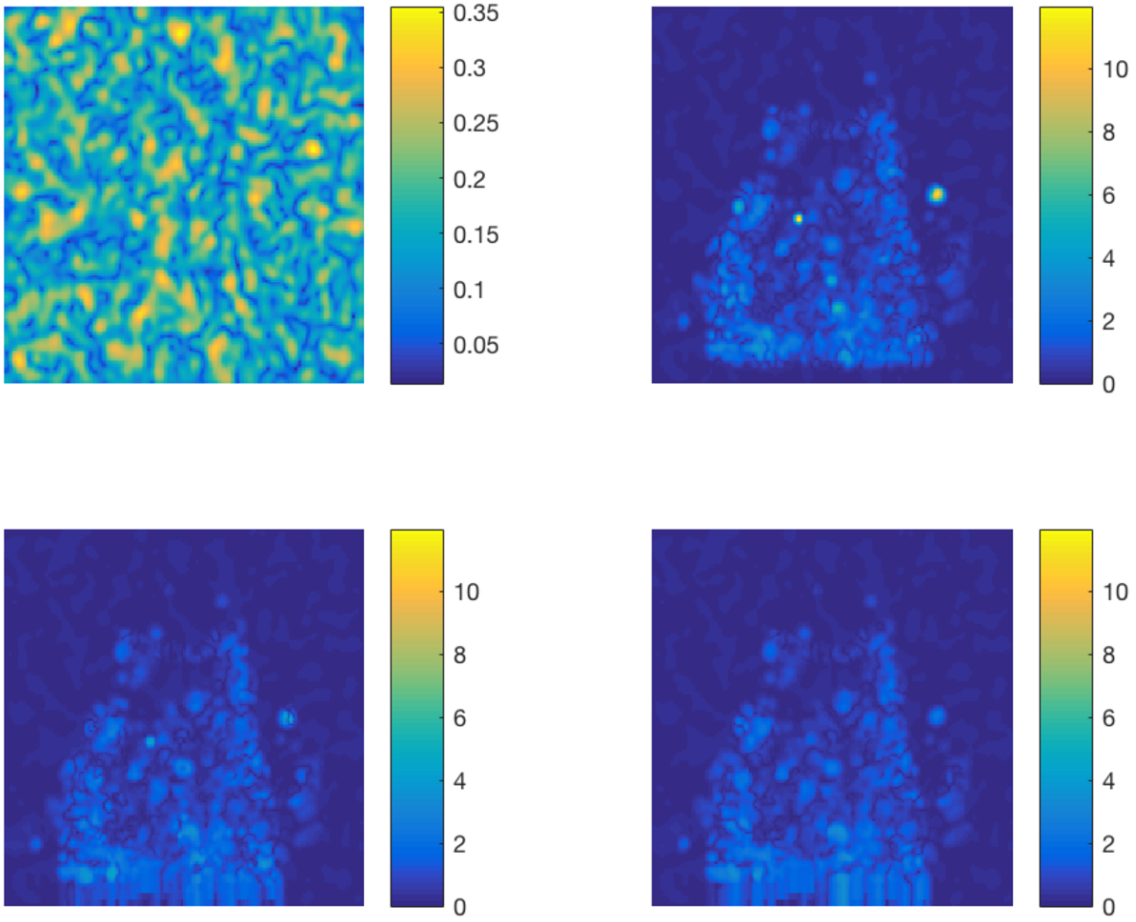
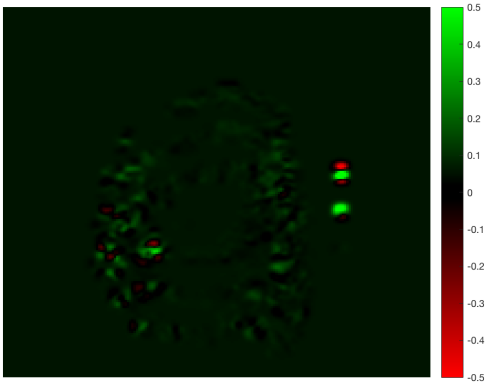
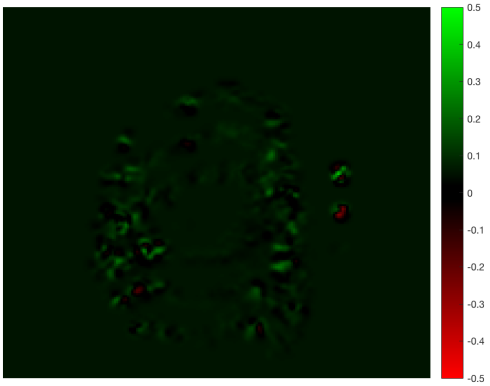


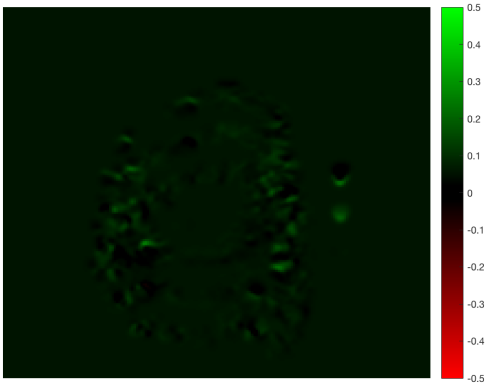
Figure 6.24: Long axis slice magnitude error of each voxel. Top left: deformation field, top right: magnitude error of additive method, bottom left: magnitude error of composite method, bottom right: magnitude error exponential composite method



(a) Additive method

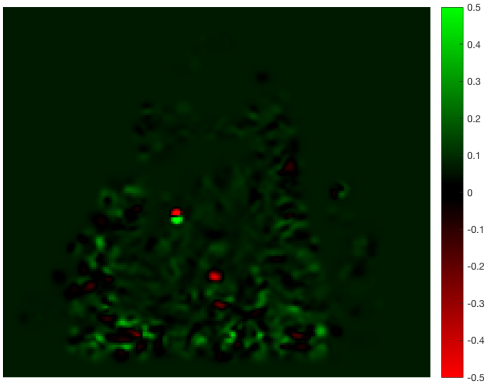


(b) Composite method

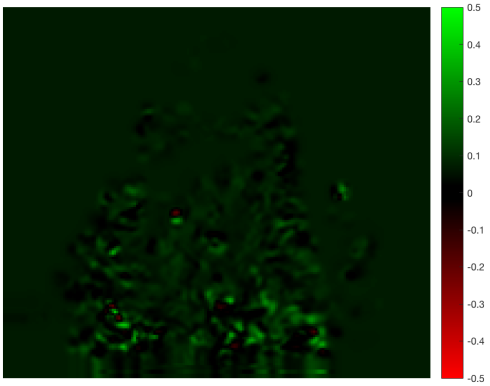


(c) Exponential composite method

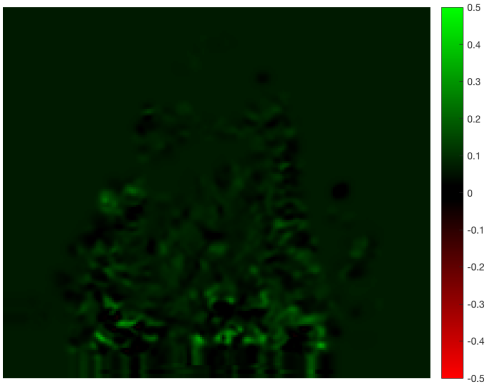
Figure 6.25: Short axis slice for the Jacobian determinant of each voxel



(a) Additive method



(b) Composite method



(c) Exponential composite method

Figure 6.26: Long axis slice for the Jacobian determinant of each voxel

6.7 Real cases

To check if the code would work in a real scenario, two different 3D echocardiographic images were tested. Both image sets were tested between two pairs of frames to check whether the resulting vector field was diffeomorphic or not. In one pair, the volume difference between the frames was the minimum volume for that 3D echocardiographic image set and in the other it was the maximum volume for that image set. In all the Figures where the Jacobian determinant is represented by images with red and green colours, the red represents a negative Jacobian determinant and the green represents a positive Jacobian determinant.

6.7.1 Image set 1

The first 3D echocardiographic image set had dimensions $(x, y, z) = (192, 171, 193)$, minimum volume difference between frames equal to 2.77207ml and maximum volume difference equal to 70.34682ml.

Minimum volume difference

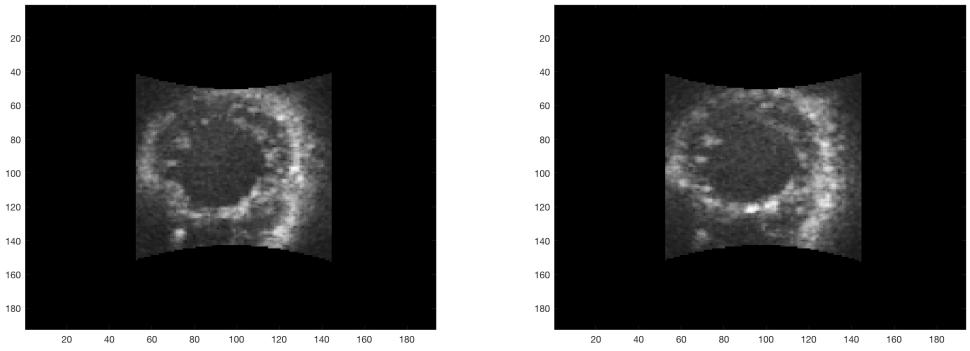
The minimum volume difference for the first image was 2.77207ml and was between two frames in diastole. The moving and reference images along a slice of the short axis are shown in Figure 6.27. Figure 6.28 shows the Jacobian determinant for all the pixels along the same slice on the short axis. Figure 6.29 shows the moving and reference images of one long axis for the small volume difference while Figure 6.30 shows the Jacobian determinant along one slice on a long axis. Figure 6.31 shows a box plot of all the Jacobian determinants of the entire 3D image.

Maximum volume difference

For the maximum volume difference of 70.34682ml of the first image one of the frames was in systole while the other was in diastole. Figure 6.32 shows the short axis of the reference and moving image while Figure 6.33 shows the same slice on the short axis of the Jacobian determinant for each pixel of the three different accumulation methods. The same is shown for the long axis in Figures 6.34 and 6.33. A box plot of the Jacobian determinants over the entire 3D image for all three accumulation methods can be seen in Figure 6.36.

6.7.2 Image set 2

The second image set had dimensions $(x, y, z) = (211, 171, 214)$ with the minimum volume difference between frames equal to 1.85355ml and the maximum volume difference equal to 111.80095ml.



(a) Reference image (b) Moving image

Figure 6.27: Short axis slice for the reference and moving image with a volume difference of 2.77207ml

Minimum volume difference

For the second image, the minimum volume difference was 1.85355ml and the moving and reference images were taken during systole. The short axis slice of the reference and moving image is shown in Figure 6.37 while the Jacobian determinant of each pixel along this slice for all three accumulation methods is in Figure 6.38. The same images for the long axis are shown in Figures 6.39 and 6.40. A box plot of the Jacobian determinant for all the voxels in the image is in Figure 6.41.

Maximum volume difference

For the final real case the second image set was used as well with a large volume difference of 111.80895ml between the reference and moving image taken during systole and diastole. For the short axis, the slices of the reference and moving image can be seen in Figure 6.42 and the Jacobian determinants of the separate pixels in the same slice can be seen in Figure 6.43. The same information about the long axis slice can be seen in Figures 6.44 and 6.45. The box plots of the Jacobian determinant for all voxels in the entire 3D image can be seen in Figure 6.46.

6.7.3 Timing

A comparison of the computational expense for the different accumulation methods and the two 3D echocardiographic image sets can be seen in Table 6.12.

The timing of a desktop computer on the same image sets was computed to show the effect of a more powerful graphics card on the computational expense. Table

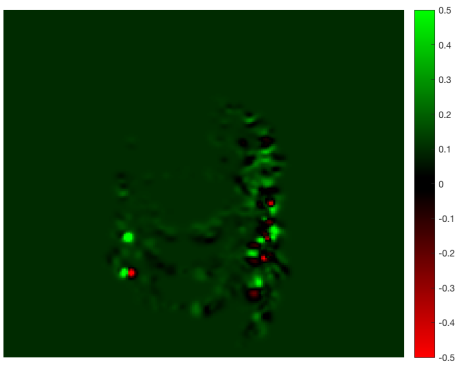
Volume difference (ml)	Image dimensions (x, y, z)	Additive accumulation method (ms)	Composite accumulation method (ms)	Exponential composite accumulation method (ms)
2.77207	(192, 171, 193)	3435.5	3691.1	7386.1
70.34682	(192, 171, 193)	3349.8	3700.4	7605.2
1.85355	(211, 171, 214)	4240.3	4780.9	10262
111.80895	(211, 171, 214)	4348.3	4833.6	10687

Table 6.12: Table showing the timing of the four real cases for the three accumulation methods

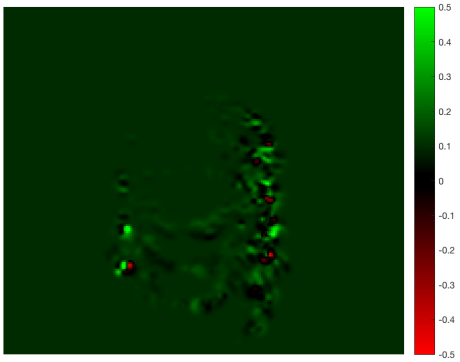
6.13 shows the timing for the two image sets on a desktop computer with a GeForce GTX 690 graphics card that could handle a maximum work-group size of 1024 and had 8 parallel compute units. The desktop used the same version of OpenCL as the laptop; OpenCL 1.2.

Image dimensions (x, y, z)	Additive accumulation method (ms)	Composite accumulation method (ms)	Exponential composite accumulation method (ms)
(192, 171, 193)	744.9832	812.6449	1529.0
(211, 171, 214)	834.8487	924.9172	1791.6

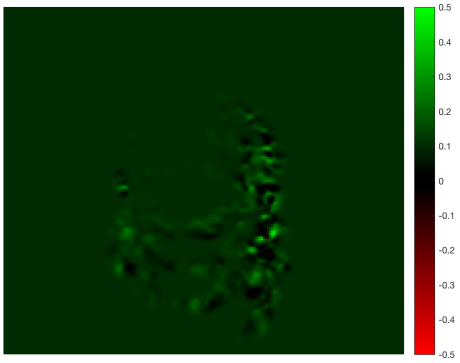
Table 6.13: Table showing the timing of the two image sets for the three accumulation methods with a desktop computer



(a) Additive method

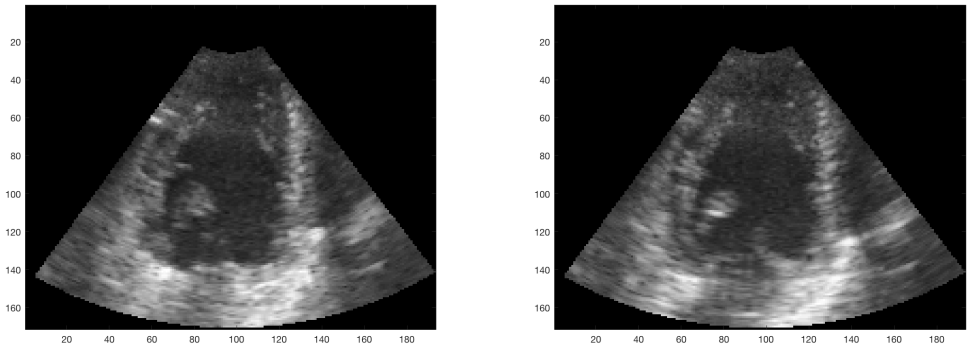


(b) Composite method



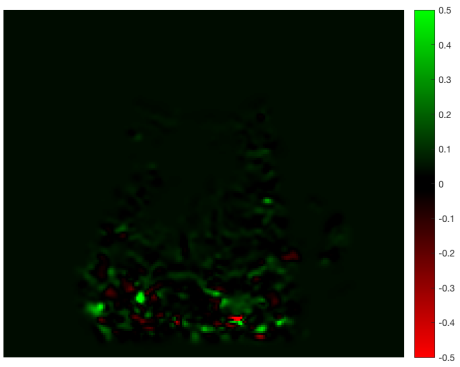
(c) Exponential composite method

Figure 6.28: Short axis slice for the Jacobian determinant of each voxel with a volume difference of 2.77207ml between the reference and moving image

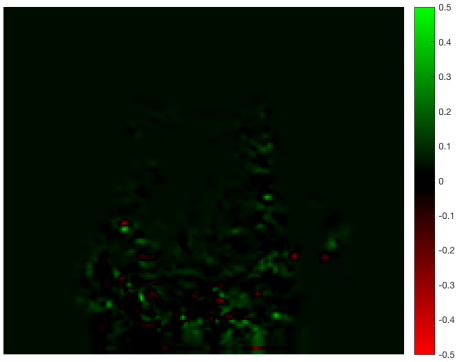


(a) Reference image (b) Moving image

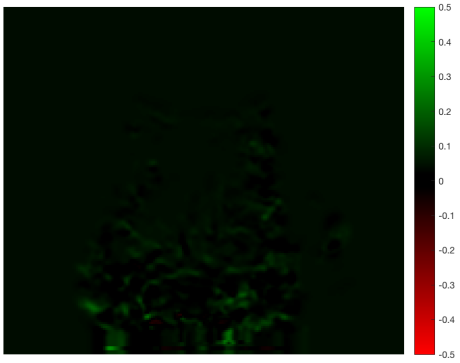
Figure 6.29: Long axis slice for the reference and moving image with a volume difference of 2.77207 ml



(a) Additive method



(b) Composite method



(c) Exponential composite method

Figure 6.30: Long axis slice for the Jacobian determinant of each voxel with a volume difference of 2.77207ml between the reference and moving image

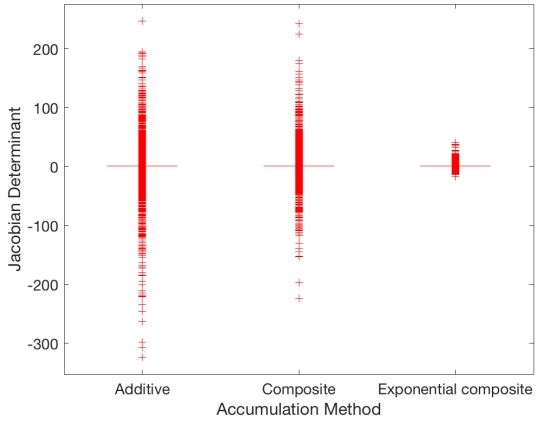
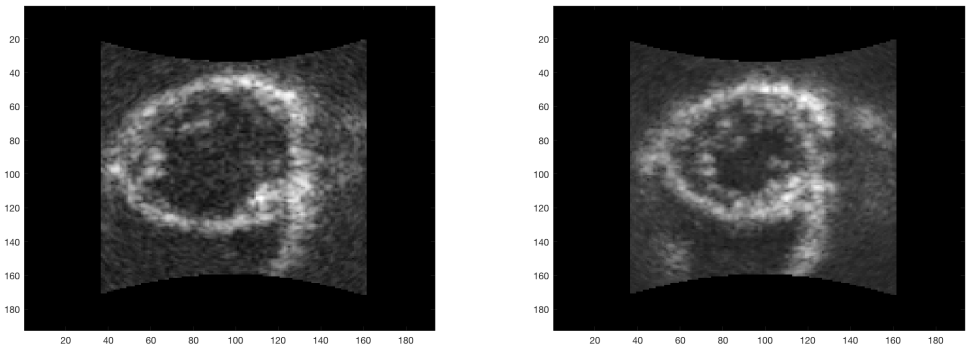


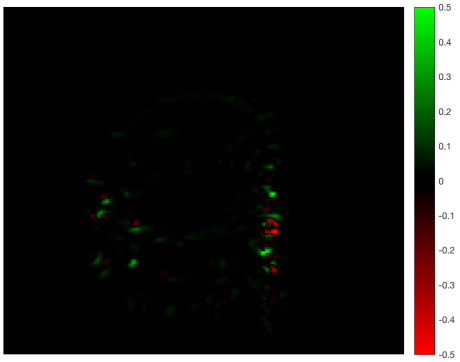
Figure 6.31: Box plot of Jacobian determinant of the images with a volume difference of 2.77207ml between the reference and moving image



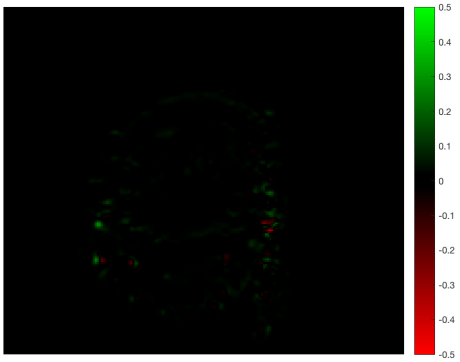
(a) Reference image

(b) Moving image

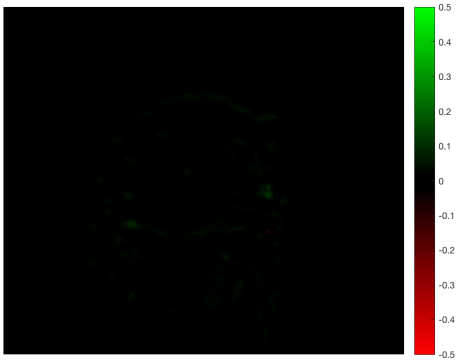
Figure 6.32: Short axis slice for the reference and moving image with a volume difference of 70.34682ml



(a) Additive method

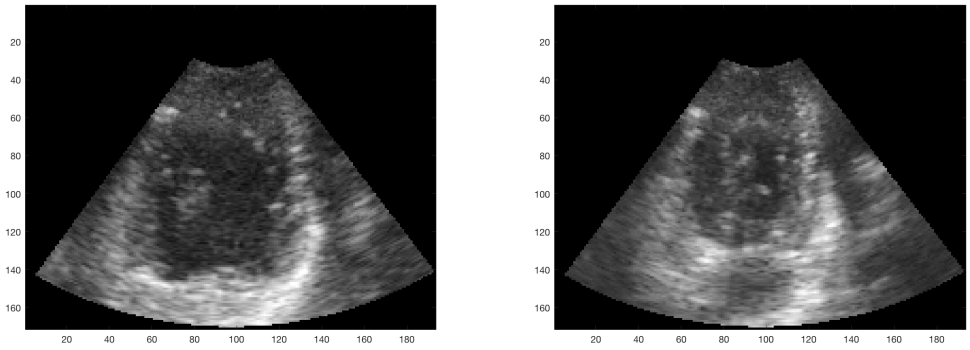


(b) Composite method



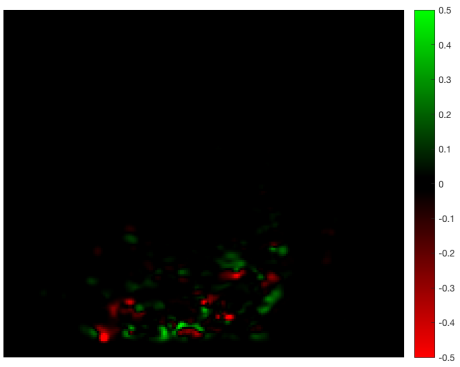
(c) Exponential composite method

Figure 6.33: Short axis slice for the Jacobian determinant of each voxel with a volume difference of 70.34682ml between the reference and moving image

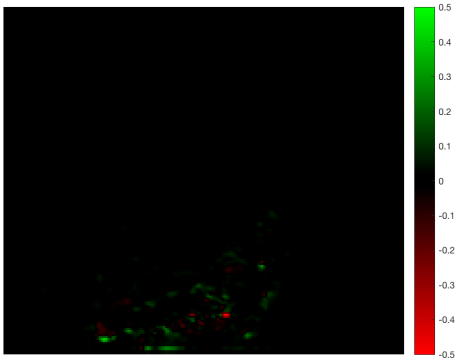


(a) Reference image (b) Moving image

Figure 6.34: Long axis slice for the reference and moving image with a volume difference of 70.34682ml



(a) Additive method



(b) Composite method

(c) Exponential composite method

Figure 6.35: Long axis slice for the Jacobian determinant of each voxel with a volume difference of 70.34682ml between the reference and moving image

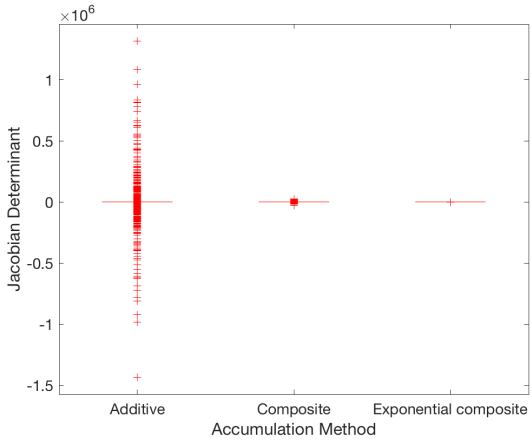
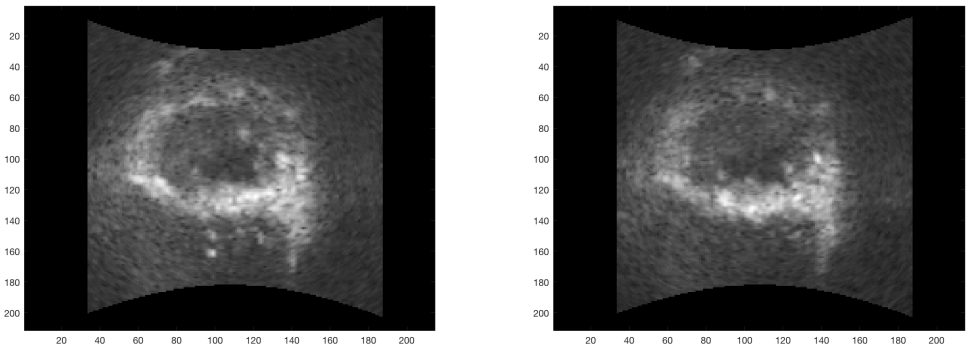


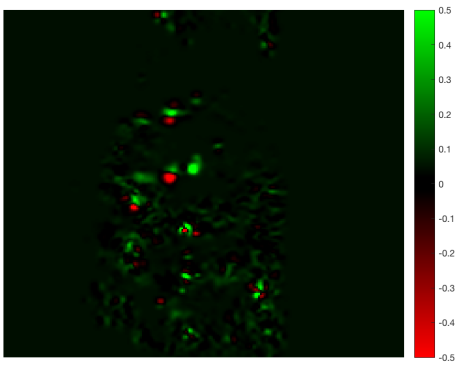
Figure 6.36: Box plot of Jacobian determinant of the images with a volume difference of 70.34682ml between the reference and moving image



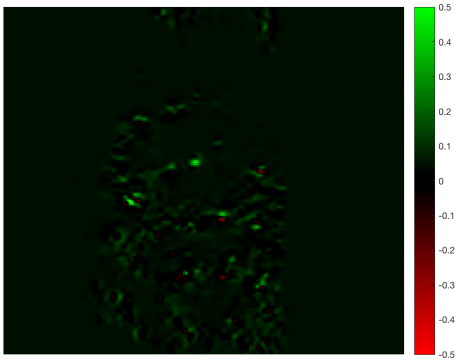
(a) Reference image

(b) Moving image

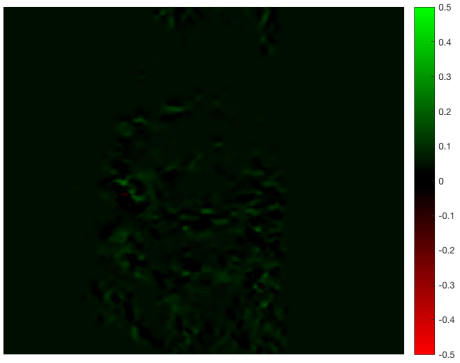
Figure 6.37: Short axis slice for the reference and moving image with a volume difference of 1.85355ml



(a) Additive method

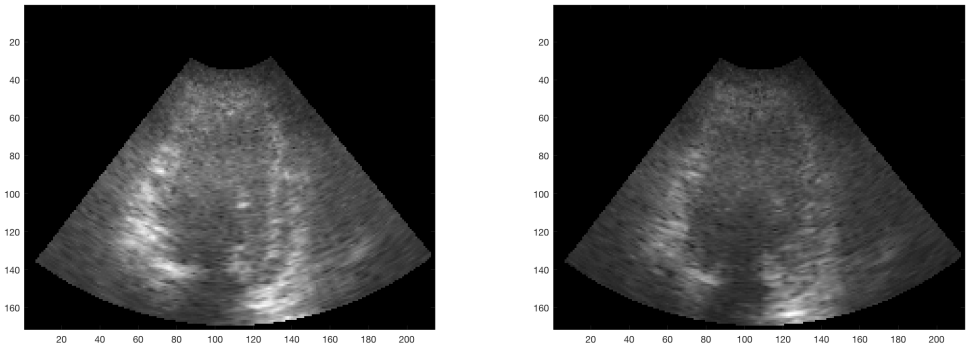


(b) Composite method



(c) Exponential composite method

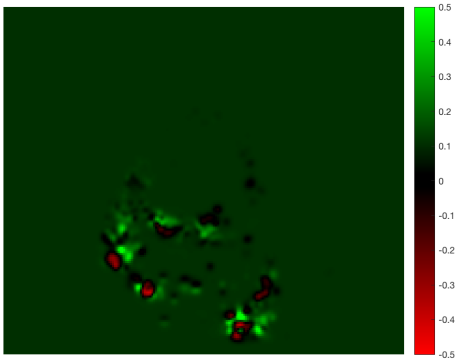
Figure 6.38: Short axis slice for the Jacobian determinant of each voxel with a volume difference of 1.85355ml between the reference and moving image



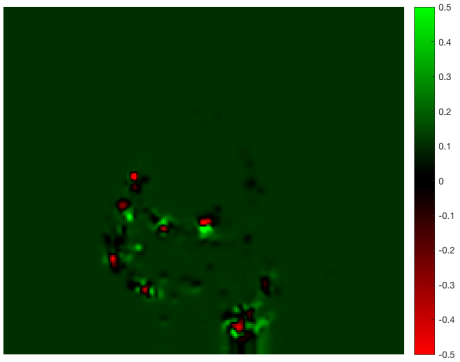
(a) Reference image

(b) Moving image

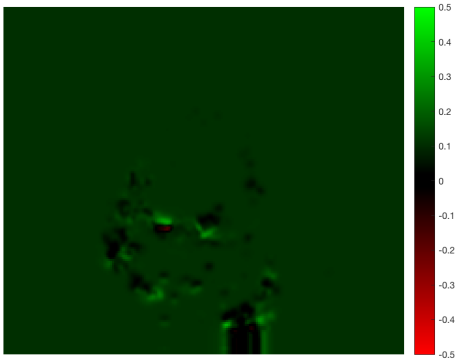
Figure 6.39: Long axis slice for the reference and moving image with a volume difference of 1.85355ml



(a) Additive method



(b) Composite method



(c) Exponential composite method

Figure 6.40: Long axis slice for the Jacobian determinant of each voxel with a volume difference of 1.85355ml between the reference and moving image

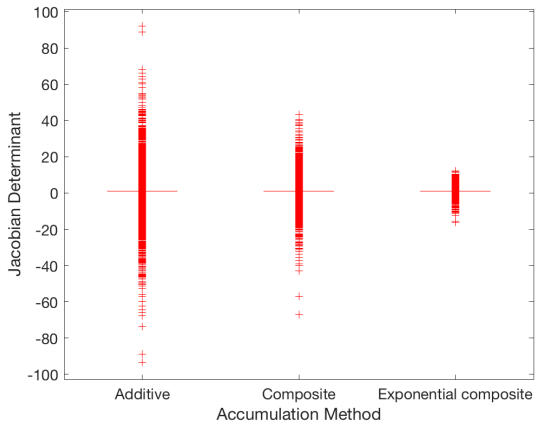
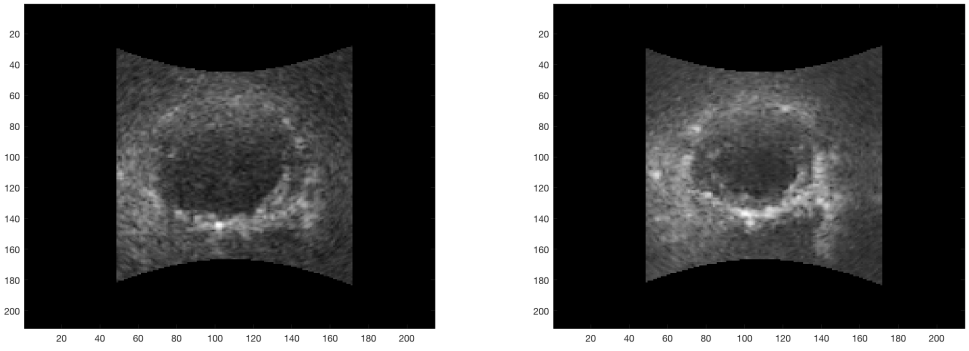


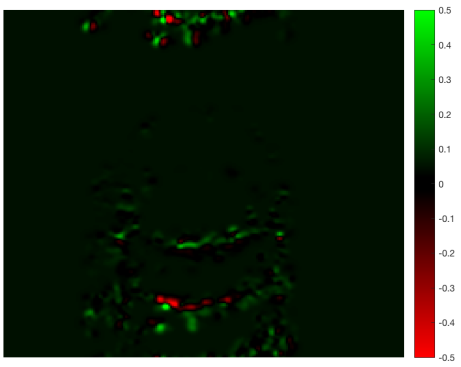
Figure 6.41: Box plot of Jacobian determinant of the images with a volume difference of 1.85355ml between the reference and moving image



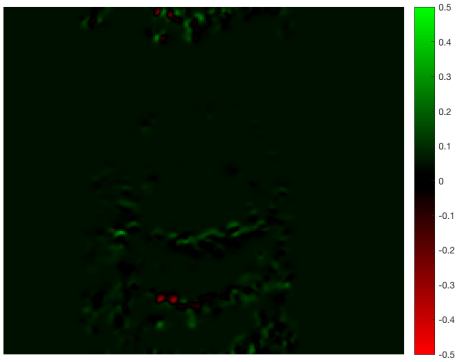
(a) Reference image

(b) Moving image

Figure 6.42: Short axis slice for the reference and moving image with a volume difference of 111.80895ml



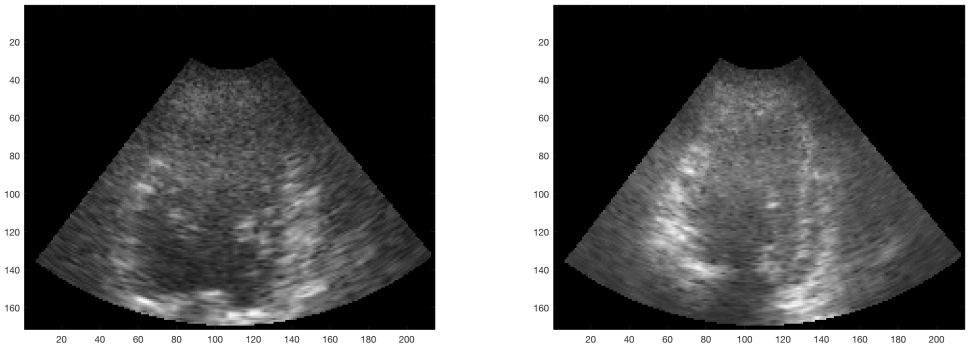
(a) Additive method



(b) Composite method

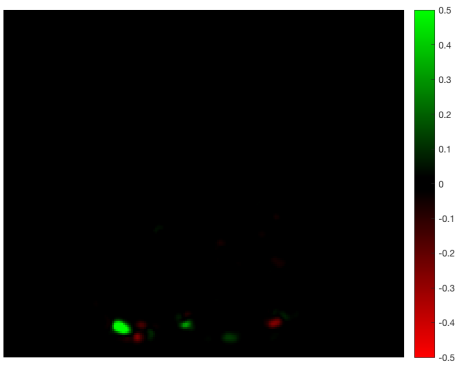
(c) Exponential composite method

Figure 6.43: Short axis slice for the Jacobian determinant of each voxel with a volume difference of 111.80895ml between the reference and moving image

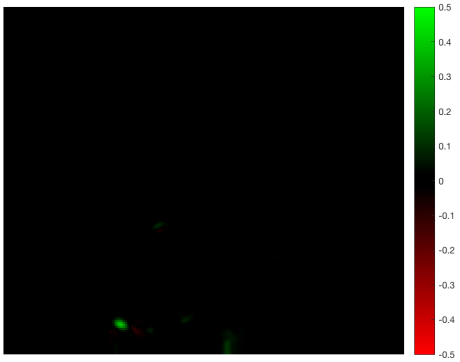


(a) Reference image (b) Moving image

Figure 6.44: Long axis slice for the reference and moving image with a volume difference of 111.80895ml



(a) Additive method



(b) Composite method

(c) Exponential composite method

Figure 6.45: Long axis slice for the Jacobian determinant of each voxel with a volume difference of 111.80895ml between the reference and moving image

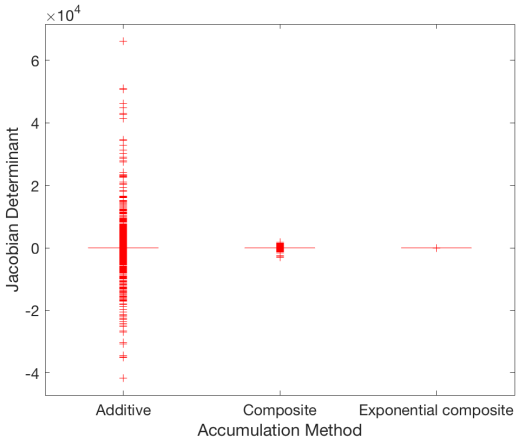


Figure 6.46: Box plot of Jacobian determinant of the images with a volume difference of 111.80895ml between the reference and moving image

7 | Discussion

To determine the optimal set-up for low computational expense and high accuracy, multiple parameters were tested for all three accumulation methods. When generating the results, some outer limits of deformation were tested to ensure that the maximum possible deformation of the heart could be picked up. This deformation is along the long axis where the base of the heart comes up towards the apex and can be up to around $12 - 15$mm for a normal adult. The aim was to complete the registration in real time as well.

7.1 β

The β relationship was tested to see what the effect of using primarily the quadratic polynomial or the linear polynomial image decomposition was on the timing and accuracy. The outliers in Figures 6.1 and 6.2 show that the linear image decomposition causes a much larger spreading of the outliers than the quadratic image decomposition. It is also clear that using only the quadratic polynomial is preferable to using a combination of a quadratic polynomial and a linear polynomial. It is possible that the reason the linear polynomial decomposition worked so poorly was because the linear component in the image was destroyed by speckles and therefore couldn't be picked up by the linear decomposition. Table 6.2 shows that within each accumulation method, the β relationship has little to no effect on the timing. This is because no matter what the beta relationship is, the same number of calculations are performed for the image registration.

7.2 Number of iterations

The effect of the number of iterations on the accuracy and timing was tested at two different thresholds; 0 and 0.5. Figures 6.3, 6.5, 6.4 and 6.6, and Tables 6.3, 6.4 and 6.5 all show that a higher number of iterations leads to outliers that are more spread out. This is probably because with more iterations, an erroneous estimation can spread even further away from where it is supposed to be. More iterations

is, however, necessary when registering large deformations because the image registration is incapable of registering very large deformations in one iteration, but the iterative scheme allows for a previous deformation estimation to be used as a priori information that the image registration algorithm can work off of. Figures 6.7 and 6.8 show that the number of iterations greatly affect the timing of the code. More iterations require the entire deformation field to be recalculated multiple times based on new a priori values, so this definitely causes the computational expense to increase.

7.3 Maximum theoretical deformation

The maximum theoretical deformation was varied between 1 and 20 voxels, meaning 0.7 and 14mm to check the accuracy and timing. Figures 6.9 and 6.10 show very little difference in the spread of the outliers with the higher maximum deformation. However, it can be seen in Table 6.6 that the interquartile range is larger the larger the maximum deformation is. Therefore, the error is, unsurprisingly, larger the bigger the maximum deformation is, but not very much larger. It is possible that this is because the size of the Gaussian kernel used to smooth the image before the image decomposition had an $n = 9$ and this would limit the maximum deformation the algorithm was able to pick up. While the estimated deformation is more accurate for small deformations, the results are still good for large deformations. This can be very helpful when trying to achieve a real-time implementation because in an imaging situation with a high framerate, multiple frames could be skipped and the deformation estimate would still be accurate even with larger deformations. Figure 6.11 shows that the timing was almost unaffected by the maximum deformation level. As the decomposition and registration are done in the same way regardless of the size of the deformation, this result is not surprising.

7.4 Size of neighbourhood

The size, n, of the neighbourhood was tested for two different thresholds as well; 0 and 0.5. Figures 6.12, 6.14, 6.13 and 6.15, and Tables 6.7, 6.8 and 6.9 show a trend where the accuracy goes up as the size, n, of the neighbourhood goes up. This is because with a larger neighbourhood, each voxel is identified by more information and is easier to register between the reference and moving images, and a large n smooths the image more so that noise does not affect the registration as much. The fact that the program crashed due to OpenCL's inbuilt timeout functionality when it was run on large neighbourhood with threshold equal to zero makes it clear that the computational expense suffers greatly with a larger neighbourhood. This can

also be seen in the plots shown in Figures 6.16 and 6.17. This is because a larger neighbourhood includes more voxels in the image registration algorithm, and will, therefore, also increase the computational expense.

7.5 Threshold

While two different thresholds were tested with many of the parameters that were examined, a separate set of results were found where only the threshold differed and the other parameters remained constant to see its effect on timing and accuracy. It is clear in Figures 6.18 and 6.19 as well as Tables 6.10 and 6.11 that the threshold affects the accuracy greatly. The outliers are much further spread out at lower thresholds because the lower voxel values that are included in the image registration resemble noise and are difficult to register between the fixed and the moving image. While the lowest number of outliers are present at the highest threshold, there is very little information in the remaining voxels in the image registration. It is therefore recommended to use a threshold around $0.5 - 0.6$ because, as seen in Figure 6.18, the interquartile range of the magnitude error box plot is similarly low between $0.6 - 1.0$ and the lowest possible threshold will give the most amount of information for accurate image registration. The challenge with a low threshold, however, is seen in Figure 6.20, where the lower the threshold, the higher the computational expense because more voxels are included in the image registration code.

7.6 Accumulation methods

Three different accumulation methods were compared for accuracy and timing. The box plots and tables in Chapter 6 all show that the exponential method has a significantly smaller spread in outliers than the composite and additive methods for both the magnitude error and the Jacobian determinant. This is likely because the flow constraints of the exponential composite method restrict the estimate from moving too far in the wrong direction in each recursion, so it can correct itself if it moves too far in the wrong direction within one iteration. The plots of the computational expense in Chapter 6 show that the accuracy of the exponential composite method comes at the expense of the timing which, in some cases, is almost twice as long due to its recursive estimation. The difference between the composite and additive methods is not as large, with a slightly more accurate and diffeomorphic result with the composite method but a slightly larger computational expense. The observations about the magnitude error are further strengthened by Figures 6.23 and 6.24 where the larger magnitude errors represented by the lighter colours can be seen more extensively in the additive and composite accumulation methods. In Figure

6.24 the bottom of the images for both the composite and exponential composite method appear to have some dragging errors which are likely due to edge effects in the image registration algorithm where the reference image went from containing valid data to no data. This occurred in the composite and exponential composite methods because they both use linear interpolation to accumulate the images and this can cause the dragging of the edge effects. Figures 6.23 and 6.24 show that the area of the image that contained valid data resulted in more erroneous estimations while the areas with no data had almost identical errors. These errors line up well with the theoretical vector field and are likely the exact same because when all the voxels in the area have a zero value, the estimated vector displacement becomes zero as well, as no other voxel in the immediate neighbourhood is a better match for the estimate than the original voxel in the reference image. The greater errors in the area with valid data are likely due to speckle noise. Figures 6.25 and 6.26 show a clear difference between the different accumulation methods when it comes to the Jacobian determinant which was represented by red colour tones when negative and green when positive. The additive method has bright red spots while the red spots in the composite method are less clear but still present and the exponential method does not have any visible red spots. This is in line with the Jacobian determinant box plots in Chapter 6.

7.7 Real cases

Real cases were tested for accuracy with two 3D echocardiography image sets where the deformation field was estimated for the maximum and minimum heart volume differences. In the coloured images representing the Jacobian determinant in Figures 6.28, 6.30, 6.33, 6.35, 6.38, 6.40, 6.43 and 6.45, the red spots are slightly more prevalent in the additive method than the composite method and both these methods are much more red than the exponential composite method indicating that they won't estimate as diffeomorphic a vector field as the exponential composite method in a real situation. This is also clear from the box plots of the Jacobian determinants in each voxel in the entire 3D image in Figures 6.31, 6.36, 6.41 and 6.46, where the exponential composite method has much less spread out outliers than both the additive and composite methods. Even in the cases where the Jacobian determinant was observed for the largest volume differences of 70.34684ml in Figures 6.32 and 6.34 or 111.80895ml in Figures 6.42 and 6.44, it was mostly positive for the exponential composite method and less positive for the additive and composite methods. From Table 6.12 it can be seen that the size of the image affects the speed of the computation greatly and that the exponential composite method, while accurate, is very computationally expensive compared to the other two. As the largest volume differences led to a mostly diffeomorphic vector field, it is reasonable to believe that

some 3D echocardiographic frames could be skipped while maintaining accuracy, to achieve a real-time implementation. The timing can also be improved by using a more powerful graphics card, which is seen when comparing Tables 6.12 and 6.13 where it is clear that the timing of the laptop is up to five times slower than the timing on the desktop computer.

7.8 Further work

While the code implemented and tested in this thesis is a step in the right direction, many things can be improved. The first is implementing a pyramid scheme such as the one used in [1]. In [1] it was shown that a pyramid scheme could both reduce the computational expense and increase the accuracy of the results as well as making the image registration more robust against large deformations. While the image registration in [1] was rigid, the timing and accuracy trends found there are likely to occur in non-rigid image registration as well. The challenge with a pyramid scheme is that with non-rigid image registration, it is difficult to test the accuracy of the pyramid scheme at each pyramid level and not just at level 0 (the original image). Another image registration method that was implemented in [1] was an affine transformation. In [1], the affine transformation was more accurate than the linear transformation at the expense of timing, so it is reasonable to believe that this would improve a non-rigid scheme as well at the cost of computational expense. One of the challenges with affine image registration is that the inverse of a 12×12 matrix is required and openCL does not have a built in inverse matrix calculation. Therefore, the built-in matrix inverse calculation in the ViennaCL library could be used or the SVD (singular value decomposition) could be found of the matrix whose inverse was required. Using the ViennaCL library requires slightly complex set-ups on the CPU side, but is easy to implement once that is done, while the SVD can be set up simply in existing kernels, but the actual algorithm is more complex.

8 | Conclusion

While the non-rigid diffeomorphic image registration code that was developed in this thesis could be used in many different situations, the testing was done only on 3D echocardiographic images to determine diffeomorphic vector fields that represented the movement of the heart. The Farnebäck algorithm was used for image decomposition and a non-rigid direct translation image registration was tested with three different accumulation methods. It was found that of the three methods, the exponential composite method, while computationally expensive, delivered a more accurate and diffeomorphic result. As the heart is only capable of moving in ways that can be described by a diffeomorphic vector field, the exponential composite method is the preferred method. In addition to the accumulation methods, multiple parameters within the code were tested for their effect on the timing and accuracy, and very good results were achieved with the right set-up. The optimal set-up was used to test some real cases and the estimated vector fields were mostly diffeomorphic. Writing the code with parallel programming on the GPU using OpenCL provided good results regarding timing, but comparing the timing on two different computers made it clear that the graphics card influenced the computational expense of the estimation greatly. It is clear, then, that a powerful device will be necessary to ensure a real time result of this method. The fact that we can pick up large deformations, however, can make real-time implementation easier as the image registration will be able to skip some frames and still be able to estimate the deformation field accurately.

Bibliography

[1] M. Ranestad. Fast registration and fusion of echocardiographic images. 2016.

[2] A. Sotiras, C. Davatzikos, and N. Paragios. Deformable medical image registration: A survey. *IEEE: Transactions on Medical Imaging*, 32(7), 7 2013.

[3] H. Zhang, F. Banovac, and K. Cleary. Increasing registration precision for liver movement with respiration using electromagnetic tracking. In *International Congress Series*, pages 571–576. Elsevier B. V., 2015.

[4] M. Dandel, H. Lehmkuhl, C. Knosalla, N. Suramelashvili, and R. Hetzer. Strain and strain rate imaging by echocardiography – basic concepts and clinical applicability. *Current Cardiology Reviews*, 5(2), 5 2009.

[5] C. Szmigielski, K. Rajpoot, V. Grau, S.G. Myerson, C. Holloway, J.A. Noble, R. Kerber, and H. Becher. Real-time 3d fusion echocardiography. *JACC: Cardiovascular Imaging*, 3(7), 2010.

[6] A. Danudibruto, O. Gerard, M. Alessandrini, O. Mirea, J. D'hooge, and E. Samset. 3d farnebäck optic flow for extended field of view of echocardiography. In *Functional Imaging and Modeling of the Heart*, pages 129–136. Springer, 2015.

[7] A. Danudibruto, J. Bersvendsen, O. Gerard, O. Mirea, J. D'hooge, and E. Samset. Spatiotemporal registration of multiple three-dimensional echocardiographic recordings for enhanced field of view imaging. *Journal of Medical Imaging*, 3(3), 2016.

[8] D. Forsberg, M. Andersson, and H. Knutsson. Extending image registration using polynomial expansion to diffeomorphic deformations. 2013.

[9] H. Blessberger and T. Binder. Two dimensional speckle tracking echocardiography: basic principles. *Education in Heart*, 96(9), 5 2010.

[10] A. Elen, H.F. Choi, D. Loeckx, H. Gao, P. Claus, P. Suetens, F. Maes, and J. D'hooge. Three-dimensional cardiac strain estimation using spatio–temporal

elastic registration of ultrasound images: A feasibility study. *IEEE: Transactions on Medical Imaging*, 27(11), 11 2008.

[11] X. Pennec, P. Cachier, and N. Ayache. Understanding the "demon's algorithm": 3d non-rigid registration by gradient descent. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI'99*, pages 597–605, 1999.

[12] G. Farnebäck and C-F. Westin. Affine and deformable registration based on polynomial expansion. *Medical Image Computing and Computer-Assisted Intervention*, 9(1):857–864, 2006.

[13] G. Farnebäck. Two-frame motion estimation based on polynomial expansion. In *SCIA'03 Proceedings of the 13th Scandinavian conference on Image analysis*, pages 363–370, 2003.

[14] G. Farnebäck. *Polynomial Expansion for Orientation and Motion Estimation.* PhD thesis, Linköpings Universitet, Department of Electrical Engineering Linko pings universitet, SE-581 83 Linko ping, Sweden, 11 2002.

[15] G. Janssens, L. Jacques, J. Orban de Xivry, X. Geets, and B. Macq. Diffeomorphic registration of images with variable contrast enhancement. *International Journal of Biomedical Imaging*, 2010.

[16] C. Woolley. Introduction to opencl. http://www.cc.gatech.edu/ vetter/keeneland/tutorial-2011-04-14/06-intro_to_opencl.pdf, 2010.

[17]

[18] D. Coimbra de Andrade. Opencl/opengl interoperation with textures. http://www.cmsoft.com.br/opencl-tutorial/openclopengl-interoperation-textures/, 2010.