# Technologies for Biomechanically-Informed Image Guidance of Laparoscopic Liver Surgery

*Stian Flage Johnsen*

A dissertation submitted in partial fulfillment
of the requirements for the degree of
**Doctor of Philosophy**
of
**University College London**.

Department of Medical Physics and Biomedical Engineering
University College London

Saturday 19th November, 2016

I, Stian Flage Johnsen, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

London, Saturday 19th November, 2016 _____

Stian Flage Johnsen

# Abstract

Laparoscopic surgery for liver resection has a number medical advantages over open surgery, but also comes with inherent technical challenges. The surgeon only has a very limited field of view through the imaging modalities routinely employed intra-operatively, laparoscopic video and ultrasound, and the pneumoperitoneum required to create the operating space and gaining access to the organ can significantly deform and displace the liver from its pre-operative configuration. This can make relating what is visible intra-operatively to the pre-operative plan and inferring the location of sub-surface anatomy a very challenging task. Image guidance systems can help overcome these challenges by updating the pre-operative plan to the situation in theatre and visualising it in relation to the position of surgical instruments.

In this thesis, I present a series of contributions to a biomechanically-informed image-guidance system made during my PhD. The most recent one is work on a pipeline for the estimation of the post-insufflation configuration of the liver by means of an algorithm that uses a database of segmented training images of patient abdomens, where the post-insufflation configuration of the liver is known. The pipeline comprises an algorithm for inter and intra-subject registration of liver meshes by means of non-rigid spectral point-correspondence finding.

My other contributions are more fundamental and less application specific, and are all contained and made available to the public in the NiftySim open-source finite element modelling package. Two of my contributions to NiftySim are of particular interest with regards to image guidance of laparoscopic liver surgery: 1) a novel general purpose contact modelling algorithm that can be used to simulate contact interactions between, e.g., the liver and surrounding anatomy; 2) membrane and shell elements that can be used to, e.g., simulate the Glisson capsule that has been shown to significantly influence the organ's measured stiffness.

# Acknowledgements

First I would like to thank my supervisors Prof. Sebastien Ourselin, Prof. David J.Hawkes, and Dr. Zeike A. Taylor for their guidance and support, and also for organising the funding that made this PhD possible (given my former overseas student status, a far from trivial matter).

Many thanks also to Dr. Matthew J. Clarkson who oversaw my maintenance of the NiftySim software package and work on the NifTK framework, and was a collaborator on most of my publications.

I would also very much like to extend a my gratitude to Dr. Tom Vercauteren and Dr. Fernando Bello, and Dr. Danail Stoyanov, who conducted my PhD viva and my MPhil to PhD upgrade viva, respectively, and whose constructive and detailed comments were instrumental in taking what started as an MPhil upgrade report to the document in front of you.

Finally, a general thank you to all my other paper co-authors, lots of their DNA can be found in this text via their input and helpful comments on the publications that form the basis of most of the subsequent chapters. They are in no particular order: Dr. Stephen Thompson, Dr. Marc Modat, Dr. Yi Song, Dr. Johannes Totz, Dr. Kurunchi Gurusamy, Prof. Brian Davidson, Dr. John Hipwell, Dr. Bjoern Eiben, Dr. Lianghao Han, Dr. Yipeng Hu, Dr. Thomy Mertzanidou.

The funding for my work was provided by the Intelligent Imaging Programme Grant (EPSRC reference: EP/H046410/1) and the EU-FP7 project PASSPORT (ref.: ICT 223894).

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Clinical Motivation

Liver cancers are normally divided into primary liver cancers, mostly hepatocellular carcinoma (HCC), and metastases from cancers of a range of other organs. In the United Kingdom, 5413 new cases of primary liver cancer were recorded in 2013 with an increasing trend. Worldwide the picture is direr, in excess of 782,000 new cases were diagnosed in 2012[1]. In 2008 it was the third leading cause of cancer deaths worldwide, with hepatitis B and C being named as the primary causes of the disease [116]. It has been reported that as many as 40-50% of extrahepatic cancers can lead to metastases in the liver [6]. Of the HCC cases, only 5-15% can be treated surgically, and alternative treatment options have an unsatisfactory success rate [43]. The overall 5-year survival rate for HCC is only 30-40% [43] and a 5-year survival rate of 36-58% has been reported for metastatic colorectal cancer [91].

Surgical resection of the parts of the organ affected by cancer, whether primary or metastatic, is considered the gold standard treatment [45, 88]. Laparoscopic liver resection (LLR) has a number of medical, and ultimately financial, advantages over open surgery, such as lower blood loss during the procedure, fewer post-operative complications and, as a result, quicker patient recovery times [35, 33]. LLR is typically only considered if the resection target is sufficiently small, localised and easily accessible, i.e. a part of the left lobe or an anterior/inferior, peripheral part of the right lobe. However, the type of resections performed laparoscopically range from non-anatomical wedge resections of small peripheral lesions to segmentectomies (removal of an entire Couinaud segment) and even hemihepatectomies (removal of an entire half of the liver), although a major laparoscopic resection on the right lobe normally requires hand ports (hand assisted procedure) and is considered technically very challenging [21, 79, 26].

Standard LLR procedures are conducted with the patient in a supine position (on their back) with the surgeon positioned between their legs, and monocular laparoscopic video and laparoscopic ultrasound (LUS) used for intra-operative guidance. The abdomen is insufflated with $CO_2$ to create the operating space for the surgeon (pneumoperitoneum) and with 12mmHg

[1]Incidence figures courtesy of Cancer Research UK. Retrieved 06/08/2016: `http://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/liver-cancer/incidence`

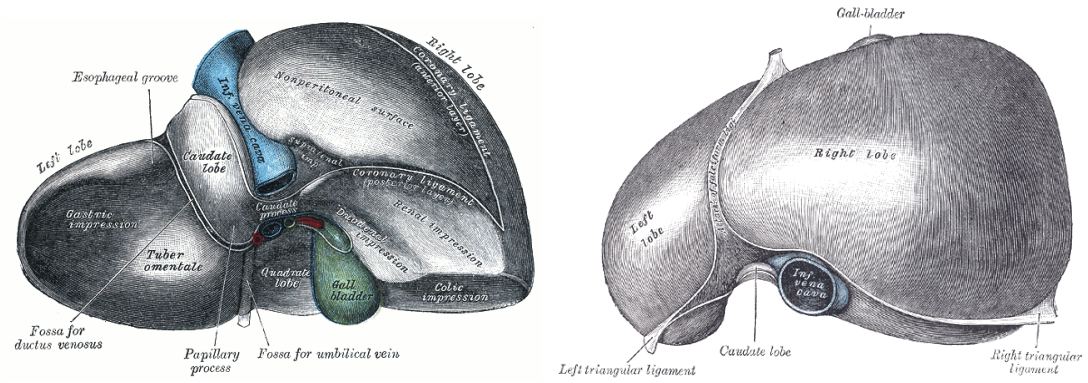



Figure 1.1: Illustrations of the anatomy of the human liver courtesy of Gray and Lewis [48]. Left: posterior view. Right: cranial view.

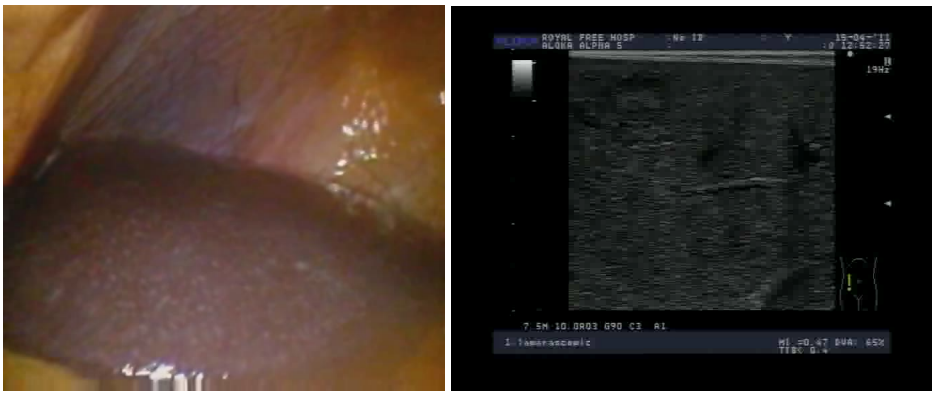


Figure 1.2: Left: uncropped video frame from a laparoscopic inspection of a left lobe of a human liver with parts of the falciform ligament in the left part of the picture and the diaphragm visible in the background. Right: Frame from a left lobe laparoscopic ultrasound scan acquired during the same procedure. The image data was acquired for the PASSPORT project.

(1.6kPa) above atmospheric being a typical gas pressure. Multiple ports with varying sizes are placed between the umbilical and the ribcage for the camera, instruments, mobilisation of the liver, and extraction of excised material. The actual procedure normally begins with the surgeon examining the organ with the video camera and LUS, to amongst other things look for lesions that went undetected in pre-operative scans. Cutting of the falciform, coronary, and/or round ligament may be required to mobilise the liver and gain access to the areas affected by the procedure. Precautionary measures for aiding an emergency clamping of the portal vein and hepatic artery in the event of excessive bleeding (Pringle manoeuvre) are typically put in place. The transection of the liver parenchyma can then be performed with, depending on the depth, the harmonic scalpel and/or the ultrasonic dissector. Where necessary, clipping/stapling of biliary and blood vessels leading to the resected region that are too large to cauterise is carried out [21, 26]. Lesions are ideally excised with negative margins in the range 1-2cm [33]. Fig. 1.1 shows the anatomy of the human liver for better understanding.

While beneficial from a patient perspective, LLR poses a number of intrinsic technical challenges: The procedures are typically planned based on computed tomography (CT) or magnetic resonance images (MRI) with a high signal-to-noise ratio (SNR) and contrast enhancement to highlight the location of blood vessels. Occasionally, positron emission tomography (PET) is employed for tumour localisation. The imaging modalities routinely employed during the procedures, laparoscopic video and LUS, offer only a very limited field of view (Fig. 1.2), though, that is frequently further reduced by blood staining of the video lens and smoke formation. A lack of easily identifiable landmarks make the relating of what is seen to the larger anatomical context of the pre-operative images difficult [92, 106, 59]. The estimation of the location of sub-surface anatomy is further complicated by the insufflation of the patient abdomen. It leads to a deformation of the organ and its surroundings with respect to the pre-operative configuration, with the liver and diaphragm having been observed to move several centimetres in pig experiments [58, 121]. Finally, the option of palpating tissue, e.g., to locate tumours, is not available in laparoscopic procedures.

Practically, these complications lead to situations where, in some patients, a laparoscopically accessible part of the liver cannot be resected laparoscopically due to concerns over cutting into major vascular or biliary structures. These patients are considered ineligible for a laparoscopic procedure and must undergo the more arduous open surgery [144, 21]. Patients may also become ineligible when preservation of sufficient liver volume can't be ensured. This is especially a problem in patients with a liver weakened by, e.g., cirrhosis, and entails that as much as possible of the vascular and biliary structures to and from non-resected regions must be maintained, and clipping/stapling be kept at a minimum [144, 33], which in turn demands more precision from the surgeon.

## 1.2 Image-Guidance

Image guidance aims to overcome many of the problems of LLR by updating the pre-operative image data to the intra-operative situation and giving the surgeon a more accurate understanding of the regions of the liver they are looking at through the laparoscope and that they are about to cut into. By showing the location of invisible subsurface structures image-guided surgery (IGS) may enable the surgical treatment of patients who would otherwise have been considered inoperable with LLR, or even generally inoperable [77]. While providing image guidance becomes very difficult once the parenchymal transection has begun, the usefulness of these systems is greatest after the mobilisation of the liver since at this point the surgeon has the option of outlining resection planes or mark points of interest on the liver surface with an electrocautery device [128].

Many of the proposed image-guidance systems introduce additional hardware, such as intra-operative 3D imaging scanners, into the workflow. In the case of intra-operative CT or MR imaging [109, 80], these systems come with great financial costs and added radiation and magnetic hazards. The envisaged system that provided the motivation for my work,

aims to make do with a tracked stereo laparoscope and LUS, i.e., hardware that is mostly a comparatively inexpensive upgrade on standard laparoscopic surgery equipment, to create targets for the registration of pre-operative images. With this type of system, the surgical workflow is only minimally affected and the surgical staff do not need any significant training to use it, and it also leads to a system that can be widely deployed with minimal hardware procurement costs. Thompson et al. [144] found by working backward from the desired clinical outcome that, to ensure the excision of the tumour within the safety margins defined by the surgeon, and more importatantly avoid damage to blood or biliary vessels too large to be sealed with the harmonic scalpel, an image-guidance system for LLR should have about 3mm total system accuracy as a realistic goal. Typical metrics for measurement of the overall clinical outcome of novel laparoscopic procedures include: 3/5-year survival rate, length of post-operative hospital stay, recurrence rate, rate of conversion to open surgery, blood loss, operating time, and tumour margin [144, 21, 26]. The latter can be evaluated relatively easily for any given procedure, but a direct evaluation of the image-guidance accuracy on patient data would require the acquisition of additional data, such as volumetric intra-operative imaging data. Hence, phantom experiments and animal models are most frequently employed in the evaluation of image-guidance system accuracy.

## 1.3 Overview Over My Work

A 3D to 2D image registration problem is very ill-posed in most circumstances, even without the aforementioned exacerbating factors present in laparoscopic procedures, and while methods for direct 2D-3D image to model registration exist [29], they have, to the best of my knowledge, never been extended to incorporate non-rigid deformation and successfully employed in guiding endoscopic or laparoscopic surgery. Therefore, additional constraints need to be imposed on the problem. What has been focused on is the registration of pre-operatively created meshes to sparse point clouds representing the intra-operative organ surface, and incorporation of biomechanics to simulate the organ's motion based on sensible material parameters and segmentations of their pre-operative configuration, and the learning of the deformation from training data sets comprising segmented intra-operative 3D image data. A biomechanical model allows for an improved registration between pre-operative segmentations and intra-operatively acquired surface reconstructions. It also provides realistic estimates of the location of sub-surface structures such as blood vessels or lesions. This estimate can be further improved by imposing sub-surface displacement constraints, e.g., by identifying corresponding blood vessel branchings in the pre-operative data and LUS.

The biomechanical simulation work I have conducted is using the open-source NiftySim finite element modelling (FEM) software package. NiftySim contains C++ and CUDA implementations of the Total Lagrangian Explicit Dynamics (TLED) algorithm, that has been shown to be a very fast option for simulation of soft tissue biomechanics [53, 152], and a significant portion of my work has been on the extension of this software with features required for the in-

tended application in image guidance for laparoscopic liver surgery, such as a general-purpose contact modelling algorithm, that can simulate contacts between the liver and surrounding viscera, and membrane elements for the modelling of e.g. the Glisson capsule surrounding the liver. Being general-purpose FEM algorithms, these new features have found applications in other areas as well, such as the mammography-image registration algorithm of Mertzanidou et al. [95].

Typically when biomechanical simulations are used in image-guidance systems in conjunction with optical surface reconstruction, it is by computing displacements via point-set registration applied to point-clouds intra-operatively recovered from the surface of the target organ and meshes generated from segmentations of pre-operative data. These point-set registration algorithms have a tendency to converge to local minima of the cost function, unless they are initialised with an already reasonable alignment of the two point sets to be registered. A non-parameteric machine learning algorithm was developed to provide an estimate of the organs shape after insufflation of the patient for the purpose of improving the chances of finding the correct point correspondences between the pre-operative data and the reconstructed surfaces. It is built on a database of segmented pre and intra-operative images of patient abdomens. Spectral mesh registration is employed to determine the insufflation displacements on the liver surface in the training data sets, as well as for establishing point correspondences between training and patient segmentations. These correspondences are then used to point-wise compute local similarities between training and unseen livers, which in turn can be used to interpolate between the known displacements and provide an estimate of the deformation the patient liver is likely to undergo due to pneumoperitoneum.

## 1.4 Organisation of This Thesis

The next chapter, Chapter 2, gives an overview of the literature on IGS, with a focus on biomechanically-informed methods and laparoscopic liver surgery. The literature review attempts to cover the entire guidance pipeline from segmentation of the pre-operative, high-quality image data to the point where it is warped to match the intra-operative configuration of the anatomy prior to the start of parenchymal transection. The algorithm for estimation of the liver post-insufflation configuration is discussed in Chapter 3. A description of the software package NiftySim is given in Chapter 4, and a more detailed description of the work I did on NiftySim's contact modelling pipeline can be found in Chapter 5. Finally, Chapter 6 contains a discussion of these contributions and how they fit into an image guidance pipeline, and how they could be extended in the future to even better fulfill their role in that context.

## Chapter 2

# Literature Review

This chapter gives an overview of the algorithms and components comprised in an image-guidance system, from segmentation of pre-operative data to visual output to the surgeon. The focus is on methods for the recovery of the intra-operative configuration of the liver using optical methods and the registration of the pre-operative data to that reconstruction of the intra-operative liver surface.

## 2.1 Pre-Operative Segmentation and Intra-Operative Surface Recovery

Regarding the segmentation of pre-operative data, a fairly recent survey of algorithms specifically for extraction of the liver from CT images, was conducted by Mharib et al. [97]. There are also commercial services providing high-quality segmentation of abdominal images. One such service is offered by the company Mevis[1], and a beta service is provided by Visible Patient[2].

Passive approaches to the problem of reconstruction of the organ's surface from intra-operatively acquired video, i.e. such that rely solely on the standard video data, include: Reconstruction from stereo-video, such as [130], which requires the use of a calibrated stereo-video laparoscopes and algorithms for performing feature matching in the stereo images. Work has also been done on creating algorithms that reconstruct the 3D surface by matching features found in different frames of the same monocular video, which goes by the name of Structure from Motion algorithm [61]. While the latter has the advantage that it can use a standard mono laparoscope, it is computationally expensive and may require additional steps in the surgical workflow to acquire a sequence of images showing the surface from sufficiently many angles to allow for a 3D reconstruction. Further, its applicability is severely hampered by the deformability of the geometry [92]. Shape from Shading [159] can reconstruct surfaces from single frames in a post-processing step. However, it was derived based on the assumption of a *Lambertian* surface, and special lighting models must be adopted, incorporating specular highlights. A more comprehensive list of passive surface reconstruction methods and

[1]Company website: `http://www.mevis.de`

[2]`http://www.visiblepatient.com/en/service/`

corresponding references can be found in Maier-Hein et al. [92].

Among the active systems for intra-operative surface reconstruction, the incorporation of 3D laser range scanners (LRS) in open liver resection was first proposed more than 10 years ago by Cash, Miga, et al. [25, 24], and demonstrated a high degree of accuracy in tracking the surface. However, laparoscopic LRS systems [117] are still experimental hardware, and no such systems are currently commercially available. The use of a structured-light source, e.g. [3], that projects a fixed pattern on the geometry and exploits a known geometric relationship between camera and light source to reconstruct a surface, is an active optical option with an acceptable hardware overhead. This technique can be used for reconstruction of surfaces from mono-video, but it also has applications in conjunction with stereo-video where it is used to improve the reliability of the reconstruction of surfaces with few intrinsic features at the expense of point-cloud resolution [92]. Time of Flight is another, more experimental, active optical method based on a principle akin to radar, that can also be found in Maier-Hein et al.'s overview of optical surface recovery methods [92].

Volumetric intra-operative imaging modalities, relatively common in neurological procedures, have been used for laparoscopic surgery as well. The most commonly encountered ones are intra-operative CT [109, 43, 123] and MRI [74, 156]. The use of 3D ultrasound has also been proposed [80, 126]. While ultrasound does not provide the same FOV and signal-to-noise ratio of the former two modalities, it is significantly less costly to operate and procure, it does not entail any health hazards to the clinical staff or the patient, unlike CT, and does not require the use of special instruments and materials, unlike MRI. Further, being a volumetric modality, its registration to pre-operative data is easier than that of 2D US, although the latter has also been demonstrated [120]. If sufficiently accurate tracking is available, 3D volumes can also be reconstructed from images acquired with 2D probes [81].

However the intra-operative surface configuration is acquired, point-correspondences to the pre-operative images need to be established.

## 2.2 Instrument Tracking

Instrument tracking is of crucial importance in image guidance of minimally invasive surgery (MIS). It provides a common coordinate system in which surface patches reconstructed from frames of the laparoscopic video feed can be related to, e.g., blood vessel displacement information extracted from LUS data. However more importantly, it allows to visualise the tracked instruments in relation to segmented, high-quality pre-operative image data, thus helping the surgeon to resect the right parts of the organ and avoid damaging vital structures, such as major blood vessels.

A common choice for image guided surgery, due to its relatively low overhead and high compatibility, is optical tracking of infrared light emitting diodes (LED) attached in a pattern to collars that are in turn attached to the proximal end of the instruments. Its main disadvantage is that it requires a clear line of sight between the tracking camera and the LEDs. The error of

these systems in tracking the LEDs is typically very low, i.e. sub-millimetre, but this error is propagated from the proximal end of the laparoscope down to its tip, where rotational errors can get significantly amplified. Nonetheless, robust tracking of the laparoscope tip with an error of 1.6mm is achievable with a well designed tracking collar [145, 144]. Feuerstein et al. [43] proposed a system solely relying on optical tracking to overlay intra-operative CT and laparoscopic video. In phantom experiments with conditions resembling the surgical situation, they determined the root mean square (RMS) error to be 1.58mm for the overlay.

Another popular choice is the use of electro-magnetic tracking (EM tracking). It is less error prone than optical tracking, due to not being subject to the line-of-sight restriction of optical tracking, and very useful when working with articulated instruments due to having sensors that can be attached directly to the distal end of the instruments. On the other hand, these systems are sensitive to ferro-magnetic metals and electrical interference, and their use can therefore put restrictions on the kind of instruments being used and other nearby objects in the operating theatre [13]. That situation appears to have improved, in recent years, though [126]. EM trackers have tracking errors very comparable to optical tracking, i.e., in the 1mm range, when relatively close to the field generator [126].

An interesting alternative to the above methods is offered by visual instrument tracking, as it dispenses with any additional hardware and solely relies on the video feed. In [4], an algorithm to estimate the 3D pose of instruments from segmented laparoscopic stereo-video images was presented by Allan et al. The segmentation was done automatically with a random-forest classifier, and the segmentations then served as the input for a level-set algorithm giving the instrument pose estimate. Mainly to improve the robustness of the pose estimates through temporal constraints, they also added optical-flow based tracking of instrument features. The disadvantages of this approach include a rather large translational error in the out of image plane direction for the pose estimate, and that it was only validated for the better constrained case of robotic MIS and a need for manually labelled training images.

Other approaches to instrument tracking, such as mechanical tracking, are covered in the survey on IGS methods by Baumhauer et al. [13]. Solberg et al. [126] evaluated the accuracy of a number of systems for the tracking of LUS probes.

## 2.3 Correspondence Finding

The methods for registration of point clouds found in the IGS literature are frequently derived from the Iterative Closest Point (ICP) algorithm of Besl and McKay [17]. ICP takes its name from the algorithm, in every iteration, matching every point of the source point cloud – in this context sometimes referred to as the *data* – to the closest point in/on the target point-cloud/surface – also called the *model*. With these correspondences, an optimal rigid-body transform for the source point set can be found by means of the closed-form solution to the orthogonal Procrustes problem [7].

Su et al. [132] devised a mesh registration algorithm for guidance of minimally-invasive

partial nephrectomy that used the basic ICP algorithm and an anchoring via a few known reference points. Their algorithm also comprised a method for estimation of the part of the pre-operatively generated mesh visible in a given video sequence, which served as the source mesh in the ICP registration to a surface extracted from the video frames. They reported a surface registration error of less than 1mm for the method and an update frequency of 10Hz using only, by today's standards, very modest computer hardware. However, it has to be noted that providing image guidance for kidney surgery is greatly simplified by the stiffness of the organ.

Maurer et al. [93] developed an extension of ICP where source and target are represented by a weighted combination of geometrical feature shapes which can be surface patches, but also, e.g. blood-vessel centrelines. They called their method Weighted Geometrical Features (WGF) algorithm. A slightly modified version of Maurer et al.'s WGF method was used in a pipeline proposed by Clements et al. [31] for the image guidance of open liver surgery via LRS. They named their method Weighted Patch ICP (wICP) algorithm, and the patches giving rise to the name were anatomical features, e.g. falciform ligament, extracted from the intra-operative image data via texture and colour information. A key mechanism of their algorithm was the reduction of the Euclidean distance in the nearest point operator for points, in the source and target point sets, believed to be part of the same anatomical feature. Dumpuri et al. [38] used wICP to determine displacements for the visible liver surface in their work, this displacement was then diffused over the entire liver surface by solving a Laplacian equation, and used to drive a biomechanical simulation.

Bano et al. [11] also proposed a biomechanically-informed registration algorithm for liver MIS. They attempted to overcome the need for complete surface data or a high-quality initialiser with sophisticated point matching, in registration of pre-operatively generated liver meshes to the intra-operative configuration extracted from C-arm images. The algorithm comprised a rigid initialisation of the transform performed on the posterior surface of the liver, where the deformation due to pneumoperitoneum is known to be small. In a subsequent non-rigid registration step, they identified common anatomical landmarks on the liver's anterior surface from which they constructed a surface coordinate system based on geodesic distances to the landmarks, to provide dense correspondences between the surfaces, thus effectively performing an ICP-type nearest-neighbour point matching in a custom coordinate system. Finally, the anterior surface point correspondences were used to define displacement boundary conditions on a biomechanical simulation to propagate the displacements obtained for the anterior surface to the other parts of the liver.

The robustness of ICP was looked at by Benincasa et al. [16], who investigated how much of the organ surface is needed for a successful ICP registration of a pre-operatively generated mesh to an intra-operatively recovered surface. The study was based on a kidney phantom whose surface was reconstructed with LRS, and kidney resection was the only type of procedure

considered. Their findings indicate that approximately 28% of the kidney surface must be recovered for a robust registration with larger perturbations. However, they did note that by spreading out the recovered surface patches and including geometrically intricate features that number could be reduced.

A more theoretical overview of algorithms evolved from ICP can be found among other methods in Audette et al. [8]. A point that received particular scrutiny in their survey is that of transforms for alignment of source and target, beyond the rigid-body transform of the basic ICP. Examples of non-rigid transforms covered are the Thin Plate Spline (TPS) and B-splines. Another area of interest in this context, that was extensively covered, is that of augmentation of the registered point sets with additional information such as surface curvature. A varied list of references on the augmentation of point sets can also be found in Liu [87].

Two classes of registration methods not covered in Audette et al.'s review, and that are used in the pipeline presented in Chapter 3, are spectral point matching and Gaussian Mixture Model (GMM) based methods:

Spectral embeddings have received some attention as a means for dimensionality reduction and data clustering [14], but they also provide a powerful method for the registration of surfaces with identical topology. In most spectral methods, a *spectral embedding* is computed for source and target by computing the eigenvalue decomposition (EVD) of the graph Laplacian ($\boldsymbol{L}$), corresponding to the mesh. Doing so, one has to compensate for the fact that the orientation of eigenvectors is not fixed, i.e. for a given eigenpair $\lambda$, $\boldsymbol{v}$ it holds both $\lambda\boldsymbol{v} = \boldsymbol{L}\boldsymbol{v}$ and $\lambda(-\boldsymbol{v}) = \boldsymbol{L}(-\boldsymbol{v})$. Further, if there are very close eigenvalues involved, the ordering of the eigenvectors can switch. These phenomena were referred to as *eigenmode non-robustness* by Jain et al. [66]. Lombaert et al. [89] solved this problem by coupling the Laplacians of the meshes being registered, and computing the EVD of one large Laplacian matrix for the entire registration problem. However, this approach necessitates a good initial guess for the correspondences between the points in the two meshes. Jain et al. [66], whose spectral embeddings were computed from Gaussian affinity matrices, in turn computed from geodesic distances rather than graph Laplacians, solved the eigenmode non-robustness problem with a greedy algorithm that built the embeddings eigenpair-by-eigenpair and minimised the distance between corresponding points in the intermediate spectral embeddings. Since this type of matching only relies on the relative position of the mesh vertices, any rigid misalignment and differences in scale between the meshes is ignored. While these methods are very elegant, they cannot be used for the registration of intra-operatively recovered surfaces to segmented pre-operative image data, due to the pre-operative mesh and the surface patches having different surface topologies.

In the GMM-based methods a GMM, in which every source node gives rise to its own *normal distribution*, is fitted using Expectation Maximisation (EM). In the presence of noise or when it is known that the source is only a subset of the target surface, an additional outlier distribution is frequently introduced in the GMM. Assignment of target-mesh to source-mesh

nodes is probabilistic and expressed through a correspondence matrix $\boldsymbol{M}$, where rows hold the assignments of source to target points, and columns represent target points and satisfy $\sum_i m_{ij} = 1$. This matrix replaces the hard, 0 or 1, assignments of source to target nodes in ICP. An extension and comparison of the most common representative of this family, EM-ICP, and a relative incorporating simulated annealing, SoftAssign, can be found in Liu [87]. Evolutions of these GMM-based methods have also frequently incorporated non-rigid deformations with TPS being a popular transformation model, which is used in the non-rigid version of the Robust Point Matching (RPM) algorithm [27]. A particularly versatile representative of the GMM family of algorithms, called Coherent Point Drift (CPD), was introduced by Myronenko and Song [105]. The algorithm allows for general non-rigid deformation of the source, while a Gaussian regulariser ensures that the point-set topology is preserved. Further, unlike most of the non-rigid registration algorithms covered so far, it is not limited in its applicability to 2D or 3D data and can process data of any dimensionality, such as $k$-dimensional spectral embeddings or augmented spatial point clouds. A modified version of this algorithm was also used by Hu et al. [64] to register blood-vessel centrelines, e.g., such extracted from pre-operative CT data to LUS. Their algorithm added landmark constraints to the displacement field regularisation term to guide the registration with prior knowledge.

An alternative to repeated registration with a point cloud registration algorithm is offered by feature tracking. Stoyanov et al. [129] presented a method for the recovery of intra-operative surfaces from stereo laparoscopic video and tracking of the image features used in the reconstruction over an extended period of time. In Pratt et al. [113] that tracking algorithm was used in a guidance system for robotic heart surgery to impose boundary conditions on a biomechanical model of the heart, by determining for a simplified heart mesh, pre-operatively generated from 4D CT data, the intra-operative surface displacements. They combined these tracked features on the visible surface with a computational model of the cardiac motion, also generated from the 4D CT data, to obtain sensible displacements for the whole of the heart. Haouchine et al. [54] proposed a system that, starting from a mostly manual initialisation, tracks the displacement of feature points on the liver surface using the optical flow algorithm of Lucas and Kanade [90]. A biomechanical model was used for the regularisation of the feature point displacements.

Simultaneous Localisation and Mapping (SLAM) is a class of algorithms that, given an initial camera position and world coordinates for features detected in the video stream, can solve both for the camera position in subsequent steps as well as track the world coordinates of the features, even if they are part of moving or even non-rigidly deforming structures. The algorithm works by iterative prediction of the camera/scene state and refinement of the prediction based on measurement of the error in the backprojected estimated feature position. Early real-time SLAM work was carried out by Davison [36]. Using a Bayesian Extended Kalman Filter (EKF) framework, the algorithm estimated the position of a mono-camera moving on a smooth

trajectory through a static scene via image patches serving as the features. Mountney and Yang [104] combined EKF SLAM with an analytic motion model to compensate for respiratory motion in minimally invasive liver surgery. They named their method Motion-Compensated SLAM (MC-SLAM). With this relatively simple approach to motion correction, they achieved a more than 5-fold improvement in accuracy over the basic static SLAM algorithm, in experiments on a virtual phantom and an ex-vivo tissue sample. Grasa et al. [47] developed a SLAM variant suitable for laparoscopic applications that incorporated RANSAC and Randomised List Relocalisation to account for tissue deformation, occlusion by instruments, and sudden camera movement. However, the algorithm was limited in that it was not fully capable of running in real-time and it only allowed for temporary tissue deformation. Agudo et al. [1] developed a method called FEM-EKF SLAM that used FEM to model the non-rigid deformation of the scene geometry.

## 2.4 Initial Pose Estimation

A common problem with ICP-like algorithms is their tendency to converge to local minima of the cost function and their need for a good initial-guess alignment of the surfaces being registered. In neurological procedures, the rigidity of the patient's skull can be exploited via bone-implanted or externally applied markers, or a stereotactic frame to provide a sufficiently accurate initial guess [93].

While a rough initial alignment can still be obtained in liver resection procedures, with e.g., skin-applied markers, a rigid alignment of landmarks is normally not sufficient to avoid misregistration of reconstructed surfaces and pre-operative configuration meshes due to the sizeable deformation. In laparoscopic procedures, this deformation is mainly caused by the insufflation of the patient, whilst in open procedures, by the opening of the abdomen and packing of the liver. The deformation extends to much of the surrounding anatomy that itself is highly deformable. One option for obtaining a better alignment is the introduction of a semi-automatic initialisation by the surgeon, e.g. Dumpuri et al. [38] employed an optically tracked pen with which the surgeon had to "select" pre-operatively determined anatomical features in the operating theatre. Another interactive option is the manual alignment, in theatre, of the pre-operatively generated model to the anatomy visible in the video feed, e.g., by manipulation of the mesh [106]. Zhang et al. [158] proposed the pre-operative implantation of electro-magnetically tracked needles in the liver that, while quite invasive, could be used in a manner similar to bone implanted markers in neurosurgery. Stefansic et al. [128] described a somewhat related method. Their approach was to create artificial landmarks on the liver surface with an electrocautery device to facilitate registration. However, the application of this technique was proposed for a different purpose, that of repeated direct linear transform of 3D data to a 2D laparoscopic video overlay. Bano et al. [10] presented a simple algorithm for the prediction of the post-insufflation position of arteries in the abdominal wall, a task that is of great importance for the correct placement of surgical ports. They employed skin-

attached optical markers to determine the displacement of the outer surface of the abdominal wall and propagated the resulting displacement through linear interpolation to points inside the volumetric mesh they constructed for the abdominal wall. They measured an average error of 6mm on artery bifurcation landmarks using their method, which they later improved upon with a biomechanical model [9].

## 2.5 Biomechanical Simulation

FEM has been the most widely adopted technique for simulation of biomechanics in IGS, in recent years [23]. Other continuum mechanics-based approaches have been explored for simulation of soft-tissue biomechanics and shown to provide comparable deformation results. Examples include: meshless methods that evaluate the continuum-mechanical governing equations on arbitrarily positioned nodes and avoid the volumetric discretisation of FEM [99, 149, 157]; the boundary element method (BEM) that eliminates the interior of the body from the problem and solves the motion equations solely for the surface of an elastic body [103]. However, rarely do these approaches have the accuracy of FEM in terms of strains and stresses, nor have any gained nearly as much traction in the community as FEM, and therefore the same wealth of available implementations and theoretical extensions for scenarios such as contact or cutting.

FEM requires the spatial discretisation of the geometry into smaller geometric primitives in which the strains and stresses in the material are evaluated [12], with the geometry typically being extracted from a pre-operative image. Most IGS algorithms formulate the problem such that a number of time-invariant boundary conditions and loads are defined, corresponding to the intra-operative situation, and essentially one could treat the problem with a *static* solver, i.e., one that assembles a system of the type

$$\boldsymbol{K}\boldsymbol{U} = \boldsymbol{F} \tag{2.1}$$

where, ideally, $\boldsymbol{U}$ are the displacements warping the pre-operative geometry into the intra-operative configuration, $\boldsymbol{F}$ are the accumulated external loads, i.e., bodyforces (typically gravity) and surface loads, and $\boldsymbol{K}$ denotes the stiffness matrix, computed using a material constitutive model. Eq. (2.1) is then solved for $\boldsymbol{U}$ and one obtains the displacements corresponding to the equilibrium of the internal stresses with the external loads. Displacement boundary conditions are imposed by elimination of the rows and columns corresponding to the constrained mesh vertices from (2.1) and an update of the external load vector $\boldsymbol{F}$ [12].

If non-linear constitutive models are used, the stiffness matrix $\boldsymbol{K}$ becomes a function of the displacement and eq. (2.1) has to be solved iteratively. It has been shown repeatedly that a treatment of the problem as a dynamic problem, i.e., one containing an inertial term

$$\overbrace{\boldsymbol{M}\ddot{\boldsymbol{U}}}^{\text{inertia}} + \boldsymbol{K}(\boldsymbol{U})\boldsymbol{U} = \boldsymbol{F} \tag{2.2}$$

can be significantly faster, particularly if a solver is chosen that can be efficiently implemented on GPUs, such as TLED [138, 53].

Simulating the 3D motion of a body with FEM requires the generation of solid meshes, i.e., 3D meshes consisting of 3D elements, for the anatomy whose deformation is to be modelled. The most common element choice for irregular geometry, such as that of an internal organ, is the linear, 4-node tetrahedron. Since the tetrahedron is the Euclidean simplex in $\mathbb{R}^3$, the discretisation of the geometry is significantly simpler with this type of element, as are certain modelling operations such as spatial integration. This fact is also reflected in the sheer number of open source and commercial tetrahedral meshing solutions, such as GMSH[3] and TetGen[4]. Their main disadvantage for soft tissue simulation is their inability to simulate nearly incompressible materials due to *volumetric locking*, although there are remedies for this problem, such as the average nodal pressure (ANP) tetrahedron [73] and displacement-pressure formulations with the addition of *bubble functions* [160]. Some of the following examples use tri-linear hexahedral elements, which have the advantage of faster computation times, since their use frequently leads to fewer elements in the mesh, and they, unlike tetrahedral elements, are compatible with a displacement-pressure mixed formulation of the governing equations, in turn allowing for simulated incompressibility [12]. However, there are few general purpose hexahedral mesh generators, and those which one can find are typically commercial software, e.g., ANSYS ICEM CFD Autohexa[5]. If multiple tissue types with distinct material models are considered, one has the option to use a mesh generator that has the ability to generate labelled meshes from multi-label masks such as the Computational Geometry Algorithms Library (CGAL)[6]. This can lead to a very large number of elements, though, if the boundary between the different tissues has a complex geometry. Alternatively, one can use a single-domain mesher with a fixed element diameter and subsequently determine which elements belong to which tissue, as proposed by Han et al. [53].

The use of FEM-based biomechanical models in IGS probably has the longest history in neurosurgery where *brainshift*, the deformation of the brain after opening of the dura, poses problems akin to those caused by liver deformation in liver-cancer surgery. The history of these methods goes back some 15 years [42, 125, 23, 152]. Initially, a distrust of the regularisers employed by standard image registration algorithms at the time (e.g., Demons algorithm with fluid regularisation) played an important role in their adoption. Intensity-based image registration algorithms are the primary competition to biomechanical simulation in these procedures due to the intra-operative application of dense volumetric imaging, through interventional CT or MRI, being more common in neurosurgery. This early FEM-based work in the area of brainshift modelling mostly employed linear elastic models of the brain, but tissue inhomogeneity was cited by Ferrant et al. [42], more than 12 years ago, as a major motivation for using a biomechanical model instead of a conventional image registration algorithm. Their method employed a

[3]Open source, available from `http://www.geuz.org/gmsh`

[4]Open source, available from `http://wias-berlin.de/software/tetgen/`

[5]Company website: `http://www.ansys.com/`

[6]Open-source, dual-licence, available from: `http://www.cgal.org`

combination of rigid image registration and an active-surface boundary tracking algorithm [41] to define displacement boundary conditions on the biomechanical model. Despite their simple biomechanical model, they achieved a satisfactory registration error for the intended application with a sub-surface error of less than 3mm. The work of Dumpuri et al. [38], mentioned in Sect. 2.3, also employed a homogeneous linear elastic constitutive model for the liver tissue. Using this biomechanical model they managed to improve their registration errors obtained with wICP by a factor of 2-3 in phantom experiments and most of their clinical test cases.

Later work incorporated non-linear deformation models. Particularly, Wittek et al. [152] employed GPU-accelerated TLED with an inhomogeneous neo-Hookean tissue model for modelling the brain in neuro-surgical resection procedures, with different material parameters for brain parenchyma, tumour, and the cerebrospinal fluid. They further employed contact modelling to simulate the sliding interaction between brain tissue and skull to achieve accurate results on comparatively inexpensive computer hardware, in a timeframe that is easily reconciled with the surgical workflow.

The work on modelling the biomechanics of breast tissue for IGS has to deal with very soft tissue undergoing large deformation, similar to what we are faced with in liver IGS. Here too inhomogeneous models are commonly found, even in older work such as Samani et al. [119]. One particular area of focus has been the registration of diagnostic MRI images acquired with the patients in a prone position to the surgical setting where the patients lie on their backs (supine) [22, 51]. For this application, the biomechanical model is frequently used to provide an initial guess for a subsequent registration of pre-operative to intra-operative MR images with an intensity-based method. Han et al. proposed a pipeline comprising a biomechanical model with iteratively refined parameters for initialisation of a non-rigid B-spline-based image registration [51]. The biomechanical simulation was carried out with GPU-accelerated TLED. The parameters of the simulation were optimised with simulated annealing and comprised material and mesh-position parameters. For the tissue material model, they chose a homogeneous, isotropic neo-Hookean model with a Poisson ratio near the incompressible limit of 0.5 and a Young's modulus between 0.5 and 50kPa, i.e., they considered a range of two orders of magnitude for the material stiffness. Their model employed a fixed gravity bodyforce driving the deformation and a simple contact model to simulate the sliding of the breast with respect to the chest wall. With this relatively simple model, they achieved maximum fiducial errors of less than 4mm. In a later, more general formulation of their algorithm [53], they used models comprising up to 5 tissues with distinct stiffness values – fat, fibroglandular, muscle, and tumour tissue, and incorporated the effects of material anisotropy by using a different tissue compliance along the vertical coordinate axis. While their algorithm is widely applicable, its use with only intra-operative surface data would require significant modifications, due to their reliance on an image similarity metric for the assessment of the quality of the simulation parameters. However, if dense volumetric intra-operative images are available, the basic idea behind the

algorithm can be applied to laparoscopic liver surgery as well, as shown by Oktay et al. [109]. They created a very similar algorithm, incorporating a simple biomechanical simulation driven by a pressure boundary condition, in turn used to simulate the insufflation of the patient, and consisting of a liver mesh with homogeneous material properties and two other meshes representing the surrounding tissues, with distinct material settings. Their mesh-to-mesh surface distance results mostly below 3mm on the liver are very satisfactory. It has to be said, though, that for clinical practice this is a poor error metric, since it does not allow for an assessment of the error on subsurface structures such as blood vessels.

Bano et al. [9] employed a simulation setup similar to that of Oktay et al. [109], to model the displacement of the viscera and the abdominal wall due to pneumoperitoneum. Their simulation comprised three bodies with different material settings: the abdominal viscera, including the liver, the thoracic viscera, and the abdominal wall. The interactions between these meshes was simulated with a contact model. The meshes were first subjected to an inverse gravity force to eliminate the effects of gravity on the reference configuration obtained from segmented pre-operative data, and then subjected to combined gravity and insufflation pressure loading. While the errors on the abdominal wall surfaces obtained with this setup were satisfactory, for the abdominal viscera the errors were noticeably higher with a concentration of higher errors near the liver, which they attributed to the gas inside the viscera.

Other structures of the liver have also been considered for independent modelling. Haouchine et al. [54] proposed the use of a heterogeneous liver model. This heterogeneous model consisted of a standard volumetric, homogeneous model for the liver parenchyma that was coupled to a separate, stiffer model for the vasculature. In their phantom experiments this model yielded sub-surface registration errors less than half of those obtained with a homogeneous liver model alone, at most evaluation time points. However, this evaluation was based on a computer model using the exact material parameters of the parenchyma and the vessels of the phantom, and they also reported a stronger effect on the registration error of an incoporation of a feature quality metric on the tracked points used to constrain the model. Bosman et al. [20] looked at the incorporation of ligaments in surgical simulations. They noticed significant differences in the displacement result of gas-insufflation and tool interaction simulations depending on whether the falciform ligament was simulated through zero-displacement boundary conditions or as an independent elastic structure. The influence on the final result of the ligament model was found to be on par with that of the material parameters chosen for the liver parenchyma.

The choice of realistic constitutive models for biomechanical simulation of in-vivo tissue has been the subject of considerable research, experimental work, and investment. Carter et al. [23] provide an overview and history of the constitutive models employed for some of the most common tasks for which a biomechanical simulation has been employed in surgical guidance, specifically they also provide references for in-vivo measurements of liver tissue. As

is unfortunately frequently the case, the reported values vary greatly, with most researchers having found a greater compliance for healthy liver tissue than diseased tissue, while others report the opposite [23]. Generally, a Young's modulus in the 10s to low 100s of kPa and near-incompressibility can be assumed to be sensible values for the human in-vivo liver [23]. Roan [118] also specifically looked at the stiffness of the Glisson capsule, which has a stiffness typically measured in MPa and whose contribution has been reported to increase the measured stiffness of the organ up to threefold. These effects are compounded in diseased livers, where the capsule is often thickened. If the stiffness of only the in-vivo liver parenchyma is considered, a single-digit value or a value in the low 10s of kPa is probably a more realistic value [118].

While choosing the right material model is important in surgical training applications [2], where realistic haptic feedback matters for the realism of the simulation, their importance in image guidance is more questionable. Wittek et al. [151] went as far as publishing a paper with the contentious title "On the unimportance of constitutive models in computing brain deformation for image-guided surgery", where they argued that biomechanical models driven by known displacements can make do with the simplest of geometrically non-linear constitutive models, isotropic neo-Hookean elasticity. Their reasoning was that viscous effects are negligible due to the small strain rates (speed of deformation) involved in surgery, and that, with known surface displacements, the incompressibility of the material is sufficient to push/pull all subsurface structures into the right place. They also evaluated linear elasticity, and noticed unrealistic behaviour due to its inability to accurately model rotational rigid body motion. Han et al. [51], for their application where material parameters due to the bodyforces involved are more important, circumvented this problem by optimising the model to fit the input data.

By far the majority of algorithms get their boundary conditions from a combination of prior knowledge and geometric correspondence finding algorithms such as those presented in Section 2.3. A soft-tissue biomechanics algorithm that itself takes care of the solving of the correspondence problem via a physical model, instead of getting a set of input displacements, was developed by Suwelack et al. [133] for image-guidance of laparoscopic liver surgery. Their method assigned to both the recovered intra-operative surface and the visible part of the surface of a pre-operatively solid-meshed liver segmentation an electric charge, such that the solid mesh would be attracted by the intra-operative surface. The electro-static forces were directly integrated in the standard solid-dynamics equilibirum equation as force boundary conditions. While algorithmically succinct and computationally efficient, the authors noted a tendency of their method to, similar to ICP-like methods, converge to local minima, if not properly initialised. The liver was simulated with a homogeneous, co-rotational formulation of linear elasticity, which is commonly considered a good choice for the efficient simulation of large displacements, but not so much for large deformations. Their physical experiments subjected a liver phantom containing tracked markers to deformation by an indenter. The most interesting results from those experiments were based on a partial surface recovery from stereo

video for which they obtained a mean error of 8.7mm. Lower errors were achieved with an extraction of the surface of the deformed phantom from CT scans.

In the future, especially with feature-tracking based image-guidance algorithms, one may also want to consider including the effects of the interaction between the liver and the diaphragm due to breathing motion which at this point in time is mostly neglected since the liver tends to fall away from the diaphragm during insufflation. Pratt [112] presented a TLED-based method suitable for recovering unknown forces from measured displacements. This force-based approach allows for an easy combination of simulated breathing motion with external forces, displacement constraints, and/or contact forces.

## 2.6 Presentation to the Surgeon

How the data is presented to the surgeon is of great importance for the usability of an image-guidance system. A good overview of augmented-reality display techniques that can overlay updated segmentations on the video feed or reality itself, as is the case with video projections onto the patient's body, can be found in Nicolau et al. [106], and a wider, but slightly older survey can be found in Baumhauer et al. [13].

## 2.7 Conclusions

The liver consists of very soft and deformable tissue, and therefore primarily the insufflation of the patient abdomen, required to create the operating space in MIS liver resection, leads to large deformation of the organ. Thus, unlike in neurological procedures, a simple rigid alignment of the pre-operative plan with the intra-operative patient position via landmarks is insufficient for image guidance purposes. The best results in terms of the reconstruction of the intra-operative configuration of the liver can be expected with volumetric intra-operative imaging, such as, intra-operative MRI or CT, but these modalities require expensive hardware and pose radiation or magnetic hazards. Their use is also disruptive to the surgical workflow. A combination of instrument tracking and routinely employed intra-operative imaging modalities, laparascopic video and LUS, with algorithmic processing to reconcile intra-operative with pre-operative imaging data, provides a less intrusive, more cost-effective alternative. This relatively inexpensive hardware can be leveraged via surface reconstruction and intra-operative vessel segmentation, respectively, in conjunction with point-cloud registration algorithms, by updating high-quality, segmented pre-operative image data to the intra-operative setting. However, this approach also requires the availablilty of facilities for extrapolation of tissue displacements from the reconstructed, registered surface patches/vessel branches to the rest of the organ parenchyma, and, due to point-cloud registration algorithms' susceptibility to convergence to local minima, a good initial guess for the intra-operative liver configuration. These problems are what is primarily addressed in the subsequent chapters of this thesis, covering my work on FEM biomechanical simulation and insufflation deformation estimation.

## Chapter 3

# Estimation of Liver Deformation Due to Pneumoperitoneum

## 3.1 Introduction

The work that is the topic of this chapter was carried out in collaboration with the Smart Liver Surgery project (HICF-T4-317). It was published in a condensed form in the proceedings of the 2015 MICCAI conference under the title "Database-Based Estimation of Liver Deformation Under Pneumoperitoneum for Surgical Image-Guidance and Simulation" [71]. My involvement in that project led to me being a co-author on the following further publications: Song et al.: "Locally Rigid, Vessel Based Registration for Laparoscopic Liver Surgery" [127] and Thompson et al.: "Accuracy Validation of an Image Guided Laparoscopy System for Liver Resection" [146].

The pipeline presented in this chapter aims to provide a better approximation of the liver configuration after gas insufflation of the patient abdomen than that visible in the pre-operative images. The main motivation for having such an estimate is, as mentioned in Chpt. 2, the tendency of many of the mesh registration algorithms proposed for the intra-operative updating of the surgical plan to converge to local minima of the cost function, if not properly initialised. Having an automatic algorithm perform this initialisation instead of, e.g., a manual rigid alignment of the pre-operatively meshed liver with a configuration seen in video images is preferable both for accuracy and workflow reasons.

There is rather little literature on algorithms for the tackling of this specific task, with most of that literature reviewed in Chpt. 2.

Work conducted by Bano et al. [10], investigated the possibility of simulating pneumoperitoneum to guide port placement. The method they proposed comprised three independent meshes, representing thoracic and abdominal viscera, the latter including the liver and the abdominal wall. The simulations modelled the effects of the gas pressure and gravity. While it was not their primary objective, they also evaluated their method's ability to predict the position of the abdominal viscera in a pig model, and found errors in terms of mesh vertex distances of 5.7 ± 4.9mm with the larger errors being concentrated near the liver. A complete pipeline, from image segmentation through to visual output to the surgeon, for the task of simulation

of the abdominal wall deformation due to gas insufflation was also presented by Kitasaka et al. [78]. The tissue biomechanical model they employed was a simple mass-spring one to which they directly applied gas-pressure forces. However, none of the simulation parameters were given and no registration error evaluation was performed.

Bosman et al. [20] looked at the incorporation of ligaments in surgical simulations with pneumoperitoneum being one of the simulations they performed. There was no ground truth to compare their results to, and all that was presented in the paper was a comparison to a reference simulation in terms of the displacement solution depending on whether the falciform ligament was simulated as an elastic structure or its presence was simulated by constraining the nodes on the liver surface adjacent to the ligament with displacement boundary conditions. However, the paper does illustrate how difficult it is to generate a model that correctly takes into account the effects of surrounding anatomy on the liver configuration under pneumoperitoneum.

Atlas-based methods have been proposed for a number of tasks similar to the estimation of the deformation due to pneumoperitoneum: Clements et al. [32] presented a method for the compensation of liver deformation in open hepatic surgery. The atlas used in the method consisted of solutions for a range of loads applied to a computational model of the patient's liver. Their deformation estimates were computed as weighted averages of model solution displacement fields with weights (one per model) determined iteratively by measuring the atlas solutions' distances to intra-operatively acquired surface data. The results they presented, while good, were obtained with only simple deformations being applied to a physical liver phantom. A very similar algorithm to that of Clements et al. was used by Dumpuri et al. [39] for neurosurgical IGS to extrapolate sparse brain-shift measurements to the whole of the brain volume. Plantefeve et al. [111] developed an atlas-based transfer of FEM boundary conditions (BCs). The algorithm mapped idealised BCs, such as points constrained by ligaments, onto a patient liver after a physics and feature-based registration to the atlas. The atlas BCs were computed with Principal Component Analysis performed on a large number of input segmentations of ligaments, blood vessels, and similar structures. They obtained a good agreement in the location of mapped anatomy compared to ground-truth segmentations, and also between results obtained with simulations where the BCs were given by the ground-truth constraint obtained from segmented anatomy and simulations with BCs based on mapped anatomy. However, their method was limited to the mapping of BCs that arise from easily identifiable anatomy, and no validation involving the multitude of interactions determining the liver deformation in real surgery was performed.

The algorithm presented in this chapter is based on the assumption that accurately modelling the interactions occurring during insufflation between the liver and the surrounding tissues is too challenging, since the latter largely consist of tissues that due to their softness and mobility cannot be accurately segmented, meshed, and modelled. It is therefore based on an approach closer related to the aforementioned atlas-based methods. The proposed system

is built on a database of known pre-operative to intra-operative configuration displacements, determined for a series of training subjects whose intra-operative abdominal organ configuration was imaged with, e.g., interventional CT or another gold-standard intra-operative image guidance method. Once such a database is built, it can be used in operating theatres where intra-operative 3D imaging is not available to estimate the displacement for a patient about to undergo a laparoscopic procedure, and for whom only the pre-operative configuration is known. In this chapter it will also be shown how it can compliment the biomechanics-based pneumoperitoneum simulation methods and other progress recently made in the simulation of liver biomechanics, by combining the displacements estimated with the database with a patient-specific biomechanical simulation to incorporate patient-specific BCs and other prior knowledge.

This chapter is organised as follows: The next section (Section 3.2.1) gives a high-level overview over the algorithmic framework. In Section 3.2.2, the data underlying this work are introduced. In Section 3.2.3 and Section 3.2.4, the spectral embeddings and their augmentations used for registration and subject similarity computation are described. Section 3.2.5 explains how the augmented spectral embeddings are used to find mesh point correspondences. Section 3.2.6 presents the full intra-subject registration pipeline used for the computation of the displacements stored in the database. Section 3.2.7 describes the computation of a first estimate of the deformation under pneumoperitoneum. Section 3.2.8 then explains how this first estimate can be refined by means of a biomechanical simulation. Later on, modifications to make the method suitable for large databases are also discussed (Section 3.2.9). Section 3.3 contains a validation of all major components of the algorithm.

## 3.2 Methods

### 3.2.1 Algorithm Overview

The proposed algorithm computes for a subject with an unknown liver configuration under pneumoperitoneum (henceforth referred to as an unseen subject) an estimate for the displacement of its liver during insufflation. The estimate is computed as a weighted average of displacements that have been observed in other subjects (later referred to as training or database subjects) by means of intra-operative 3D imaging and stored in a database. In the simplest case, the database only has to store for the training subjects the original pre-operative liver mesh, an augmented spectral embedding of that mesh, the displacements, and the insufflation pressure. The spectral embedding is used for registration with unseen-subject livers. The original liver mesh and the gas pressure are used in the computation of the weights used in the averaging process.

A high-level overview of the algorithm is given in the flowchart in Fig. 3.1. It comprises two distinct processing pipelines: 1) A pipeline processing training data consisting of CT images of the pre-operative and the intra-operative configuration of the abdomen, and the corresponding

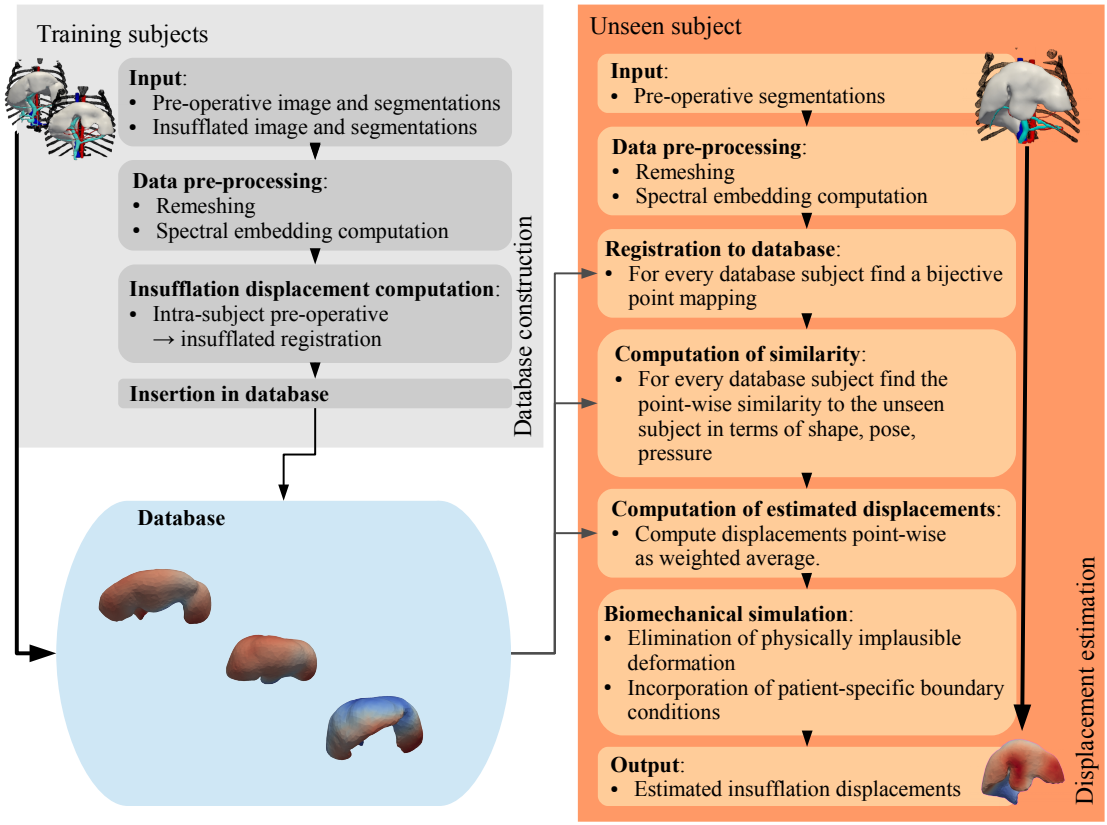



Figure 3.1: Flowchart overview over the proposed deformation estimation pipeline. Left: processing of pre-operative/intra-operative training configurations. Right: processing of an unseen subject for displacement estimation.

anatomical segmentations. This pipeline consists of the computation of the augmented spectral embeddings used for registration, an intra-subject registration for the determination of the liver displacements from pre-operative to intra-operative, and the insertion of the data in the database. 2) A pipeline for the estimation of the deformation due to pneumoperitoneum for unseen subjects. This pipeline only requires a pre-operative segmentation as input. It too contains a stage computing an augmented spectral embedding that allows for a registration of the unseen liver to those stored in the database. A first set of displacements is computed by point-wise averaging the database displacements based on local measures of similarity between the unseen liver and those of the database, based on correspondences found in an inter-subject registration step. The final displacement prediction is obtained from a biomechanical simulation that incorporates patient-specific BCs and is driven by the first displacement estimate.

### 3.2.2 Image Data and Segmentations

For this work I had segmented, contrast-enhanced CT images of the pre-operative and insufflated anatomy of 5 pigs (subsequently referred to as P1, P2, P3, P4, and P5) at my disposal courtesy of the Smart Liver Surgery team: Drs. M. Clarkson, S. Thompson, J. Totz, Y. Song,

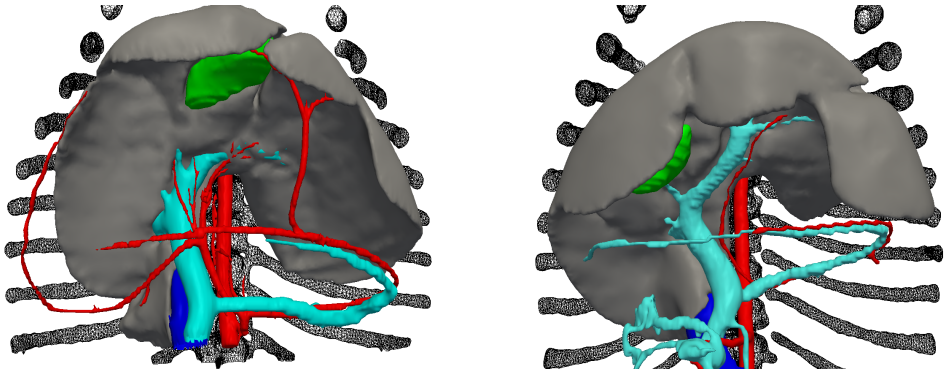


Figure 3.2: Examples of two pig-abdomen segmentations, un-insufflated/pre-operative state: rib-cage (black wiremesh), liver (grey), gall bladder (green), arteries (red), hepatic vein (blue), portal vein (turquoise)

K. Gurusamy, and Profs. B. Davidson and D. Hawkes. The animals were anaesthetised with isoflurane and insufflated for the experiments, and a total of 4–5 ports were put in place (1 umbilical, 1 epigastric, 1 left upper quadrant, and 1 or 2 right upper quadrant ports). The CT images were acquired at full exhale. The segmentations were done by Visible Patient[1], and all included the liver, the hepatic artery, the hepatic vein, the portal vein tree, the gall bladder, and the rib cage (Fig. 3.2). All pigs were scanned with the same scanner and protocol. A more detailed description of the image data is given in Section 3.3.

As can be seen in Fig. 3.2 showing some of the segmentations, inter-subject variability in terms of shape and size of the liver is large in pigs. Nevertheless, as with human livers, there are sizeable surface patches devoid of significant geometrical features, complicating the registration.

The segmentations were provided as VTK surface meshes, with liver segmentation meshes initially containing about 200,000 facets. The segmentations are downsampled to about 2000 facets with a custom pipeline to save time doing the registration, eliminate small inconsistencies in the segmentations, and make them more suitable for finite-element simulation. The pipeline consists of a back-conversion of meshes into masks, the masks are then blurred and an intensity threshold is applied. This relatively crude procedure proved the most effective, out of a number of options evaluated, for closing small cavities in the segmentation meshes without altering the general surface outline. The binary mask resulting from this procedure is then used for meshing. The pre-processing pipeline was implemented with ITK[2] and CGAL[3].

### 3.2.3 Spectral Embeddings

The use of spectral embeddings for mesh registration and mapping of attributes between subjects is motivated by the fact that the relative size of pig liver lobes can vary considerably.

[1] `http://www.visiblepatient.com/en/service/`
[2] Insight Segmentation and Registration Toolkit v. 4.4: `http://www.itk.org`
[3] Computational Geometry Algorithms Library v. 4.2: `http://www.cgal.org`

The livers are also highly deformable and their lobes can move independently to a large extent. Hence, establishing point correspondences based on the topology of the liver surface rather than its spatial configuration is highly advantageous for this application.

The basic form of the spectral embeddings is based on the work of Jain et al. [66]. They consist of the dominant eigenmodes of a dense, symmetric affinity matrix $A$ whose entries are invariant with respect to scale and bending, and that is computed from the mutual geodesic distances between all nodes, denoted with $D$:

$$A_{ij} = \exp(-\alpha D_{ij}^2),\ \alpha = 4/\max_{kl} D_{kl}^2 \tag{3.1}$$

Since the geodesic-distance matrix is used in a number of other places in the algorithm, it is advisable to use a high-quality approximation, such as that produced by the algorithm of Kimmel and Sethian [76].

After the eigen factorisation of $A$, the spectral embedding consisting of the $k$ most significant eigenmodes is given by

$$\boldsymbol{S} = \left[ \sqrt{\lambda_1}\boldsymbol{V}^{(1)} | \sqrt{\lambda_2}\boldsymbol{V}^{(2)} | .. | \sqrt{\lambda_k}\boldsymbol{V}^{(k)} \right],\ A\boldsymbol{V}^{(i)} = \lambda_i \boldsymbol{V}^{(i)},\ \lambda_1 > \lambda_2 > ... \tag{3.2}$$

To compensate for the eigenmode ordering and sign ambiguity, the *eigenmode non-robustness*, a greedy algorithm similar to that proposed by Jain et al. is employed. The algorithm works by finding for the $i$-th eigenmode $\boldsymbol{V}^{(i)}\sqrt{\lambda_i}$ the optimal position and sign in an intermediate source-mesh spectral embedding consisting of the $1, .., i-1$ most significant eigenmodes. It does this by evaluating for all position and sign combinations the sum of the distances of all nodes of the source spectral embedding to the nearest neighbour node in the target spectral embedding. The projections of some of the anatomical structures adjacent to the liver are used to provide some baseline correspondences for the distance evaluation and safeguard against a mis-ordering, similar to how Jain et al. propose to use *anchor points*. The reordering process is more formally described in Alg. 3.1. The process by which the projections are computed is given in Section 3.2.4.

In Alg. 3.1, the sub-routine `FindOptimumSign`($\hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}}, j$) toggles the sign of the $j$-th column in the input spectral embedding $\hat{\boldsymbol{X}}$, and then determines which sign yields the smaller accumulated distance for the nearest neighbours in the two augmented spectral embeddings. It also returns that minimum distance.

A faster alternative to using Jain et al.'s embeddings and non-rigid point-cloud registration is the coupled graph-Laplacian algorithm of Lombaert et al. [89]. It constructs one large linear system comprising affinities of both meshes to be registered. Since the method only computes affinities of adjacent nodes, the graph-Laplacian matrix, while significantly larger, is very sparse, and can thus be factorised efficiently using sparse spectral matrix decomposition algorithms, and point-correspondences are obtained without needing a separate registration step. Unfortunately, it proved unsuitable for the purposes described in this chapter; most likely because it was not able to cope with the sometimes visible topological mesh differences,

**Algorithm 3.1** Eigenmode non-robustness compensation algorithm

$\boldsymbol{S}_{\text{target}} \leftarrow$ Target mesh's spectral embedding
$\boldsymbol{V}, \boldsymbol{\Lambda} \leftarrow \text{evd}(A)$ {Eigen decomposition of the source mesh's affinity matrix (Section 3.2.3)}
$\boldsymbol{A}_{\text{source}}, \boldsymbol{A}_{\text{target}} \leftarrow$ anatomical-projection vectors for source and target meshes (Section 3.2.4)
$k \leftarrow$ desired number of eigenmodes in spectral embedding
$(\boldsymbol{S}_{\text{source}}, d) \leftarrow \texttt{FindOptimumSign}\left(\left(\boldsymbol{V}^{(1)}\sqrt{\lambda_1}, \boldsymbol{A}_{\text{source}}\right), \left(\boldsymbol{S}^{(1)}_{\text{target}}, \boldsymbol{A}_{\text{target}}\right), 1\right)$
**for** $i = 2$ **to** $k$ **do**
  $(\boldsymbol{S}_{\text{current}}, d_{\text{min}}) \leftarrow (\boldsymbol{S}_{\text{source}}, \infty)$ {Initialisation for next iteration}
  **for all** $j \in \{2, \cdots, i\}$ **do**
    $\tilde{\boldsymbol{S}} \leftarrow \texttt{InsertColumnAt}\left(\boldsymbol{S}_{\text{source}}, \sqrt{\lambda_i}\boldsymbol{V}^{(i)}, j\right)$
    $(\tilde{\boldsymbol{S}}, d) \leftarrow \texttt{FindOptimumSign}\left(\left(\tilde{\boldsymbol{S}}, \boldsymbol{A}_{\text{source}}\right), \left(\boldsymbol{S}^{(1..i)}_{\text{target}}, \boldsymbol{A}_{\text{target}}\right), j\right)$
    **if** $d < d_{\text{min}}$ **then**
      $(\boldsymbol{S}_{\text{current}}, d_{\text{min}}) \leftarrow (\tilde{\boldsymbol{S}}, d)$
    **end if**
  **end for**
  $\boldsymbol{S}_{\text{source}} \leftarrow \boldsymbol{S}_{\text{current}}$
**end for**

resulting from segmentation errors and image-quality deficiencies. The method also requires that meshes be reasonably well aligned before construction of the matrix to establish the coupling between the graph Laplacians of the meshes, which given the large discrepancies in the physical space appearance of pig livers, can be very difficult.

### 3.2.4 Anatomical Projections

To guide the registration algorithm and provide additional information on top of the liver geometry to the displacement prediction algorithm, the following anatomical structures are projected onto the liver: the centreline of the trunks of the hepatic and portal veins and the hepatic artery, and the gall bladder. The bloodvessel centrelines are extracted with VMTK[4] in a semi-automatic process. The resulting polylines, as well as the entire gall bladder are subsequently projected onto the liver surface by means of Alg. 3.2. The projection value at a liver mesh surface node is inversely proportional to the distance to its nearest neighbour node in the segmentation of the projected structure, and since there is a one-to-one correspondence between spatial mesh nodes and spectral-embedding nodes the projections can be incorporated as additional channels in the spectral embeddings. Fig. 3.3 illustrates the projection of the hepatic vein and gall bladder.

The result of the projection of these 4 anatomical structures is a 4-dimensional feature vector, $\boldsymbol{a}$, for every point in the input liver mesh that is appended to the spectral embeddings, as an augmentation of the latter, similar to how Lombaert et al. [89] incorporated the sulcal depth

[4]Vascular Modeling Toolkit v. 1.2: `http://www.vmtk.org`

**Algorithm 3.2** Algorithm for projection of nearby anatomy onto liver.

$(X, D) \leftarrow$ nodes of liver mesh and corresponding geodesic distances
$Y \leftarrow$ nodes of blood-vessel poly-line/gall bladder mesh
**for** $n \in \{1...\texttt{NumberOfNodes}(X)\}$ **do**
  $j \leftarrow$ index of node in $Y$ closest to $\boldsymbol{x_n}$
  $i \leftarrow$ index of node in $X$ closest to $\boldsymbol{y_j}$
  **if** $i = n$ **then**
    $P_n \leftarrow \|\boldsymbol{x_n} - \boldsymbol{y_j}\|$ {Set distance value iff $\boldsymbol{x_i}, \boldsymbol{y_j}$ are mutually nearest neighbours.}
  **else**
    $P_n \leftarrow \infty$
  **end if**
**end for**
**for** $n \in \{1...\texttt{NumberOfNodes}(X)\}$ **do**
  **for** $i \in \{1...\texttt{NumberOfNodes}(X)\}$ **do**
    $P_i \leftarrow \min\{P_i, P_n + D_{ni}\}$ {Use geodesic distances to get values for non-nearest neighbour nodes}
  **end for**
**end for**
$P \leftarrow \exp(-P^2/(8 \max P^2))$

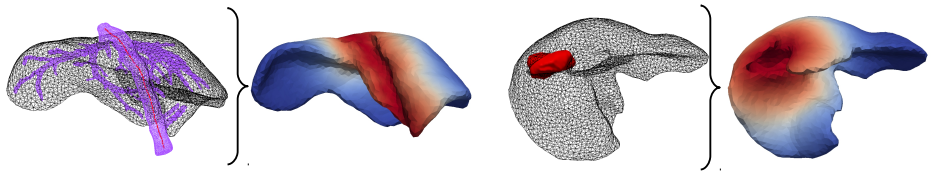


Figure 3.3: Left: projection of the hepatic vein; initial hepatic vein mesh (purple, wiremesh), its centreline (red), the liver surface (black wiremesh), then hepatic-vein projection colour mapped onto the liver surface. Right: initial gall bladder mesh (red, solid), liver surface (black wiremesh), then gall bladder-projection colour mapped onto the liver surface.

at each vertex as an additional component in their brain mesh-derived spectral embeddings.

The finished augmented spectral embeddings consist of feature vectors that take the following form

$$\hat{\boldsymbol{x}} = \begin{pmatrix} \overbrace{\boldsymbol{s}}^{\text{basic spectral embedding}} & , & \overbrace{\boldsymbol{a}}^{\text{anatomical projections}} \end{pmatrix} \tag{3.3}$$

For the spectral embedding component $\boldsymbol{s}$, the $k = 6$ dominant eigenmodes are used. To ensure an equal weighting between these components, the augmentation columns $\boldsymbol{A}$ are rescaled to the mean of the spectral-embedding column value ranges, as follows:

$$\boldsymbol{A} \leftarrow \frac{\text{mean}_i(\max_j \boldsymbol{S}_{ji} - \min_j \boldsymbol{S}_{ji})}{\max \boldsymbol{A}} \boldsymbol{A} \tag{3.4}$$

### 3.2.5 Establishing Point Correspondences and Mapping

Once the augmented spectral embeddings are constructed and consistently ordered, a registration with the Coherent Point Drift[5] (CPD) algorithm [105] is performed. Using CPD instead of thin-plate splines (TPS), as suggested in [66], not only circumvents the inherent mathematical problem with applying TPS that is derived and valid only in a 2D/3D context to higher dimensional data [19], but it also has the advantage of using the entire target surface in the matching and warping process, and not only single closest points. CPD is applied iteratively with alternating source and target point clouds for a fixed number of iterations, as described in Algorithm 3.3, thus ensuring bijectivity of the correspondences. The output of the CPD stage

**Algorithm 3.3** Iterative Coherent Point Drift (CPD)-based registration of spectral embeddings.

$K \leftarrow$ Number of algorithm iterations
$\hat{\boldsymbol{x}} \leftarrow$ Source augmented embedding
$\hat{\boldsymbol{y}} \leftarrow$ Target augmented embedding
$C \leftarrow \{\}$ {Point correspondences}
**for** $k \in 1, .., K$ **do**
  $\hat{\boldsymbol{y}}', C \leftarrow \texttt{CPD}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$
  $\hat{\boldsymbol{x}}', C \leftarrow \texttt{CPD}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$
  **if** $k < K$ **then**
    $\hat{\boldsymbol{x}} \leftarrow (\hat{\boldsymbol{x}}' + (K-1)\hat{\boldsymbol{x}})/K$
    $\hat{\boldsymbol{y}} \leftarrow (\hat{\boldsymbol{y}}' + (K-1)\hat{\boldsymbol{y}})/K$
  **else**
    output $\leftarrow (C, \hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$
  **end if**
**end for**

is a pair of warped spectral embeddings – warped such that they represent the same geometry and only differ in terms of discretisation.

[5] Reference implementation: `https://sites.google.com/site/myronenko/research/cpd`.

For mapping a quantity $Q$, e.g. displacements, from one subject $A$ onto another $B$, after registration and establishment of correspondences, an interpolation in spectral space is employed, as done by Lombaert et al. [89]:

$$\begin{aligned} Q(\hat{\boldsymbol{x}}) &\leftarrow \frac{1}{\sum_{\hat{\boldsymbol{y}} \in {}^{A}\mathcal{N}(\hat{\boldsymbol{x}})} w(\hat{\boldsymbol{y}}, \hat{\boldsymbol{x}})} \sum_{\hat{\boldsymbol{y}} \in {}^{A}\mathcal{N}(\hat{\boldsymbol{x}})} w(\hat{\boldsymbol{y}}, \hat{\boldsymbol{x}}) Q(\hat{\boldsymbol{y}}) \\ w(\hat{\boldsymbol{y}}, \hat{\boldsymbol{x}}) &= \exp\left(-\|\hat{\boldsymbol{y}} - \hat{\boldsymbol{x}}\|^2 / (2\sigma^2)\right) \end{aligned} \tag{3.5}$$

where ${}^{A}\mathcal{N}(\hat{\boldsymbol{x}})$ denotes the neighbourhood of the 3 nodes in the registered and warped spectral embedding of $A$ closest to the point $\hat{\boldsymbol{x}}$ of the spectral embedding of $B$. The variance $\sigma^2$ is computed from the spectral mesh edge lengths.

In the remainder of the chapter the notation $Q(\boldsymbol{x_A})$ is employed to represent a field $Q$ evaluated at the (spatial) position $\boldsymbol{x}$ on the surface mesh $A$. Similarly, $Q(\boldsymbol{\phi_{A2B}}(\boldsymbol{x_A}))$ is used to denote $Q$ evaluated at the position on surface $B$ corresponding to $\boldsymbol{x_A}$, which implies the use of (3.5) and where $\boldsymbol{\phi_{A2B}}$ is the coordinate mapping from $A$ to $B$.

### 3.2.6 Intra-Subject Registration

The proposed algorithm requires a database of known pre to post-insufflation displacements, which in turn necessitates an algorithm for intra-subject registration.

The initial intra-subject registration is performed by computing the spectral point correspondences between the pre- and post-insufflation configuration meshes (Section 3.2.5). This is followed by an intensity-based registration of masked, augmented CT images. The image augmentations consist of the segmentation masks corresponding to the same anatomical structures used for the augmentation of the spectral embeddings, i.e. hepatic vein, portal vein, gall bladder, and hepatic artery. All segmentation-mask images are blended with the original CT image (denoted w/ $\mathtt{Im}$) by setting their foreground value to some unique value $i_{\text{seg}} > \max \mathtt{Im}$ and their background value to $-\infty$, and taking the voxel-wise maximum of the two images. An example of a source-target image pair blended in this manner can be found in Fig. 3.7. A cost function based on Normalised Mutual Information [131] is employed in the subsequent intensity-based registration, therefore the actual value of $i_{\text{seg}}$ does not matter beyond being unique to the structure.

The so augmented source image is then warped to closely match the displacements computed with the spectral point matching, by means of a GPU-accelerated NiftySim [70] (TLED) FEM simulation run on a hexahedral mesh with the same dimensions as the image and soft, fixed material parameters $G = 0.1$, $K = 0.2$, and displacement BCs are imposed at the position of all nodes of the surface of the liver mesh, thus making the simulation very insensitive to the material settings. This intermediate result image is subsequently registered to the augmented target image with the velocity field registration algorithm of NiftyReg [101, 100]. This algorithm is symmetric and produces a forward and a backward result. The resulting transform is composed with the TLED warp by sampling from the backward displacement field produced by NiftyReg at the node positions in a TLED-warped source mesh corresponding to the inter-

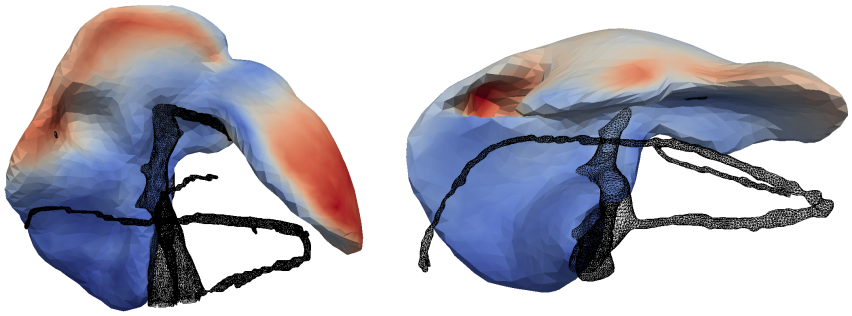


Figure 3.4: Caudal view of the insufflated configuration of two livers with colour-mapped displacements (blue meaning low, red meaning high) and hepatic and portal veins drawn as wiremeshes.

mediate image, and added to the latter displacement values to yield the final insufflated nodal displacements:

$$\boldsymbol{u}_{\text{insuf}}(\boldsymbol{x}) = (\text{I} + \boldsymbol{u}_{\text{TLED}}) \circ (\text{I} + \boldsymbol{u}_{\text{NiftyReg}})(\boldsymbol{x}) = \boldsymbol{u}_{\text{TLED}}(\boldsymbol{x}) + \boldsymbol{u}_{\text{NiftyReg}}(\boldsymbol{x} + \boldsymbol{u}_{\text{TLED}}(\boldsymbol{x})) \tag{3.6}$$

with I denoting the identity transform.

### 3.2.7 Displacement Computation

The displacement computation for an unseen liver requires a registration and known point-wise correspondences to every liver in the database. This is accomplished with the spectral correspondence finding algorithm described in Section 3.2.5. With these mutual point correspondences in place, the displacement field is computed node-wise as a weighted average of the corresponding point displacements found in the database. The similarity metrics providing the weights are the main topic of this section, and are motivated by the three factors deemed dominant in determining the livers motion during insufflation: its shape and anatomical configuration, its spatial pose, and the gas pressure applied in the procedure.

The first similarity metric $s_S$ is given by the similarity of the *unwarped*, augmented spectral embeddings of the two meshes. For the purpose of the metric's evaluation the spectral embedding is treated like any multivariate attribute associated with mesh nodes, that can be mapped from one subject onto another.

$$^{ut}s_S = -\frac{||\hat{\boldsymbol{x}}(\boldsymbol{x_u}) - \hat{\boldsymbol{x}}(\boldsymbol{\phi_{ut}}(\boldsymbol{x_u}))||}{||\hat{\boldsymbol{x}}(\boldsymbol{x_u})||} \tag{3.7}$$

This similarity metric takes into account information about both the shape of the meshes and, thanks to the anatomical projections used for augmentation, nearby anatomy.

As can be seen in Fig. 3.4 showing the displacement field obtained for two of the animal subjects with the intra-subject registration pipeline, presence of major vasculature is correlated with a lower displacement magnitude. Therefore, the usefulness of the anatomical projections extends beyond the standard augmentation of the point sets in point-cloud registration; it can directly help to improve the quality of the pipeline output.

The computation of the similarity for all nodes of the unseen mesh and all training meshes results in a similarity matrix $S_S \in \mathbb{R}^{N\times M}$, where $N$ is the number of nodes, and $M$ is the number of database subjects. The similarity matrix $S_S$, and all subsequent similarity measures are then normalised to $[0, 1]$ via

$$\tilde{S} = \frac{S - \min S}{\max S - \min S} \tag{3.8}$$

The next similarity metric considers a more localised shape similarity and the similarity of the spatial configuration between two corresponding surface patches. This is accomplished by computing subject-to-subject local affine transforms, mapping between the unseen $u$ and the database subject $t$, with the weighted least squares approximation

$$P_{t2u}(\boldsymbol{x_u}) = (X_t^{\mathrm{T}} W X_t)^{-1}(X_t^{\mathrm{T}} W X_u) \tag{3.9}$$

the $K \times 3$ matrix $X_u$ contains a neighbourhood around the unseen mesh's node $\boldsymbol{x_u}$

$$X_u = \begin{pmatrix} \boldsymbol{x}_{\mathcal{N}^1(\boldsymbol{x_u})} - \boldsymbol{x_u} \\ \boldsymbol{x}_{\mathcal{N}^2(\boldsymbol{x_u})} - \boldsymbol{x_u} \\ \vdots \\ \boldsymbol{x}_{\mathcal{N}^K(\boldsymbol{x_u})} - \boldsymbol{x_u} \\ \boldsymbol{n}(\boldsymbol{x_u}) \end{pmatrix} \tag{3.10}$$

The training subject matrix $X_t$ is computed from the correspondences of the points in the neighbourhood $\mathcal{N}(\boldsymbol{x_u})$ and the $\boldsymbol{\phi_{u2t}}(\boldsymbol{x_u})$, and the surface normal $\boldsymbol{n}(\boldsymbol{\phi_{u2t}}(\boldsymbol{x_u}))$. $\mathcal{N}(\boldsymbol{x_u})$ is computed based on geodesic distances and the weighting function $w_{ij} = \exp\left(-\alpha_r \|\boldsymbol{x_i} - \boldsymbol{x_j}\|_g\right)$, that also provides the diagonal weight matrix $W$ in Eq. (3.9), by thresholding at $w = 0.001$. The coefficient $\alpha_r$ is chosen such that nodes outside a set radius ($r$ = 4cm in the experiments) around $\boldsymbol{x_u}$ do not contribute to the matrix $P_{t2u}$. The surface-normal $\boldsymbol{n}(\boldsymbol{x_u})$ carries a weight of 1. This gives rise to the transform similarity:

$${}^{ut}s_X = -\sum_{i,j} |(P_{t2u})_{ij} - \mathrm{I}_{ij}| \tag{3.11}$$

where I is the $3 \times 3$ identity matrix.

The last similarity metric is the similarity of the gas pressures which is given by

$${}^{ut}s_G = -\frac{|p_u - p_t|}{p_u} \tag{3.12}$$

where $p_u$ is the pressure used to insufflate the unseen subject's abdomen, and $p_t$ that used on the database subject.

Finally, the predicted displacements $\boldsymbol{u_p}$ are computed at every node of the output mesh from the following weighted average with the product of the normalised similarities providing the weights:

$$\boldsymbol{u_p}(\boldsymbol{x_u}) \leftarrow \frac{1}{\sum_{\tau\in\{\text{database}\}} {}^{u\tau}\tilde{s}_S \, {}^{u\tau}\tilde{s}_X \, {}^{u\tau}\tilde{s}_G} \sum_{t\in\{\text{database}\}} {}^{ut}\tilde{s}_S \, {}^{ut}\tilde{s}_X \, {}^{ut}\tilde{s}_G \boldsymbol{u}(\boldsymbol{\phi_{u2t}}(\boldsymbol{x_u})) \frac{p_u}{p_t} \tag{3.13}$$

The factor $p_u/p_t$ provides a first-order approximation for the displacement magnitude as a function of pressure, in cases where the insufflation of subject $t$ was performed with a lower or higher pressure than the patient $u$.

A re-orientation of the mapped displacement by means of the Jacobian of the transform $\boldsymbol{\phi_{u2t}}$ was attempted, but at least in these experiments, where all subjects were imaged under identical conditions, did not improve the results.

### 3.2.8 Biomechanical Simulation

The biomechanical FEM simulations are performed with NiftySim and comprise two meshes, a liver and a block representing the dorsal portion of the surrounding viscera and the spine (visible in the middle column of Fig. 3.9). The block is generated by subtraction of the liver segmentation mask from an all-one image the size of the input CT image and unsetting of all voxels anterior to the centre of the liver mesh. Tetrahedral meshing with CGAL is then performed to generate the mesh.

The displacements computed with the procedure described in Section 3.2.7 are not applied directly, but through penalty forces. The gas pressure and gravity forces are not explicitly modelled; all their effects are assumed to be contained in the extrapolated displacements. The forces are applied as external forces with a penalty coefficient that increases linearly with time, and given for a time $t$ in the simulated time interval $[0, T]$ by:

$$\boldsymbol{R}^{(t)} = \frac{kt}{T}\left(\boldsymbol{U_p} - \boldsymbol{U}^{(t)}\right) \tag{3.14}$$

where $k$ is the penalty coefficient, $\boldsymbol{U_p}$ is the vector of the predicted displacements, and $\boldsymbol{U}^{(t)}$ is the FEM displacement solution vector at time $t$. Applying the estimated displacements as penalty forces allows for an easy combination with patient-specfic boundary conditions that are formulated as displacement boundary conditions.

A good place to put a subject-specific constraint in the pig model is the dorsal part of the right lobe where the hepatic vein and the portal vein trunks are roughly parallel. It will however be shown in the experiments that this is not necessary, as the displacement estimate computed with (3.13), in conjunction with a minumum displacement threshold below which zero-displacement hard constraints are applied, already yields a very sensible result for this region of the liver.

To prevent an over-constraining of the simulation by the forces defined through (3.14), one can put a cap on the current time-step penalty coefficient $k_t = kt/T$ that depends on the strain energy:

$$k_t = \begin{cases} k_{\max} t/T, \text{ if } E_{\text{strain}} < E_{\max\text{strain}} \\ k_{t-\Delta t}, \text{ otherwise} \end{cases} \tag{3.15}$$

i.e., the current time-step penalty coefficient at time $t$, $k_t$, is held at the previous time step level if the strain energy in the simulation exceeds a user defined threshold $E_{\max\text{strain}}$.

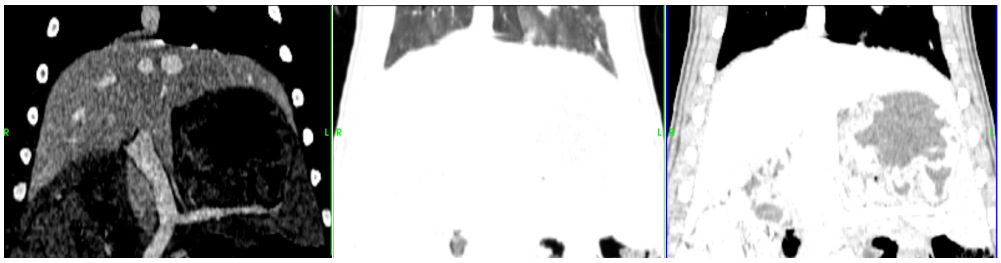


Figure 3.5: Intensity ranges associated with a particular set of material parameters; illustrated with coronal slices through P4. Left: liver $(100, 300)$; centre: lungs $(-\infty, -200)$; right: diaphragm, muscles, etc $(-200, 100)$. Intensities outside the selected range appear white (if above) or black (if below).

An isotropic neo-Hookean constitutive model, with distinct material parameters for the liver, the diaphragm and muscles, lungs, and areas that cannot be reliably assigned to any of the former categories of material, is used. In the case of the surrounding-tissue mesh, the material type is determined based on CT image intensity ranges (Fig. 3.5). The material parameters for these tissues can be optimised with the training data by using the displacements obtained from the intra-subject registration algorithm with (3.14) and anatomical landmark registration errors or another measure of registration quality. The simulation at the end of the displacement-prediction pipeline uses the average of these optimised simulation parameters. The average parameters used in the simulations in the experiment section of this chapter are, with $G$ and $K$ denoting the shear and bulk modulus, respectively: liver: $G = 10.43$kPa, $K = 27.32$kPa; lungs: $G = 2.71$kPa, $K = 2.56$kPa; diaphragm: $G = 4.86$kPa, $K = 12.64$kPa; unassigned/default: $G = 3.27$kPa, $K = 8.20$kPa. The displacements on the dorsal surface of the surrounding tissue mesh are fully constrained with $\boldsymbol{u} = 0$.

### 3.2.9 Accelerations for Large Databases

If a large database of training subjects is available, it is beneficial for performance reasons to introduce a common reference liver mesh, e.g. by computing an average liver geometry using the inter-subject registration pipeline.

In the computation of the spectral embeddings, specifically the reordering of eigenmodes Alg. 3.1, the reference's spectral embedding can take the role of $\boldsymbol{S}_{\text{target}}$, and the eigenmode reordering only has to be done once for all input meshes – training or unseen.

In the displacement computation, if the unseen liver is registered to the reference liver instead, and for every database liver the correspondences in the reference are known, the mapping between the unseen and a training mesh can be obtained through:

$$\boldsymbol{\phi_{u2t}}(\boldsymbol{x_u}) = \boldsymbol{\phi_{r2t}}(\boldsymbol{\phi_{u2r}}(\boldsymbol{x_u})) \tag{3.16}$$

With this method, the displacement computation requires only one inter-subject registration.

| Subject | $p_{gas}$ (mmHg) | Image res. (mm) | Displacement (mm) (max/mean/min) | $N_{landmarks}$ | Remarks |
|---|---|---|---|---|---|
| P1 | 12 | 0.85×0.85×2.5 | 65.8/41.4/6.3 | 8 | Due to the unusually large displacement, a gas pressure of 16mmHg is assumed for this subject. |
| P2 | 12 | 0.85×0.85×2.5 | 51.6/19.4/1.34 | 7 | – |
| P3 | 12 | 1×1×1 | 33.4/15.1/0.2 | 8 | Resampled from a 0.85×0.85×2.5mm volume. Low image-quality due to severe breathing motion artefacts. |
| P4 | 12 | 1×1×1 | 57.2/19/1.3 | 7 | Resampled from a 0.85×0.85×2.5mm volume. |
| P5 | 8 | 0.85×0.85×2.5 | 28.4/13.6/1.0 | 6 | Gas pressure had to be lowered to keep the animal stable. |

Table 3.1: Overview of data sets used in the experiments.

## 3.3 Experiments

Table 3.1 lists the most relevant properties of the data sets used in the subsequent experiments. The stated maximum, mean, and minimum displacements are the max., mean (nodal average), and min. magnitudes of the surface displacements computed with the intra-subject registration pipeline described in Section 3.2.6. Where not explicitly stated otherwise, the validation is performed with vascular landmarks (bifurcations) manually identified in the CT images. NiftyView[6] volume renderings of the pre-operative CT volumes with the validation landmarks are shown in Fig. 3.6.

### 3.3.1 Intra-Subject Registration

The evaluation of the intra-subject registration error compares the vascular landmark TRE for all stages of the intra-subject registration procedure. The RMS TREs are listed in Table 3.2. The first row shows the RMS errors after an alignment of the centre of gravity of the segmentation masks, for reference purposes. The images produced at the various stages of the pipeline for P2 are shown in Fig. 3.7. On standard consumer hardware (Intel Core i7-2600k, nVidia GeForce GTX 680), the intrasubject registration pipeline takes 10-15 mins to complete.

Table 3.3 shows the mean distance between the target and the warped source mesh before and after the image registration step. The values are computed from nearest neighbour nodes, as determined with a VTK k-D tree. Therefore, there is an intrinsic over-estimation of the surface distance proportional to the average surface facet diameter of about 4mm.

The TREs in Table 3.2 indicate that the surface registration handles large deformations well – this is especially evident in the case of P1, but to achieve the best possible registration error in most cases an intensity-based image registration step is required. Part of this can likely be attributed to the fact that the surface registration does not provide any displacements in the

[6] http://cmictig.cs.ucl.ac.uk/research/software/26-niftyview

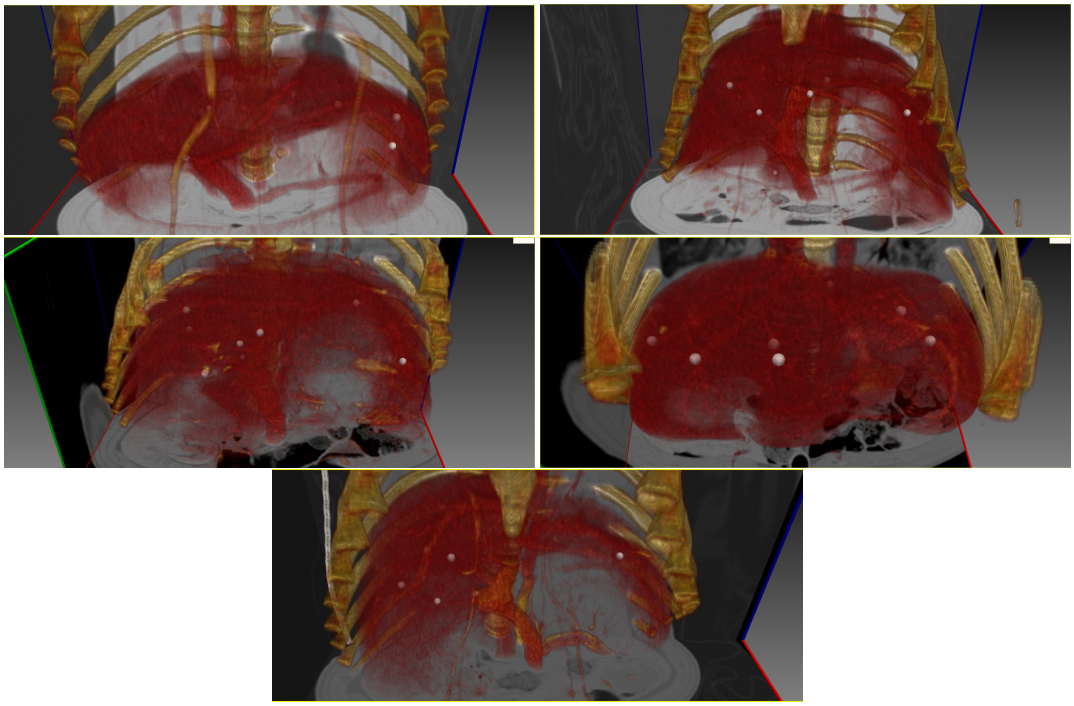


Figure 3.6: Volume renderings of the original pre-operative CT volumes (anterior/cranial views) with the validation landmarks drawn as spheres with a 3mm radius. Top row (left to right): P1, P2; centre row: P3, P4; bottom row: P5

| | P1 | P2 | P3 | P4 | P5 |
|---|---|---|---|---|---|
| RMS TRE rigid alignment (mm) | 13.83 | 13.61 | 16.27 | 12.53 | 9.69 |
| RMS TRE mesh registration (mm) | 8.00 | 7.65 | 11.31 | 9.54 | 4.29 |
| RMS TRE image unaugmented (mm) | 6.14 | 5.34 | 12.03 | 6.67 | 4.28 |
| RMS TRE image augmented (mm) | 3.88 | 3.52 | 12.04 | 3.80 | 2.96 |

Table 3.2: Intra-subject, pre-operative to insufflated configuration registration TREs for: rigid alignment (reference only), surface mesh-registration only, full pipeline without image augmentations, and full pipeline including registration of augmented images.

| Subject | Distance before | Distance after |
|---|---|---|
| P1 | 2.95 | 6.37 |
| P2 | 3.15 | 7.97 |
| P4 | 2.50 | 9.56 |
| P5 | 3.55 | 3.96 |

Table 3.3: Mean mesh distances between post-insufflation and registered pre-insufflation meshes, before and after the intensity-based registration. Values in mm.

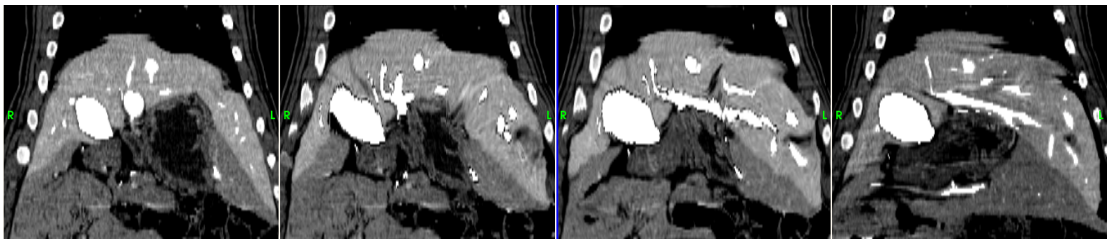


Figure 3.7: The CT images corresponding to the different stages of the intra-subject registration pipeline: coronal slices through images of P2; identical spatial coordinates. Left to right: enhanced source image; source image warped with surface-registration result; result of intensity-based registration; enhanced target image. The image registration is limited to regions of the volume associated with the liver by masking.

liver parenchyma where the landmarks are located, and TPS, which does not use any image or mesh-geometry information, being employed to obtain those displacements. In all cases but P3, whose images are severely degraded by breathing motion artefacts, the RMS error obtained after the image registration is below 5mm. In the case of P3, the image registration in most instances just failed, or if it converged to a result, it was as in Table 3.2, inferior to the surface registration result. Adding the segmentation masks to the registration yields an improvement in the RMS TRE everywhere except on the P3 data. It is also worth noting that the vascular landmarks were selected based on image intensities alone, except in the case of P3 where the contrast was so poor due to motion artefacts that some of the landmarks had to be selected via the vascular segmentations.

The error after the surface registration based on the augmented spectral embeddings is partly caused by inaccuracies in the meshes, in turn partly due to erosion caused by the pre-processing described in Section 3.2.2, and partly due to segmentation errors and inconsistencies. This is exacerbated by the surface registration algorithm being based on the assumption that there is a unique corresponding point in the target spectral embedding for every point in the source embedding, and therefore a correspondence for every point in the spatial source mesh. This is the safest possible assumption, but it is not entirely true as the results in Table 3.3 show, where the distance between the source and the target surface meshes before and after intensity-based image registration are listed. In this table, one can find the surface distance increasing in every instance after running the typically more reliable intensity-based registration of the pre-operative configuration to the intra-operative one. Comparing the results in Table 3.3 and the sub-surface landmark results in Table 3.2, one can conclude that, due to the meshing related issues, the mesh-distance that is frequently used in the literature is a poor evaluation metric, at least for this registration pipeline.

| Target \ Source | P2 | P3 | P4 | P5 |
|---|---|---|---|---|
| P1 | 3.72 | 6.57 | 6.64 | 6.35 |
| P2 | - | 6.53 | 7.64 | 10.76 |
| P3 | - | - | 3.02 | 10.00 |
| P4 | - | - | - | 11.60 |

Table 3.4: Mean portal-vein centreline distances after inter-subject registration. Distances in mm.

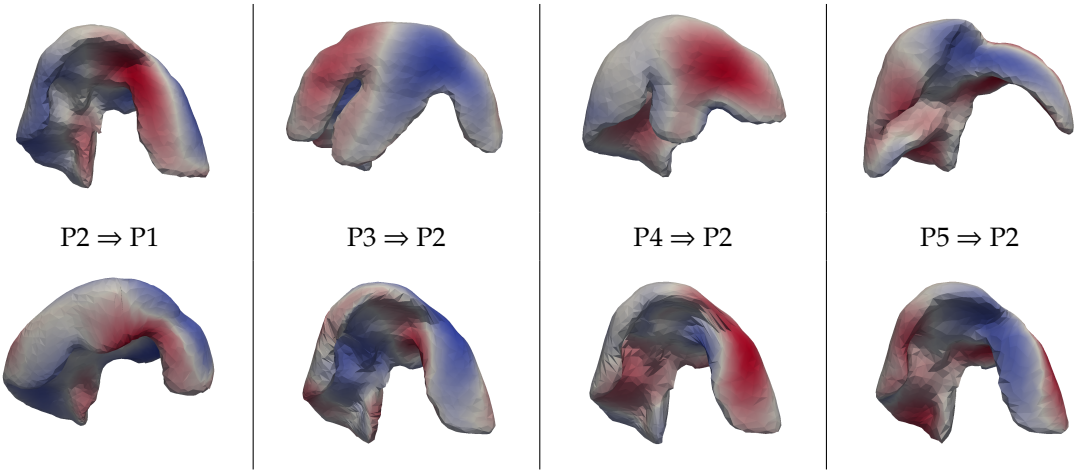



Figure 3.8: Inter-subject registration results. Top row: meshes pre-warp. Bottom row: meshes post-warp. Third eigenvector of source-mesh affinity matrix used for colouring and highlighting of correspondences.

### 3.3.2 Inter-Subject Registration

Since it is very difficult to find reliable landmarks for validation of the inter-subject registration pipeline, the following validation is based on the distance between the last 5cm of the portal vein trunk centreline after registration. The mean distances between the source centreline segment $\mathrm{pv}_s$ and the target centreline $\mathrm{pv}_t$ as given by Eq. (3.17) can be found in Table 3.4. Fig. 3.8 shows the pre and post-warp meshes for some of the target-source pairs in Table 3.4. The registrations took between 55s and 1m48s.

$$\bar{d} = \frac{1}{\int_{\mathrm{pv}_s} d\boldsymbol{x}} \int_{\mathrm{pv}_s} \min_{\boldsymbol{y} \in \mathrm{pv}_t} ||\boldsymbol{x} - \boldsymbol{y}|| d\boldsymbol{x} \tag{3.17}$$

On average, the errors in Table 3.4 are below 1cm which, for a sub-surface structure, is very acceptable considering the registration is purely based on surface information. Further, it can be seen that the higher errors are linked to subject P5, suggesting that the error in the topology of its mesh, i.e., the fusing of two right lobes resulting in one T-shaped lobe that can be seen in the top-right corner of Fig. 3.8, throws off the spectral point-correspondence finding algorithm.

| Subject | RMS TRE after computation of surface displacements | RMS TRE after biomechanical simulation $E_{\max\,\text{strain}} = 1\text{MJ}$ $k_{\max} = 200\text{N/m}$ | RMS TRE after biomechanical simulation $E_{\max\,\text{strain}} = 0.5\text{MJ}$ $\boldsymbol{u}_{\min} = 5\text{mm}$ $k_{\max} = 100\text{N/m}$ | RMS TRE after biomechanical simulation $E_{\max\,\text{strain}} = 1\text{MJ}$ $\boldsymbol{u}_{\min} = 5\text{mm}$ $k_{\max} = 200\text{N/m}$ |
|---|---|---|---|---|
| P1 | 37.57 | 35.31 | 34.46 | 34.98 |
| P2 | 7.73 | 8.90 | 9.93 | 8.89 |
| P3 | 13.89 | 11.64 | 12.10 | 11.47 |
| P4 | 8.61 | 10.41 | 11.47 | 10.53 |
| P5 | 6.03 | 6.20 | 6.59 | 5.82 |

Table 3.5: RMS TREs after application of the predicted displacements. All values in mm.

### 3.3.3 Displacement Computation

Table 3.5 lists the RMS TREs obtained with the predicted displacements on the same landmarks as used for validation of the intra-subject registration pipeline. The first data column in Table 3.5 contains the RMS TREs obtained with the pipeline described in 3.2.7; the second, third and fourth column list the errors after biomechanical simulations with varying parameters (Section 3.2.8). In the second column, no hard constraints were imposed, i.e., the estimated displacements were imposed only as penalty forces. In the two subsequent columns, displacements smaller than a threshold (5mm) were imposed as zero-displacement hard constraints. For all output subjects, the data from all pigs except the one for which the displacements are computed are used, i.e., the displacements for P1 are computed from P2, P3, P4, and P5, etc. The initial and final configurations of the biomechanical simulations are shown in Fig. 3.9.

The full pipeline results in Table 3.5 show that on all subjects except P1 the RMS TRE is comparable to the registration error after the intra-subject surface registration and significantly better than what is obtained with a rigid alignment of the pre- and post-insufflation meshes (Table 3.2). P1 is an outlier, not only in terms of its displacement magnitude, but also due to lobes being fused together in two places and the liver being slightly rotated about the ventro-dorsal axis (the right picture in Fig. 3.2 shows P1's segmentations). Since there is no match in the database for such a liver, the method can only fail.

The biomechanical simulation seems to provide the greatest benefit where the weighted average displacement computed with (3.13) works poorly, in particular on subject P1. In those cases, a less constraining simulation appears to be desirable. It can also be observed by comparing the last two columns that the biomechanical simulation is not very sensitive to the settings chosen for the maximum penalty coefficient and the strain energy threshold.

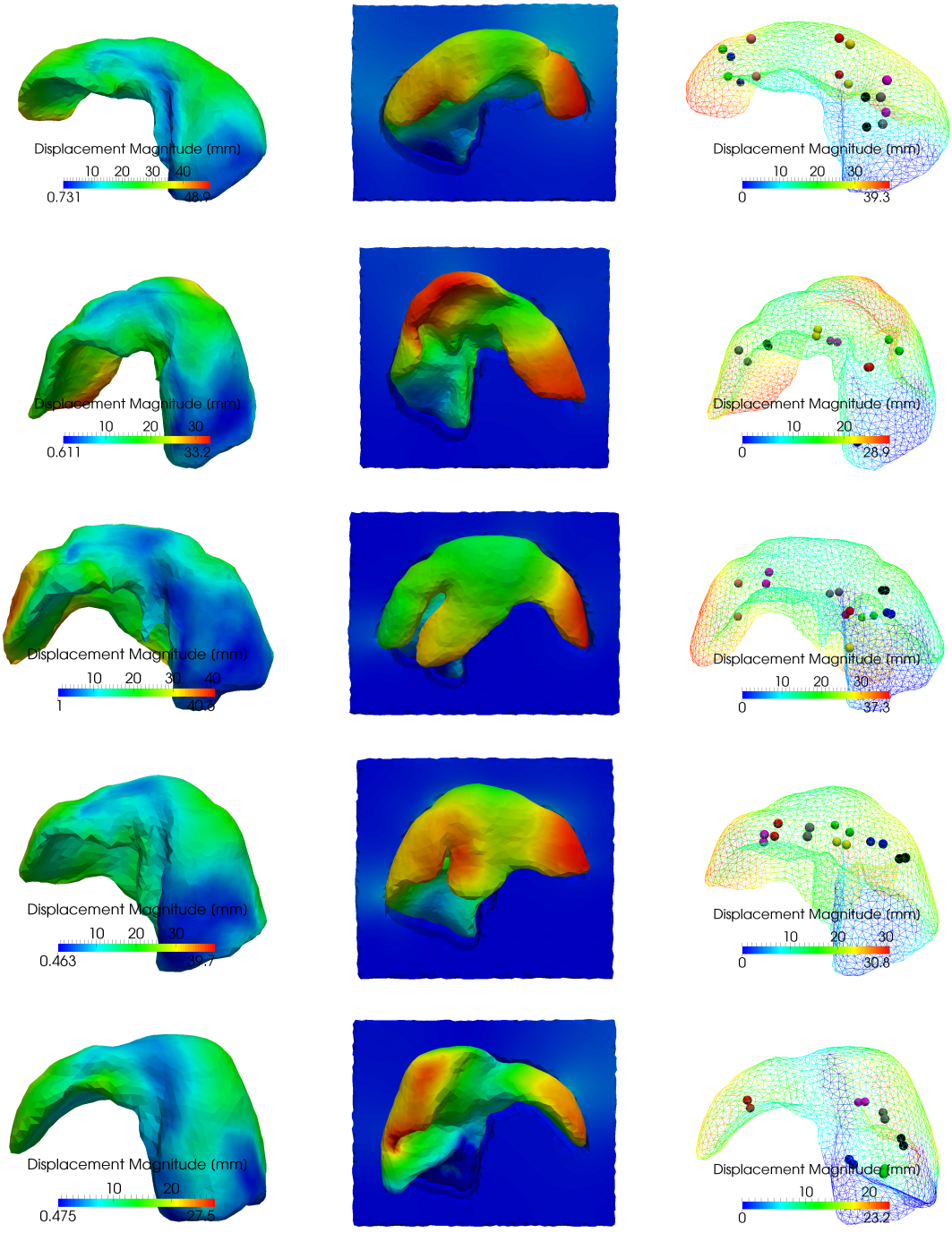



Figure 3.9: Simulation results obtained with $E_{\text{max strain}} = 1\text{MJ}$, $k_{\text{max}} = 200\text{N/m}$, and $\boldsymbol{u}_{\text{min}} = 5\text{mm}$ (top to bottom: P1, .., P5). Left column: Weighted mean surface solution, dorsal view. Centre column: Biomechanical simulation final configuration, ventral view. Right column: Simulation final configuration dorsal view with validation landmark pairs (source and target shown as spheres with same colour).

## 3.4 Conclusions

In this chapter, I have presented a new method that can approximate the effects on the liver of the complex mechanical interactions of the abdominal viscera occurring during insufflation for laparoscopic surgery. The displacements are computed node-wise as the weighted average of displacements observed in training subjects, where the insufflated configuration was captured with CT imaging. The weights are provided by custom similarity metrics comparing unseen and training data based on the anatomical and geometrical configuration and gas pressure. The pipeline also incorporates novel projections of nearby anatomy onto the liver surface, whose purpose is to guide the mesh registration and to improve the displacement extrapolation process by incorporating anatomical information beyond the basic liver geometry. Further, the projections solve the eigenmode non-robustness problem encountered in the construction of spectral embeddings in a mostly automatic way, without having to resort to manually selected anchor points.

Without using any intra-operatively acquired data for the unseen subject whatsoever and with a very small database of training subjects, it achieves an error that is comparable to what has been reported for methods relying on sophisticated biomechanical modelling with a 5.82-11.47mm RMS TRE on subsurface landmarks, on test cases that are representative of the data available in the database. The proposed method has some disadvantages handling outliers, though. Considering that the method can be easily combined with more advanced biomechanical modelling, the current implementation likely only hints at its true potential. Especially when considering applications to human livers where inter-patient variability is not as pronounced as in pigs and consistent segmentation is simpler.

In Section 3.3.1, it was also shown that the intra-subject registration component, when used on its own, can provide sub-surface accuracy in the range of 3-4mm. It could be used, e.g., to register pre-operatively acquired image data to images acquired with intra-operative 3D modalities.

Chapter 4

# NiftySim: A GPU-based Nonlinear Finite Element Package for Simulation of Soft-Tissue Biomechanics

## 4.1 Introduction

This chapter is derived from a paper titled "NiftySim: A GPU-based nonlinear finite element package for simulation of soft-tissue biomechanics". It was written jointly by Dr. Zeike A. Taylor and me, and published in the International Journal of Computer Assisted Radiology and Surgery in 2014 (ref. [70]). It is primarily a software paper whose main purpose is to explain the internal structure of the NiftySim FEM solver package and how it links to the FEM mathematics of the Total Lagrangian Explicit Dynamics (TLED) framework. The sections on Reduced Order Modelling that I have only had minimal involvement in maintaining have been excluded, as has a large part of the paper looking in detail at applications of the solver in published research. Further, since it is the topic of the next chapter, sections related to the contact modelling pipeline have been significantly shortened.

The architecture underlying the current NiftySim was originally implemented in 2008 and later publically released as open-source software via Sourceforge[1] by Dr. Taylor. The day-to-day maintenance of the software was my job between early 2010 and spring of 2016. Aside from the contact modelling pipeline and the membrane/shell element implementations mentioned in the introduction, some of my major contributions to the software package include: Various useability improvements to the XML format used for simulation description, such as definition of boundary conditions via geometrical criteria, import/export of geometry from VTK/MSH files, simplified element set definitions, implementation of new ordinary differential equation (ODE) solver classes for the time ODE and addition of explicit Newmark time integration, implementation of surface traction as a boundary condition type along with its parent class that is shared with pressure BCs, `tledSurfaceConstraint`, CPU parallel solvers, and sub-models

[1] `http://sourceforge.net/projects/niftysim/`

that allow for a recursive definition of simulations. I also introduced a number of software engineering best practices into the development process, such as Doxygen API documentation and unit testing.

My work on the NiftySim software package and involvement in the development of the NifTK framework, that NiftySim is a part of, also led to my being co-author on the journal article Clarkson et al.: "The NifTK Software Platform for Image Guided Interventions" [30], and the conference paper, Tan et al.: "Registration of Automated 3D Breast Ultrasound Views" [135].

The NiftySim toolkit's key feature is its use of GPU (Graphics Processing Unit) based execution, which allows it to outperform equivalent CPU (Central Processing Unit) based implementations by more than an order of magnitude, and commercial packages by significantly more again [138, 53]. While the solver may be used for analysis of any solid materials, it has been designed and optimised for simulation of soft tissues. Its development was motivated by the growing need for robust soft tissue modelling capabilities in medical imaging and surgical simulation applications, and in particular in time-critical applications. Examples of early applications, some of which use algorithms that can be considered precursors to TLED that underlies NiftySim, include, interactive simulation systems where real-time computation was required [34, 94, 134], and intra-operative image registration and image-guidance systems [23, 22, 42] for which rapid, if not real-time, computation was necessary.

NiftySim was developed around the Total Lagrangian Explicit Dynamic FE algorithm. An important feature of the algorithm is that it correctly accommodates geometric and constitutive nonlinearities, both of which are essential for this application. Soft tissues generally can tolerate large deformations and their stress-strain response is seldom linear [60]. The efficiency of the algorithm derives from two aspects: 1) the total Lagrangian framework allows shape function derivatives to be precomputed and stored, rather than re-computed at each time step; and 2) the low stiffness of biological tissues means the critical time steps for explicit integration, normally a very restrictive constraint, are relatively large. Since explicit methods involve comparatively inexpensive computations in each time step, the latter feature can lead to very low overall computation times.

A key property of explicit methods like TLED is that all computations are carried out on an element- and node-wise basis, making these algorithms very amenable to parallel execution. Whereas the main computational task in implicit methods is the solution of a large linear system (several times per time step for nonlinear problems).

NiftySim also includes a number of features that go beyond the solid-element based TLED algorithm, the most important of which are: 1) membrane and shell formulations compatible with TLED's explicit time integration (described in [18] and [44], respectively), that can be used to simulate thin independent structures or, via node-sharing with solid elements, for simulation of membranes attached to solid organs; 2) specialised contact models for the efficient simulation of interactions between deformable geometry and simple, analytically describable surfaces; 3)

a general-purpose, Lagrange multiplier-based contact model [69]. The latter can simulate contacts between multiple deformable bodies, self-collisions, and contacts between deformable geometry and rigid surfaces.

NiftySim is primarily aimed at researchers developing algorithms in the area of medical image analysis, surgical image guidance, and surgical simulation, requiring a fast FE backend for the simulation of soft-tissue mechanics. It is mainly geared towards an algorithmic generation of simulation descriptions and post–processing of results with custom researcher–written code. Therefore, the package's objective is not to compete with end–to–end toolkits like SOFA[2] that provide their own tools for graphical simulation definition and interaction, or general-purpose finite element analysis (FEA) suites like Abaqus FEA[3]. Further, unlike the common commercial packages, which must be accessed via the command line, NiftySim can be used as a backend library in C++ applications thus allowing for the direct exchange of data with client code. Thus one of the potential usage scenarios for NiftySim is for it to be used as an explicit dynamic solver backend for one of the aforementioned end–to–end FEA software packages.

To aid the integration of NiftySim in applications, it sports the following features: It has been tested on various versions of Linux, Mac OS and Windows. A command line application capable of executing complete simulations and that can be used in conjunction with scripting languages or for prototyping simulations is included. Various features simplifying its use as a library are also available, such as a wrapper simulator class, which encapsulates all of the simulation technology and allows it to be easily embedded in other libraries and applications, and full support for CMake's[4] *config mode*.

The rest of this chapter consists of: an overview of related work (Section 4.2), followed by a summary of the core algorithm in Section 4.3. An introduction to NiftySim's usage by means of two examples is given in Section 4.4. This is followed by a description of the main classes and their implementation and XML API (Section 4.5). Finally, the chapter is concluded with a brief discussion (Section 4.6).

## 4.2 Related Work

TLED, i.e. the combination of a total Lagrangian evaluation of internal forces and explicit central difference time integration, was first identified as a potentially efficient approach for soft tissue simulation by Miller et al. [98]. However, a similar parallel algorithm was proposed earlier in Szekely et al. [134] for surgical simulation.

The earliest implentations of NiftySim [137] exploited the parallelisation potential of the TLED algorithm via OpenGL and the Cg graphics language. The introduction of the general purpose CUDA API [107] allowed for a more flexible and efficient implementation described in

[2]Simulation Open Framework Architecture, available from `http://www.sofa-framework.org`

[3]Abaqus FEA is a product of Dassault Systèmes, `http://www.3ds.com/products-services/simulia/portfolio/abaqus/`

[4]NiftySim supports CMake versions ≥ 2.8 obtainable from `http://www.cmake.org`

[140, 141, 138], resulting in the foundation of the software package described in this chapter. In a separate publication NiftySim's efficient implementation of anisotropic viscoelasticity [141] was described. NiftySim's reduced order modelling component that allows for running simulations with larger time steps than usually possible with explicit time integration was described slightly later [142, 143], with surgical simulation as the primary intended application area. Reduced order modelling works by recasting the equilibrium equation in a lower dimensional basis, in turn computed through *proper orthogonal decomposition* performed on full-order displacement solution computed with varying loads.

TLED has been applied in a number of high-profile publications, some of which were mentioned in Chapter 1: In Wittek et al. [152] TLED was used to regularise surface displacements obtained for the exposed brain surface in craniotomy from intra-operative MRI. The model of the beating heart in Pratt et al. [113] was based on TLED with the heart motion simulated using the inverse method described in ref. [112]. In Miller et al. [99], the authors provided a review of their group's work on brain tissue biomechanics modelling for image-guided neuro-surgery and surgical simulation by means of TLED. They advocated the use of TLED for these applications due to the ease with which the tissue's constitutive non-linearity and non-linearity arising from large deformation can be implemented, and allowing for simulation speeds compatible with haptic feedback and fast intra-operative image registration.

Some of these publications also used the NiftySim software that is discussed here: Hu et al. [63] used NiftySim to run a large number of simulations with varying material parameters and boundary conditions to build *statistical motion models* for the human prostate. These motion models they proposed as means for the fast initialisation of an image registration of transrectal ultrasound to MRI images, via a few known landmark displacements. Han et al. [51] exploited NiftySim's speed to run the large number of simulations required to determine material parameters using their simulated annealing algorithm. The reference also contains a very informative comparison between NiftySim and the commercial solver package Abaqus. A more recent application by Mertzanidou et al. [95] used NiftySim to simulate a large number of breast compressions with varying parameters to establish correspondences between mammography x-rays and breast MR images.

## 4.3 The Total Lagrangian Explicit Dynamics (TLED) Algorithm

### 4.3.1 The Basic TLED Algorithm

At its core, TLED, as described by Miller et al. [98], is an algorithm for the treatment of large deformation dynamic problems defined on a spatial domain $\Omega \subset \mathbb{R}^3$ for a time period $[0, T]$ given by an equilibrium equation of the form

$$\underbrace{\rho \ddot{\boldsymbol{u}}(\boldsymbol{x}, t)}_{\text{inertia}} + \underbrace{\nabla \cdot \boldsymbol{\sigma}\left(\boldsymbol{u}(\boldsymbol{x}, t)\right)}_{\text{internal forces}} = \underbrace{\boldsymbol{f}(\boldsymbol{x}, t)}_{\text{body forces}}, \quad \boldsymbol{x} \in \Omega,\ t \in [0, T] \tag{4.1}$$

where $\rho$ is the material's mass density, $\boldsymbol{\sigma}$ denotes the Cauchy stress in the simulated body, and $\boldsymbol{u}$ is the displacement field and $\ddot{\boldsymbol{u}}$ the corresponding acceleration.

The Dirichlet and Neumann BCs corresponding to Eq. (4.1) are given by:

$$\begin{aligned} \boldsymbol{u}(\boldsymbol{x}, t) &= \boldsymbol{u}^t_{\text{constraint}}, \quad \boldsymbol{x} \in \Gamma_u \\ f(\boldsymbol{x}, t) &= \boldsymbol{f}^t_{\text{constraint}}, \quad \boldsymbol{x} \in \Gamma_f \end{aligned} \tag{4.2}$$

Performing the usual substitution of a piece-wise linear approximation for the displacement field $\boldsymbol{u}$ and casting into the weak form via Galerkin weighting, the semi-discretised form of Eq. (4.1) becomes

$$\boldsymbol{M}\ddot{\boldsymbol{U}} + \boldsymbol{D}\dot{\boldsymbol{U}} + \boldsymbol{R}^{\text{int}}(\boldsymbol{U}) = \boldsymbol{R}^{\text{ext}} \tag{4.3}$$

where $\boldsymbol{M}$ is the *lumped*, i.e. diagonal, mass matrix and $\boldsymbol{D}$ is a diagonal damping matrix, introduced for the numerical stability of the time integration. In TLED the latter is linked to the mass matrix via a damping coefficient $\alpha_D$: $D = \alpha_D M$. $\boldsymbol{R}^{\text{ext}}$ are the discretised external loads, i.e. body forces and Neumann BCs.

The lumped mass matrix $M$ is generated with Algorithm 4.1, by equally distributing element masses over all element vertices.

**Algorithm 4.1** Algorithm for generation of the lumped mass matrix.

$\boldsymbol{M} \leftarrow \mathbf{o}$
**for all** $e \in$ {Elements} **do**
  $m \leftarrow$ `Volume`$(e) \cdot$ `Density`$(e)$ {Compute mass of element $e$}
  **for all** $n \in$ {Nodes of element $e$} **do**
    $M_{nn} = M_{nn} + m/N_{\text{nodes/element}}$ {Equally divide element mass between its nodes}
  **end for**
**end for**

The internal force term, $\boldsymbol{R}^{\text{int}}$ in Eq. (4.3), is given by

$$\boldsymbol{R}^{\text{int}} = \mathop{\mathbf{A}}_{e}^{N_{\text{elements}}} \boldsymbol{f}^{(e)} \tag{4.4}$$

where $\mathbf{A}$ is the assembly operator performing the accumulation of the element internal-forces, $\boldsymbol{f}^{(e)}$, that are in turn given by

$$\boldsymbol{f}^{(e)} = \int_{V^e} \partial_{\boldsymbol{X}} \boldsymbol{b} \boldsymbol{S} \boldsymbol{F}^{\mathrm{T}} \, \mathrm{d}V^e \tag{4.5}$$

where $\partial_{\boldsymbol{X}} \boldsymbol{b}$ are the derivatives of the shape functions $\boldsymbol{b}$ with respect to the reference configuration coordinates $\boldsymbol{X}$, $\boldsymbol{S}$ is the second Piola-Kirchhoff stress, and $V^e$ denotes the volume of element $e$. The deformation gradient $\boldsymbol{F}$ is defined as

$$\boldsymbol{F} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}} = \mathrm{I} + \sum_{i}^{N_{\text{nodes/element}}} \boldsymbol{U}_i \cdot \partial_{\boldsymbol{X}} b_i \tag{4.6}$$

with $\boldsymbol{x}$ being the current and $\boldsymbol{X}$ the initial position of a material point, and I denoting the $3 \times 3$ identity matrix

Use of the total Lagrangian evaluation of stresses means the shape function derivatives $\partial_{\boldsymbol{X}} \boldsymbol{b}$ only need to be computed once.

In the hyper-elastic case, the second Piola-Kirchhoff stress is given by

$$\boldsymbol{S} = \frac{\partial \Psi(C)}{\partial \boldsymbol{E}} = 2\frac{\partial \Psi(C)}{\partial \boldsymbol{C}} \tag{4.7}$$

in which the right Cauchy-Green deformation $\boldsymbol{C} := \boldsymbol{F}^{\mathrm{T}}\boldsymbol{F}$ was introduced, as was $\Psi$ denoting a strain density function that is solely a function of $C$. The simplest form of $\Psi$, which is also the one mostly used in this thesis, is the isotropic neo-Hookean hyperelastic, given by:

$$\Psi_{\mathrm{NH}} = \frac{G}{2}(\bar{I}_1 - 3) + \frac{K}{2}(J-1)^2 \tag{4.8}$$

with $G$ and $K$ being the symbols for the shear and bulk modulus material parameters, and $J := \det(F)$ and $\bar{I}_1$ being the first invariant of the modified right Cauchy-Green deformation tensor:

$$I_1 := \mathrm{trace}(J^{-2/3}\boldsymbol{C}) \tag{4.9}$$

An exhaustive list of the solid-element constitutive model available in NiftySim can be found in its user manual.

TLED employs one-point quadrature on the spatial domain, meaning the numerical approximation of $\boldsymbol{f}^{(e)}$ for the internal forces are evaluated only at the initial-configuration centre of the corresponding element. One of the following formulas is used, depending on the element type that is employed in the discretisation of the problem:

*Linear 8-node reduced-integration hexahedron*: This element employs trilinear shape functions, and the formula for its internal forces is given by

$$\boldsymbol{f}^{(e)} = 8\det(\boldsymbol{J})\partial\boldsymbol{b}\boldsymbol{S}\boldsymbol{F}^{\mathrm{T}}, \tag{4.10}$$

where $\boldsymbol{J}$ is the element Jacobian matrix. With one-point quadrature, this element is susceptible to spurious zero-energy modes – so-called hourglass modes, which can be efficiently controlled using the method proposed by Joldes et al. [72].

*Linear 4-node tetrahedron*: This element employs linear shape functions. The formula (4.5) for element nodal forces is then

$$\boldsymbol{f}^{(e)} = V^e\partial\boldsymbol{b}\boldsymbol{S}\boldsymbol{F}^{\mathrm{T}}. \tag{4.11}$$

It should be noted that this element is generally overly stiff, especially for nearly incompressible materials like soft tissues [65]. The nodal-averaged pressure tetrahedron, below, is preferable in most cases.

*Nodal-averaged pressure 4-node tetrahedron*: Developed to alleviate the volumetric locking problems that plague the standard tetrahedron, this element employs the same shape functions and nodal forces formula (Eq. (4.11)). The stress $\check{\boldsymbol{S}}$, however, is computed using a modified deformation gradient whose volumetric component has been averaged over adjacent nodes – see [73]. The performance of this formulation is generally superior to that of the standard tetrahedron.

The other major reason for the algorithm's efficiency is its treatment of the time ODE. Two distinct explicit ODE solvers are implemented in NiftySim:
*Explicit Central-Difference Method (CDM)*: With this method solving for the next time-step displacements, $\boldsymbol{U}^{(n+1)}$, at a given time step $n$, is achieved by substituting the following approximations for the velocity, $\dot{\boldsymbol{U}}$, and the acceleration, $\ddot{\boldsymbol{U}}$, into Eq. (4.3):

$$\begin{aligned} \ddot{\boldsymbol{U}}^{(n)} &\approx \tfrac{1}{\Delta t^2}\left(\boldsymbol{U}^{(n+1)} - 2\boldsymbol{U}^{(n)} + \boldsymbol{U}^{(n-1)}\right) \\ \dot{\boldsymbol{U}}^{(n)} &\approx \tfrac{1}{2\Delta t}\left(\boldsymbol{U}^{(n+1)} - \boldsymbol{U}^{(n-1)}\right) \end{aligned} \tag{4.12}$$

with $\Delta t$ denoting the time step size. Solving for the next time-step displacements yields

$$\boldsymbol{U}^{(n+1)} = \boldsymbol{A}\left(\boldsymbol{R}^{\text{ext}} - \boldsymbol{R}^{\text{int}}\right) + \boldsymbol{B}\boldsymbol{U}^{(n)} + \boldsymbol{C}\boldsymbol{U}^{(n-1)} \tag{4.13}$$

where the following coefficient diagonal matrices have been introduced:

$$\begin{aligned} A_{ii} &= 1/\left(\tfrac{D_{ii}}{2\Delta t} + \tfrac{M_{ii}}{\Delta t^2}\right) \\ B_{ii} &= \tfrac{2M_{ii}}{\Delta t^2}/\left(\tfrac{D_{ii}}{2\Delta t} + \tfrac{M_{ii}}{\Delta t^2}\right) \\ C_{ii} &= \left(\tfrac{D_{ii}}{2\Delta t} - \tfrac{M_{ii}}{\Delta t^2}\right)/\left(\tfrac{D_{ii}}{2\Delta t} + \tfrac{M_{ii}}{\Delta t^2}\right), \quad i = 1 \cdots N_{\text{nodes}} \end{aligned} \tag{4.14}$$

These coefficients are time-invariant and can be precomputed.
*Explicit Newmark Method (ENM)*: This method introduces a numerical acceleration and velocity. It is summarised by the following formulas:

$$\begin{aligned} \ddot{\boldsymbol{U}}^{(n)} &= \tfrac{1}{1+\alpha_D \Delta t/2}\left(M^{-1}\boldsymbol{R}^{\text{eff}} - \alpha_D \dot{\boldsymbol{U}}^{(n-1)} - \tfrac{\alpha_D \Delta t}{2}\ddot{\boldsymbol{U}}^{(n-1)}\right) \\ \dot{\boldsymbol{U}}^{(n)} &= \dot{\boldsymbol{U}}^{(n-1)} + \tfrac{\Delta t}{2}\left(\ddot{\boldsymbol{U}}^{(n)} + \ddot{\boldsymbol{U}}^{(n-1)}\right) \\ \boldsymbol{U}^{(n+1)} &= \boldsymbol{U}^{(n)} + \Delta t \dot{\boldsymbol{U}}^{(n)} + \tfrac{\Delta t^2}{2}\ddot{\boldsymbol{U}}^{(n)} \end{aligned} \tag{4.15}$$

As with CDM, coefficient diagonal matrices can be precomputed to accelerate the process.

Dirichlet BCs are incorporated at the end of a time step via a simple substitution of fixed values for the components of the displacement vector $\boldsymbol{U}$ that are subject to such constraints.

### 4.3.2 Incorporation of Membranes and Shells in TLED

The membrane element implemented in NiftySim is based on [18]. It is an iso-parametric triangle element in which the strain is computed via the usual reference triangle

$$T_{\text{ref}} = \{(0,0),(1,0),(0,1)\} \tag{4.16}$$

from the Jacobian matrices of the mappings from the reference to the current and the initial configurations

$$\begin{aligned} \boldsymbol{F}_{\mathbf{o}} &= \tfrac{d\boldsymbol{X}}{d\boldsymbol{\xi}}, \quad \boldsymbol{F}_{\boldsymbol{n}} = \tfrac{d\boldsymbol{x}}{d\boldsymbol{\xi}} \\ \boldsymbol{C}_{\mathbf{o}} &= \boldsymbol{F}_{\mathbf{o}}^{\boldsymbol{T}}\boldsymbol{F}_{\mathbf{o}}, \quad \boldsymbol{C}_{\boldsymbol{n}} = \boldsymbol{F}_{\boldsymbol{n}}^{\boldsymbol{T}}\boldsymbol{F}_{\boldsymbol{n}} \end{aligned} \tag{4.17}$$

Currently, the only available constitutive model for this element is incompressible neo-Hookean, whose SPK stress is given by

$$\boldsymbol{S}_{\boldsymbol{\xi}} = \mu\left(\boldsymbol{C}_{\mathbf{o}}{}^{-1} - \frac{II_{\boldsymbol{C}_{\mathbf{o}}}}{II_{\boldsymbol{C}_{\boldsymbol{n}}}}\boldsymbol{C}_{\boldsymbol{n}}{}^{-1}\right) \tag{4.18}$$

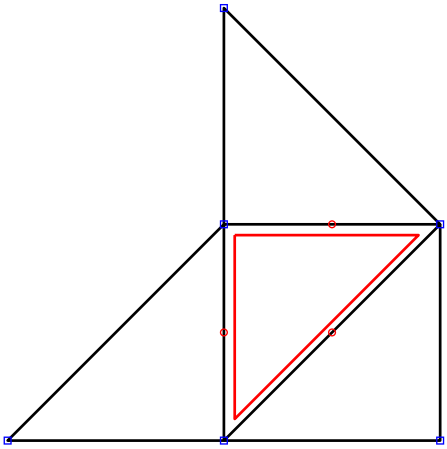


Figure 4.1: The 4-triangle patch underlying the calculations with the EBST1 shell element. The central triangle and its sampling points are highlighted in red. The blue boxes show the location of the six quadratic shape functions.

where $\mu$ is the shear modulus, and the strain invariant $II_{\boldsymbol{C}} = \det(\boldsymbol{C})$ was introduced.

The membrane internal forces are then given by

$$\boldsymbol{f}^{(e)} = A^e H^e (\boldsymbol{F_n} \boldsymbol{S_\xi}) : \partial_{\boldsymbol{\xi}} \boldsymbol{b} \tag{4.19}$$

with $A^e$ and $H^e$ denoting the initial element area and thickness, respectively, and the subscript $\boldsymbol{\xi}$ indicating quantities evaluated on the reference triangle.

The shell element supported by NiftySim is the rotation-free EBST1 described in [44]. Computations with this element are based on quadratic shape functions defined on patches consisting of 4 triangles (Fig. 4.1) with deformation and curvature functions being sampled at the mid-points of the edges of patches' central triangle and subsequently averaged. With this shell element the curvature giving rise to its bending stiffness is computed from standard nodal displacements, therefore there is no need for modifications to the time-ODE solver algorithms employed with TLED.

The standard neo-Hookean model is currently the only available constitutive model for the membrane component, the bending moments are computed from the linear expression:

$$\boldsymbol{m} = \frac{EH^{e3}}{12(1-\nu^2)} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{pmatrix} \boldsymbol{\kappa} \tag{4.20}$$

with $E$ and $\nu$ denoting Young's modulus and the Poisson ratio, $\boldsymbol{\kappa}$ being the curvature. The constitutive models for the membrane and bending component were taken from [108].

### 4.3.3 Contact Modelling

All contact modelling in NiftySim is based on prediction-correction, i.e. the basic TLED algorithm is used compute a prediction for the next time-step displacement, which is then used to search for potential contacts. If contacts are found, corrections must be computed. These

can either be displacement corrections, directly applied to the displacement value of offending nodes, or collision response forces which are incorporated in the effective load vector, $\boldsymbol{R}^{\text{eff}}$.

In the simpler of the two contact modelling algorithms implemented in NiftySim, the penetration of deformable geometry nodes into the *master* surface is found by evaluating an analytical expression. In this contact-modelling context, the deformable geometry surface is referred to as the slave surface.

The master-surface description must allow for the evaluation of a *gap function*, denoted with $g$, whose value represents the signed distance to the closest point on the master surface and, if negative, indicates that the slave node has penetrated the master surface. This also implies that there must be a means of computing the surface normal, $\boldsymbol{n_m}$, at every point on the master surface. The latter two quantities, $g$ and $\boldsymbol{n_m}$, can then be used to compute a displacement correction, $\boldsymbol{\Delta u}$, that can be directly added to the global displacement vector at the end of the simulation time step:

$$\boldsymbol{\Delta u} = -g\boldsymbol{n_m} \tag{4.21}$$

The pipeline for modelling mesh-mesh contacts implemented in NiftySim is described in significantly more detail in the next chapter (Chapter 5), but a very brief summary is given here. The pipeline detects collisions between slave-surface nodes and the interior of master-surface facets, and intersections of slave and master surface edges with bounding volume hierarchies (BVHs). The collision responses are based on the explicit Lagrange multiplier method [57], which means a contact pressure $\lambda$ is computed and applied to master and slave surfaces such that any mesh intersection is immediately resolved. The forces applied to the slave ($\boldsymbol{f_s}$) and the master ($\boldsymbol{f_m}$) take the following shape:

$$\begin{aligned} \boldsymbol{f_s} &= \boldsymbol{n_m} \overbrace{\beta_s \frac{m_s g}{\Delta t^2}}^{=\lambda} \\ \boldsymbol{f_m} &= -\boldsymbol{n_m} \beta_m \tfrac{m_m g}{\Delta t^2} \\ \beta_s &= \tfrac{m_m}{m_s+m_m}, \quad \beta_m = 1-\beta_s = \tfrac{m_s}{m_s+m_m} \end{aligned} \tag{4.22}$$

where $g$ is the gap function, $m_s/m_m$ is the mass associated with the colliding slave and the master surface point, respectively, and finally, $\boldsymbol{n_m}$ is the normal direction of the master surface at the point of collision.

These collision response forces can be directly incorporated in the effective loads and used to update the displacement vector through a second evaluation of the CDM/ENM formulas, (4.13)/(4.14).

Further details of the continuum formulation and solution algorithms can be found in earlier publications related to this software package [139, 138, 141, 143] and the software user manual. The contact modelling portion is discussed in detail in the next chapter, as well as in [69, 68].

### 4.3.4 Implementation Overview

The processing of a simulation with NiftySim consists of three main stages. The first stage deals with the parsing of the simulation XML description and the loading of the simulation geometry. In the precomputation step, the spatial derivatives of the shape functions, the node masses, and constraint and contact modelling-related data are computed. In typical usage scenarios, the precomputation happens absolutely transparently to the user in the *simulator* class's constructor.

When the precomputation is finished, the simulator initialises the solution variables and constraints, and enters the main loop. The main loop iterates over the simulation time steps. In every time step, at the very least, the internal forces of the structure and, based on these forces, displacements must be updated. Figure 4.2 shows a graphic representation of NiftySim's workflow.

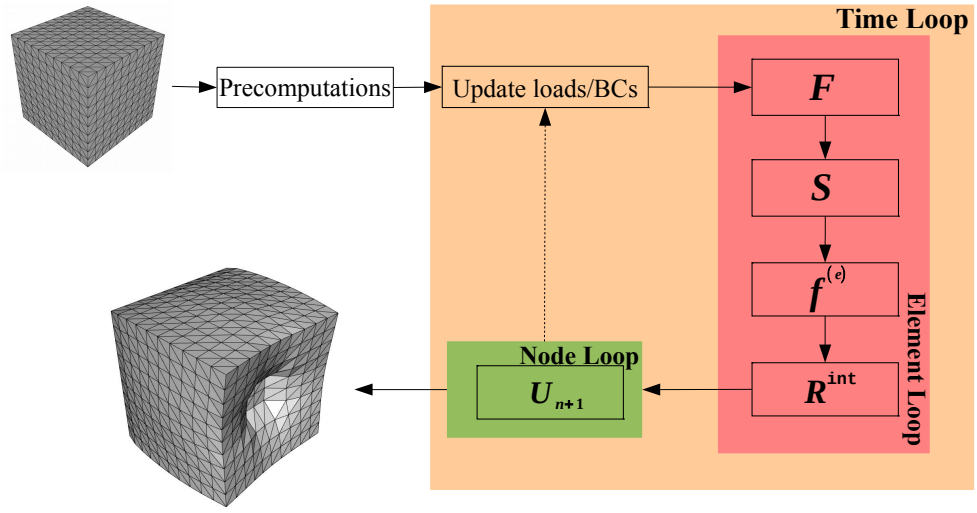



Figure 4.2: Flowchart representation of NiftySim's simulation pipeline.

In a minimal, sequential TLED implementation, Eq. (4.4) can be evaluated in one loop over all elements, computing in every element its deformation gradient, strains, stresses and from that internal forces, accumulating the per-element internal forces in a global internal-force vector. With this done, the effective loads can be computed by subtracting the internal forces from the applied external loads. A second loop is then invoked, iterating over the nodes in the mesh and updating their displacements based on Eq. (4.13). Thanks to the lumping of the mass matrix, this last step can be done for each node individually. Parallel implementations require a more complex memory layout to efficiently avoid race conditions on the internal-force accumulation buffer. The basic pattern of two main loops, one over all elements and one over all nodes, remains the same, though. A more detailed description of the strategies employed in NiftySim's parallel solvers is given in Section 4.5.5.

## 4.4 NiftySim Usage

This section gives an overview of NiftySim's usage by means of two example simulations. The geometry underlying the simulations is a human liver whose mesh was generated from manually segmented MR scans of a healthy volunteer. Its reference configuration is displayed in the left most picture of Fig. 4.5. The numerical values of the parameters are not intended to be reflective of a realistic simulation, but rather meant to highlight and demonstrate the effects of some of NiftySim's facilities and how they can be used in IGS and other image processing applications. A more comprehensive description of NiftySim's usage can be found in NiftySim's PDF user manual that ships with the source code.

While the NiftySim library API allows for a programmatical construction of a simulation in C++, the most sensible way to define a simulation in NiftySim is through its *XML* interface. At the root of any XML simulation description sits a *model* element, below which the most basic NiftySim simulation contains the following blocks:

1. A geometry block describing the simulation reference configuration. The underlying concepts and alternative ways of defining simulation geometry are discussed in Section 4.5.4.

2. An `ElementSet` block assigning material types and parameters to elements.

3. A system parameters block defining global simulation parameters such as simulation runtime, time step size, and the damping coefficient.

For a simulation to model any meaningful process, BCs and loads are required. These both fall under the umbrella term *constraints*, in NiftySim parlance, and are specified in blocks sitting directly beneath the root element in the XML simulation description. A convenient way of specifying nodes to which to apply a BC is via geometric criteria. This NiftySim facility selects surface facets through a combination of the direction of surface facet normals and optionally a bounding volume to which the search for boundary nodes is restricted. Using this type of boundary specification allows the user to define boundary geometry on meshes created with thrid-party meshing software without needing to know the exact node indexing in the mesh. Further, it allows the user to remesh the simulation geometry without having to redefine BCs.

In the example in Fig. 4.3, all facets with normals pointing in anterior direction are selected, while restricting the selection to the front-most quarter of the geometry's bounding box. A pressure constraint is then defined on this geometry. A mostly identical constraint is applied to the facets with normals pointing in postero-inferior direction. Such a configuration could be used as an approximation of the effects of gas insufflation, with the first constraint simulating the pressure being exerted on the exposed surface of the liver by the gas, while the second constraint acts as a proxy for the support offered by the viscera surrounding the liver. One further constraint is applied to the posterior part of the right lobe of the liver simulating the rear abdominal wall and spatially fixing the liver.

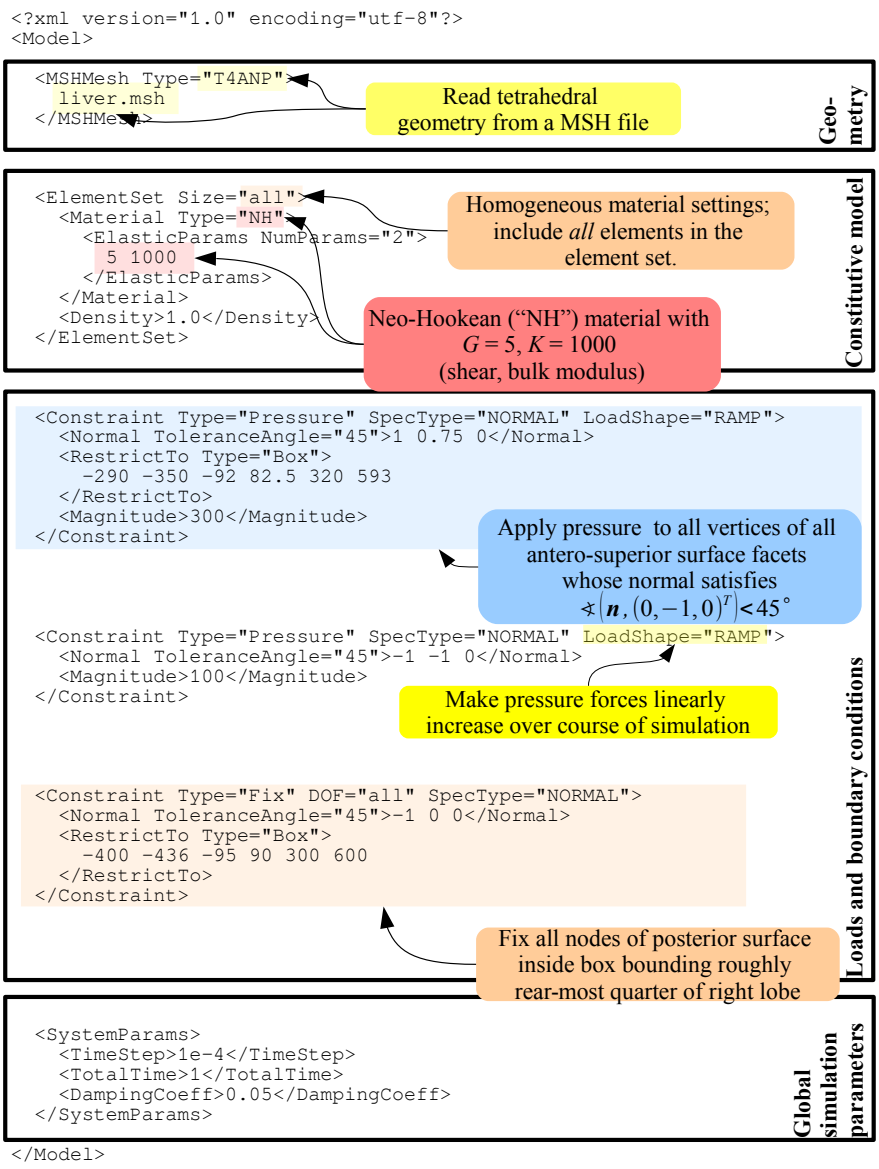



Figure 4.3: An annotated NiftySim simulation model of a liver being subjected to pressure forces from two sides.

```
<Constraint Type="Fix" DOF="all" SpecType="NORMAL">
  <Normal ToleranceAngle="45">-1 0 0</Normal>
  <RestrictTo Type="Box">
    -400 -436 -95 90 300 600
  </RestrictTo>
</Constraint>
```

Use *surface* of solid mesh geometry as membrane geometry

```
<ShellElements type="SURFACE" >
```

Membrane geometry

Homogeneous material settings; include *all* elements in the element set.

```
<ShellElementSet Size="all">
  <Material Type="NeoHookean">
    20
    <Density>1</Density>
    <Thickness>1</Thickness>
  </Material>
</ShellElementSet>
```

Neo-Hookean membrane $G = 20$

Membrane thickness

Membrane material parameters

```
<SystemParams>
  <TimeStep>1e-4</TimeStep>
  <TotalTime>1</TotalTime>
  <DampingCoeff>0.05</DampingCoeff>
</SystemParams>
```

Figure 4.4: Extension of the simulation from Fig. 4.3 with a membrane representing the Glisson capsule.

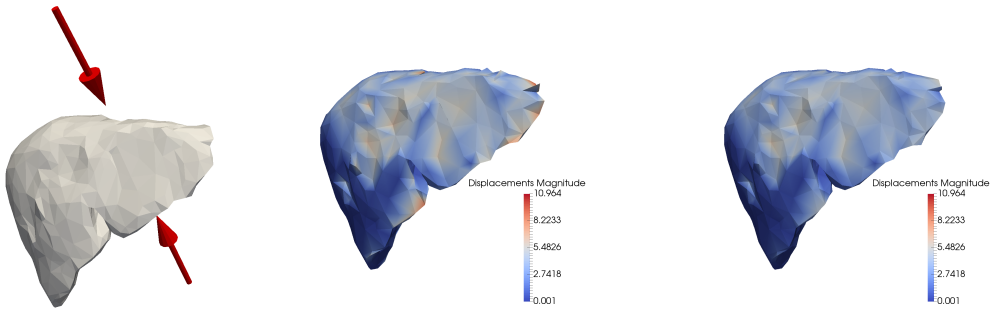



Figure 4.5: Left: initial configuration of the XML examples in Fig. 4.3 and 4.4; the red arrows indicate the approximate direction of the pressure forces. Centre: final configuration of the simulation described in Fig. 4.3. Right: final configuration of the simulation described in Fig. 4.4, highlighting the improved shape conservation and reduced displacement magnitude obtained when the simulation geometry is wrapped in a membrane.

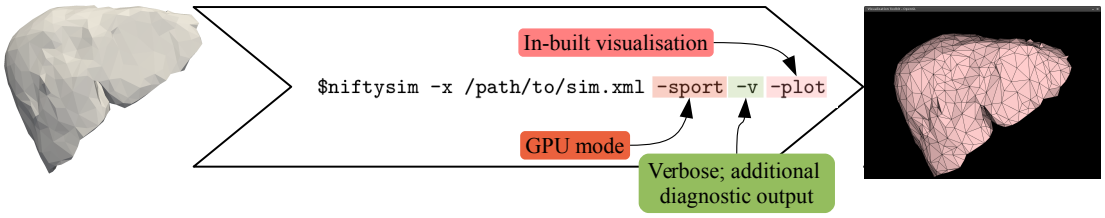



Figure 4.6: Execution of the simulation defined in Fig. 4.4 via NiftySim's stand-alone executable. Left: input geometry. Right: visual output of final configuration via NiftySim's VTK-based visualisation facilities. Centre: corresponding annotated command line.

An excerpt from a second simulation XML description is given in Fig. 4.4. The simulation is identical to the first one, except two new blocks have been added to simulate the effects of the Glisson capsule. NiftySim does not require the geometry of a membrane to be specified explicitly in cases where an organ is fully wrapped, as it can automatically extract surfaces of solid meshes and use those as the geometry for membrane and shell simulations. This is accomplished with the special attribute value `SURFACE` on the `Type` attribute of the `ShellElements` tag, that normally contains the 2D element connectivities.

To run these simulations one has two options. One option is to run it through NiftySim's command-line interface, the stand-alone `niftysim` executable. This is illustrated in Fig. 4.6 with the XML description from Fig. 4.3.

The NiftySim command-line application also has an `-export-mesh` switch that allows for an export of the simulation geometry and the final simulation displacements to a VTK unstructured grid file (this and further output options can be found in Section 4.5.9). The exported unstructured grid can then be used to visualise simulation results in applications such as *ParaView*[5], as was done in Fig. 4.5 and many other places throughout this text.

However, assuming the displacement field generated by the simulation is to be used in further processing with custom code, e.g., to warp an image, using NiftySim as a library in C++ code is the most advantageous. The simple C++ application in Fig. 4.7, consisting of a single compilation unit, `my_example.cpp`, containing only a `main` function, and a `CMakeLists.txt` for the build configuration, accomplishes the task of running *any* NiftySim simulation contained in the file residing at the hardcoded location `/path/to/my/sim.xml`.

[5] `http://www.paraview.org`

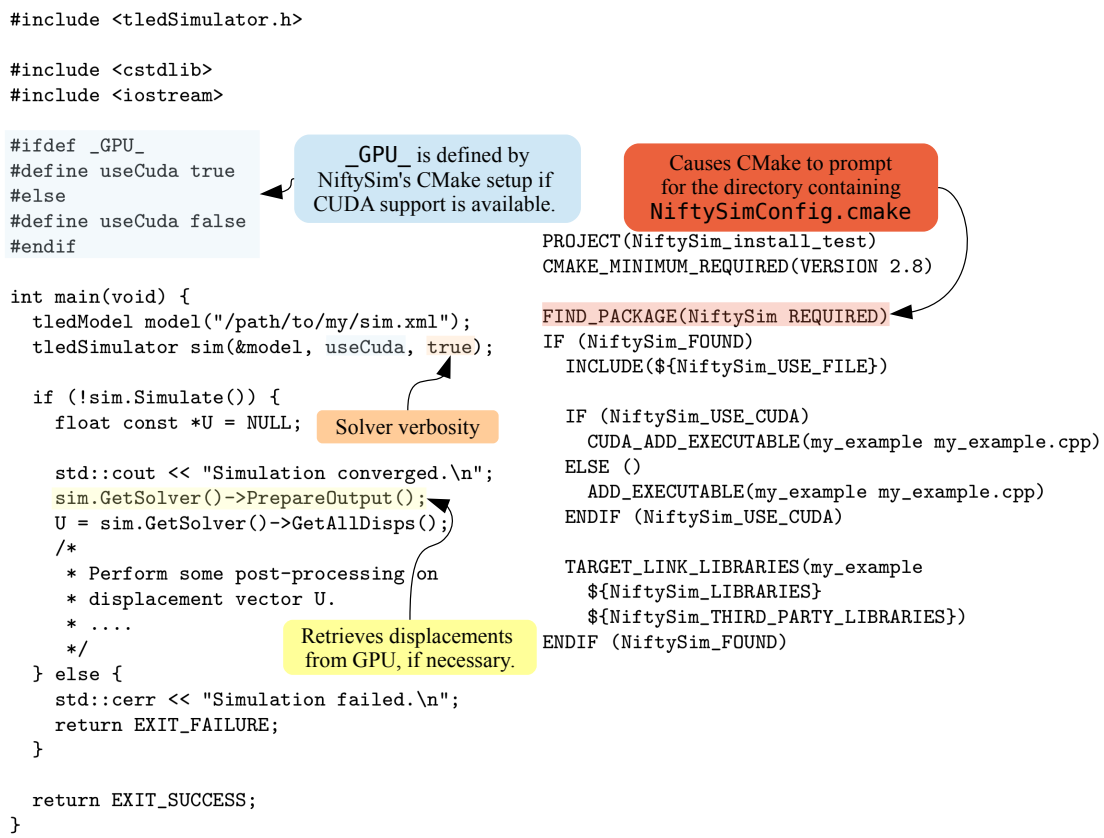



Figure 4.7: Left: a simple C++ application that uses displacements computed with NiftySim. Right: the corresponding `CMakeLists.txt` that takes care of the inclusion of the required NiftySim resources.

## 4.5 NiftySim Implementation

This section introduces the most important modules and concepts of the NiftySim software package and their implementation. A more complete list and technical description of NiftySim's modules can be found in the source code's *Doxygen*[6] documentation.

### 4.5.1 Coding Guidelines and Naming Conventions

NiftySim follows VTK[7] naming conventions, where class names have a "tled" prefix and are camel-cased, e.g. `tledExampleNiftySimClass`. Member names are also camel-cased and start with a capital letter. Names of functions normally begin with an appropriate verb.

Function signatures were until recently also based on VTK's style with no function arguments and member functions having const modifiers. Motivated by the addition of CPU parallel solvers and the potential race conditions it entails, a move towards a style more similar to that of the Insight Segmentation and Registration Toolkit[8] has been undertaken, where certain member functions such as getters have const modifiers, as do all read-only function arguments.

The CUDA portion of NiftySim was designed to be, as far as possible backward compatible. The use of complex classes in CUDA device code is therefore avoided. Instead, namespaces are used extensively to provide modularity and prevent name collisions, so that all functions and variables belonging to a particular module are wrapped in the same namespace, whose name is derived from the name of the corresponding module in the host portion of the code.

Doxygen-based API documentation is provided with most of the code, and important classes and namespaces are also associated with at least one Doxygen module.

### 4.5.2 The Simulator Class

`tledSimulator` is the normal entry point for anyone wanting to use NiftySim as an FEM backend. A major motivation for the introduction of this class was the encapsulation of all simulation components except the model, and thus the facilitation of the integration of NiftySim in C++ code, as was illustrated with the example in Fig. 4.7. Its most important member function, `Simulate`, contains the time stepping loop.

### 4.5.3 The Model Class

The `tledModel` class is the in-memory representation of the simulation description, usable by the other components of NiftySim. Internally, it stores the XML description of the simulation as a Document Object Model (DOM) tree, whose contents are accessible through member functions of `tledModel`.

A model can be defined recursively in XML through the notion of *sub-models*. Each sub-model is represented by its own `tledModel` instance whose management is done by `tledSubModelManager`.

[6] Doxygen is a tool for the extraction of inline API documentation, available from `http://www.doxygen.org`

[7] Visualisation Toolkit: `http://www.vtk.org`

[8] `http://www.itk.org`

### 4.5.4 The Mesh Representation

The `tledMesh` class only provides basic information about the mesh, such as node positions and element connectivity. For more complicated topological queries `tledMeshTopology` can be used. There is one instance of `tledMesh` accessible through the simulation's model whose purpose is to hold all solid-element geometry in the simulation, even if a simulation contains multiple disjoint bodies, as is the case with many contact problems.

NiftySim supports the loading of meshes from VTK unstructured grid files and the MSH[9] ASCII file format, as was illustrated in Section 4.4. Further, it can output simulation results in VTK unstructured grid files (Section 4.5.9). To simplify the running of simulations on algorithmically generated geometry, NiftySim provides its own mesh file format. The format is based on an inline definition of meshes through two blocks in the simulation XML description, located directly below the `Model` tag. Such a mesh specification comprises: 1) a `Nodes` element holding a list of node spatial positions, and 2) an `Elements` block containing the element connectivities by referencing nodes through their position in the list in the `Nodes` block. For an example of this more advanced way of specifying mesh geometry, the reader is kindly referred to the NiftySim user manual.

NiftySim also has some limited mesh manipulation capabilities allowing it to apply affine transforms to meshes read from files and to assemble larger connected meshes from the meshes contained in sub-models. The sub-model manager performs this mesh merging operation incrementally by searching for nodes whose positions are less than a user-specified distance apart. Therefore, its use is recommended only on conforming meshes.

There are dedicated surface-mesh classes for holding membrane and shell elements (see Section 4.5.5.4) and contact modelling (see Section 4.5.8, Chapter 5), all these classes are derived from `tledSurface`. The geometrical information necessary for shell and membrane computations is contained in a `tledShellMesh` instance that in turn depends on the global node position vector shared with the simulation's solid geometry. In cases where a solid body is wrapped in a membrane, the 2D mesh's connectivity information is directly obtained from the solid mesh by extracting its surface facets. `tledRigidContactSurface` is used for the modelling of contacts with arbitrarily-meshed rigid bodies and `tledDeformableContactSurface` holds the current-configuration surface for contact modelling purposes.

### 4.5.5 The Solver Classes

The purpose of `tledSolver` and its sub-classes is the coordination of the time step calculations involved in completing the simulation: compilation of internal forces and external loads, imposition of BCs, and update of displacements.

#### 4.5.5.1 `tledSolverCPU`

`tledSolverCPU` is the sequential C++ solver implementation of NiftySim. Precomputations of $\boldsymbol{M}$, $\partial \boldsymbol{b}$, etc. are performed in the class's constructor. The main computational tasks in each time

[9] MSH is the file format of GMSH

step are calculation of new internal nodal forces and calculation of new nodal displacements. The latter task is fully delegated to a dedicated CPU time-ODE solver class (described in Section 4.5.6). The sequential loop by which the former calculation is carried out is summarised in the pseudo-code loop at the centre of Algorithm 4.2.

The element-level calculations are performed by element classes, each of which is derived from `tledElement`. Concrete classes are provided for the three solid element types described in Section 4.3.1. The element objects are managed by the solver object. Each element object also has an associated material object (of base class `tledMaterial`), which is responsible for the constitutive behaviour of the element and enables evaluation of stress, given the element deformation. The task of computing BC values and body-forces for a given time is performed by a *constraint manager* (described in Section 4.5.7), but their accumulation and application is done by the solver. If applicable, a contact manager (`tledContactManager`) also resolves contacts between bodies in the model (see Section 4.5.8).

**Algorithm 4.2** Sequential time-step solution computation algorithm

$\boldsymbol{R}^{\text{ext}} \leftarrow$ `UpdateExternalLoads`$(t)$

$\boldsymbol{R}^{\text{int}} \leftarrow \mathbf{o}_{3\times N_{\text{nodes}}}$ {Initialise all internal forces with 0}

**for all** $e \in$ Elements **do**

$\boldsymbol{F} \leftarrow$ `ComputeDeformationGradient`$(e, \boldsymbol{U})$ {Performed by `tledElement`}

$\boldsymbol{S} \leftarrow$ `ComputeSPKStress`$(\boldsymbol{F}$, `Mat`$)$ {Compute second Piola–Kirchhoff stress based on constitutive model `Mat`, from deformation gradient $\boldsymbol{F}$}

$\boldsymbol{f} \leftarrow$ `ComputeInternalForces`$(\boldsymbol{F}, \boldsymbol{S})$ {Compute element-contribution to internal forces from stresses $\boldsymbol{S}$, deformation gradient $\boldsymbol{F}$}

$\boldsymbol{R}^{\text{int}} \leftarrow \boldsymbol{R}^{\text{int}} + \boldsymbol{f}$

**end for**

$U_{n+1} \leftarrow$ `UpdateDisplacements`$(\boldsymbol{R}^{\text{ext}} - \boldsymbol{R}^{\text{int}}, U_n, U_{n-1})$ {Operation performed by `tledTimeStepper`}

$U_{n+1} \leftarrow$ `ApplyDisplacementBC`$(U_{n+1})$

#### 4.5.5.2 `tledParallelSolverCPU`

`tledParallelSolverCPU` is a parallel CPU solver that is based on Boost[10] threads, for portability and compatibility with legacy compilers. It shares most of its code with `tledSolverCPU`. Its main distinguishing feature is that it splits the element array into blocks of equal size and assigns these sub-arrays to different threads. To avoid race conditions on the internal-forces buffer $\boldsymbol{R}^{\text{int}}$, every thread is associated with one intermediate force accumulation buffer, into which the internal forces of the elements in its sub-array are written. These temporary buffers are then summed up and the result is written to the global internal-force array.

This feature is kept purely optional to avoid a strict dependency on Boost.

[10] Boost is an open-source library available from `http://www.boost.org`

#### 4.5.5.3 `tledSolverGPU`

The nVidia CUDA solver implementation is called `tledSolverGPU`. All its precomputations are performed on the CPU with code resembling that of `tledSolverCPU`.

With most element types, only one kernel is required for the computation of the internal forces, which is invoked with one thread per-element. While conceptually there are few differences between that kernel and the loop body in Algorithm 4.2, the storage format for the element internal-forces is significantly different in that every element is assigned a `float3` buffer of size $N_{\text{nodes/element}}$, in which only the forces computed by one thread for one element are held (Fig. 4.8). These forces are later retrieved in the displacement update stage. Thanks to this storage format, no inter-thread communication or atomic operations are required.

The second important solver kernel, the displacement-update kernel, is invoked by the solver with one thread for every node. As is the case on the CPU, code associated with the solver is responsible for computation of the effective loads. The accumulation of the internal forces acting on a thread's node is performed by querying two texture arrays, one display array of type `int2` holding an offset and a range, and a second `int2`-array holding for every node the indices of the elements to which it belongs and its vertex index in those elements. Hence, these two arrays allow for a retrieval of all internal forces computed per element from the buffer that was filled by the internal-forces kernel. The look-up process is illustrated in Fig. 4.8. The external loads are computed on the CPU and passed as a global memory array to the kernel. The kernel is templated with respect to the `tledTimeStepper` sub-class used for displacement evolution, and the effective forces are next passed to the appropriate `tledTimeStepper` function, via template polymorphism that in turn returns a predictor displacement value for the thread's node. It is then checked if any of the node's components are subject to constraints through a binary mask held in texture memory, with one entry for every component of every node. If the component is constrained, the corresponding value is retrieved from another texture array.

An example of the handling of contact constraints on GPUs is given in Section 4.5.8.

#### 4.5.5.4 `tledShellSolverCPU`

Similar to how `tledSolverCPU` is responsible for the spatial discretisation with solid elements on the CPU, the `tledShellSolverCPU` class performs the tasks of computing the mass of shell and membrane elements and their internal forces.

Element sets are implemented as classes templated with respect to the membrane element type, so as to allow for a mix of membrane/shell element types in the same simulation. These templated classes are derived from a common abstract class `tledShellSolver::ElementSet` that has a pure virtual function `ComputeForces` that is responsible for the computation of internal forces in one element set, and receives a reference to the same buffer $\boldsymbol{R}^{\text{int}}$ used for accumulation of solid-element internal forces by `tledSolverCPU`. The contents of this function and its method of operation are largely analogous to the loop body of Algorithm 4.2, i.e. i) the computation of strain/curvature measures is delegated to element classes derived from `tled-`

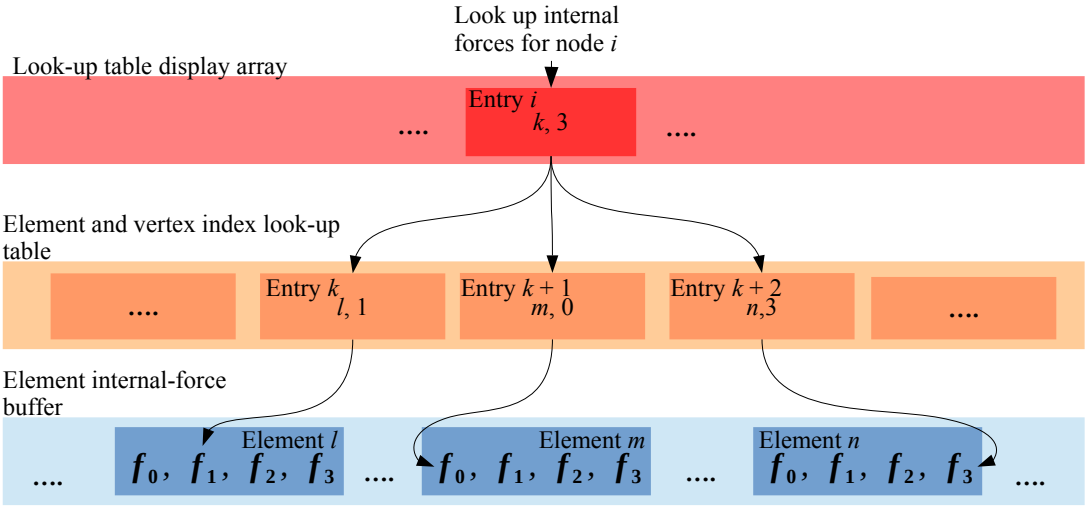



Figure 4.8: Layout of the buffer used for storage of internal forces on the GPU and illustration of their retrieval during computation of the effective loads (tetrahedral elements/4 nodes per element).

`ElementMembrane`; ii) a shell/membrane constitutive-model object associated with the element set is used for computation of the stresses arising from the strains/curvatures; iii) the element class converts the stresses to internal forces. Since the same force accumulation buffer is used as for solid elements, all BC and contact modelling operations can be performed by `tledSolverCPU`.

A class `tledParallelShellSolverCPU` exists to provide CPU parallelism. Its element set classes work by splitting their element arrays into equal parts that are assigned to different threads, very similar to how it is performed in `tledParallelSolverCPU`.

#### 4.5.5.5 `tledShellSolverGPU`

`tledShellSolverGPU` is the CUDA implementation of `tledShellSolverCPU`. Its internal organisation and a large amount of administrative and precomputation code is shared with `tledShellSolverCPU`. As with its CPU counter-part, one design goal of this class was to re-use solid-element solver code for BCs, contact modelling, etc. The strategy for force accumulation employed by `tledShellSolverGPU` is largely identical to that of `tledSolverGPU`, i.e. forces are computed and stored element-wise, to be later retrieved by a dedicated kernel invoked with one thread per node using the same type of lookup tables. The aggregated forces are directly subtracted from the external loads before these are passed to the displacement update kernel of `tledSolverGPU`.

The internal-forces kernel is templated with respect to the constitutive model and element class, and the appropriate functions for computation of the deformation, stresses, and internal forces are called via template polymorphism.

### 4.5.6 Time Integration

The base class of all ODE solvers used for the time integration is `tledTimeStepper`. Two further abstract classes, `tledTimeStepperCPU` and `tledTimeStepperGPU`, exist to provide the CPU and GPU specific parts of the ODE solver API, respectively. Mathematically, two types of explicit time integration are supported: the central difference method and explicit Newmark integration (see Section 4.3.1).

In order to maximise code reuse and consistency between the CPU and GPU implementations, a design pattern based on templated decorators, which is used in several places in NiftySim, was employed. In this case, the CDM/ENM-specific, but platform-independent parts of the implementation - e.g., getters for intermediate results such as velocity, are contained in two templated decorator classes, `tledCentralDifferenceTimeStepper` and `tledNewmark-TimeStepper`. These decorators derive from a solver base-class that is passed as a template argument, as follows

```
template <class TBaseTimeStepper>
class tledExampleDecoratorTimeStepper : public TBaseTimeStepper {
  ...
};
```

where `TBaseTimeStepper` is either `tledTimeStepperCPU` or `tledTimeStepperGPU`. These decorated CPU/GPU ODE solver base classes then serve as the parent class for the actual solver implementations, such as `tledCentralDifferenceTimeStepperCPU`.

The displacement evolution code of the GPU ODE solvers is implemented as a device function that is directly called by the displacement update kernel of the GPU solver. Unlike with the internal force computation, no precautions need to be taken to avoid race conditions, since the computation of the next displacement value of a given node only depends on its effective loads, and its current and previous time-step displacements.

### 4.5.7 Constraints

Loads and boundary conditions are incorporated under the common heading of constraints. All constraint types are represented by a sub-class of `tledConstraint`, e.g. `tledDispConstraint` implements non-zero displacement BCs. A class called `tledConstraintManager` is responsible for their management.

The constraint types accessible through the simulation XML-description were originally aimed at an algorithmic generation of BC definitions. Mostly, they are of a very basic type, such as displacement or force constraint, and require an explicit specification of the nodes directly affected by the constraint, thus making it difficult for humans to read and manually specify. More recently, a method of geometric boundary specification was added that allows the user to specify the surface facets contained in a boundary through a combination of facet normal-orientation criteria and bounding volumes. The processing and conversion to node index lists of these descriptions is done in `tledModel` with the aid of the classes `tledMeshSurface`, that

can extract surfaces of solid meshes and compute facet normals, and `tledNodeRejector` and its sub-classes that are used to filter nodes based on "is inside volume"-type criteria. An example of a geometrically defined BC was given in Section 4.4.

### 4.5.8 Contact Modelling

#### 4.5.8.1 Contacts with Analytically Described Surfaces

This feature enables the efficient simulation of contacts between soft-tissue and geometries frequently encountered in medical settings. Examples of analytical contact-surface classes are `tledContactCylinder` and `tledContactPlate`. There is no common interface for analytical contact surfaces since these are very simple classes holding only a few parameters necessary to describe the surface, such as the radius, the axis and origin of the centre line in the case of the contact cylinder.

For performance reasons, the actual computations related to these contacts are performed by `tledSolverGPU` in the displacement-update kernel. Algorithm 4.3 shows the computations performed to detect and simulate a contact between the deformable simulation geometry and a plate suitable for simulation of the breast compression in mammography. No CPU equivalent

**Algorithm 4.3** Collision detection and resolution with an analytically described plate

$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \leftarrow$ retrieve from global memory: plate corners
$\boldsymbol{n} \leftarrow (\boldsymbol{B} - \boldsymbol{A}) \times (\boldsymbol{C} - \boldsymbol{A})$ {Compute plate normal}
$\boldsymbol{p} \leftarrow$ input: node's current position
$\boldsymbol{d} \leftarrow \boldsymbol{p} - \boldsymbol{A}$
**if** $\boldsymbol{d}^{\mathrm{T}} \cdot \boldsymbol{n} < 0$ **then**
  {Node has penetrated the plane of the plate, need to check if it's within the bounds of the plate}
  **if** $0 \leq \boldsymbol{d}^{\mathrm{T}} \cdot (\boldsymbol{B} - \boldsymbol{A}) \leq \|\boldsymbol{B} - \boldsymbol{A}\|^2$ **and** $0 \leq \boldsymbol{d}^{\mathrm{T}} \cdot (\boldsymbol{C} - \boldsymbol{A}) \leq \|\boldsymbol{C} - \boldsymbol{A}\|^2$ **then**
    output $\leftarrow \left(\boldsymbol{d}^{\mathrm{T}} \cdot \boldsymbol{n}\right) \boldsymbol{n}$ {Return displacement pushing the node back to the plate surface}
    **return**
  **end if**
**end if**
output $\leftarrow \mathbf{o}$ {No displacement correction required}

exists for the analytical contact-surface feature.

#### 4.5.8.2 Mesh-Based Contact Modelling

A wide range of contacts can be modelled with the mesh-based code: contacts of multiple deformable bodies, deformable-body self-collisions, membrane and solid geometry contacts, and contacts between moving and static rigid bodies and deformable ones. A dedicated manager, `tledUnstructuredContactManager`, exists to manage the surface meshes used in the collision queries, the contact-search bounding volumes, and the *contact solvers* that compute the collision response forces. Similar to how the constraint manager provides loads and boundary

displacements to the solver for a given point in time, this manager provides member functions that can be called by the solver to get the forces arising from collisions for a given (predictor) displacement configuration without needing any in-depth knowledge of the type of contacts simulated, or the number of bodies involved in the contacts.

From an end-user perspective mesh-based contact modelling is similarly transparent. Detection and modelling of contacts between multiple unconnected deformable bodies is enabled with a single element in the `SystemParams` block of the simulation XML description. This element is called `DoDeformableCollision`, if only collisions between unconnected deformable bodies need to be detected, and `DoSelfCollision` if additionally self-collisions, i.e., one part of a deformable body colliding with another part of the same body, need to be modelled. One further XML construct called `ContactSurface` is available that allows for the insertion of arbitrarily-meshed rigid surfaces in the simulation. Concrete usage examples of these constructs will be given in Chapter 5.

The simulation's `tledUnstructuredContactManager` instance queries the model for the presence of these contact-modelling elements in the simulation description, during simulation pre-computation. If one or more are found, it will construct the necessary contact surfaces, BVHs, contact solver instances. The surface mesh extraction for the deformable geometry is fully automatic and done with the same code as used for parsing of geometric BC definitions (Section 4.5.7).

### 4.5.9 Output

#### 4.5.9.1 Visualisation

Some basic visualisation capabilities are included in NiftySim; these employ VTK for the rendering and window management. A custom render scene interactor, the *mesh sources*, which handle the conversion of NiftySim mesh objects and their attributes to VTK objects, and the source code for the creation of the render scene itself, are contained in a separate library called libviz.

#### 4.5.9.2 Mesh Output

The same converters that are used in the visualisation module can be used to export the simulation mesh with the final displacement as an attribute in VTK's `vtkUnstructuredGrid` format, or `vtkPolyData` in the case of membrane meshes. This functionality can be invoked through the NiftySim frontend with the `-export-mesh`, `-export-submesh`, and `-export-membrane` switch for the export of all simulation geometry as one mesh, as individual submeshes, or as surface meshes, respectively.

#### 4.5.9.3 Displacement and Internal Force History

`tledSimulator` also encapsulates an instance of `tledSolutionWriter`, which can record the time step displacements and internal forces. The displacements/forces are recorded in a Matlab parsable ASCII format at a frequency the user specifies through an attribute on the `Output` XML

element that is used to request the output of a variable ($\boldsymbol{F}$ or $\boldsymbol{U}$).

## 4.6 Discussion

The NiftySim toolkit has been designed to enable efficient integration of simulation technology into applications in medical image computing and computer assisted interventions. This integration is facilitated by both a command line program capable of executing simulations in a stand-alone fashion, and a library which enables simple embedding of the simulation code in third party software. High computational performance is achieved by employing a highly data-parallel FE algorithm and executing on general purpose computation-capable graphics processing units. The underlying formulation is valid for fully nonlinear problems, making it suitable for simulating materially nonlinear soft tissues undergoing large deformations. Moreover, the codebase is relatively small and can be compiled without any third party libraries, allowing a fast and easy compilation on a range of platforms, and an uncomplicated integration in client code. A series of example applications from recently published work can be found in the section on related work (Section 4.2), demonstrating the toolkit's practical utility.

Chapter 5

# Detection and Modelling of Contacts in Explicit Finite-Element Simulation of Soft-Tissue Biomechanics

## 5.1 Introduction

In this chapter, I give an overview, primarily over the mathematical and algorithmic underpinnings of the general-purpose contact modelling pipeline I implemented in NiftySim. This work resulted directly in one journal paper, Johnsen et al. [69], and two conference papers: Johnsen et al. [67], Johnsen et al. [68]. It was also employed for sensitivity analysis in one journal paper on which I was a co-author: Mertzanidou et al.: "MRI to X-ray mammography intensity-based registration with simultaneous optimisation of pose and biomechanical transformation parameters" [95].

Matrix-free explicit FEM solvers, for all their advantages with regards to parallelisation, and by extension speed, and in the implementation of complex constitutive models, suffer from the inherent shortcoming of only allowing for small time steps, which can make simulations involving large-deformation contact modelling prohibitively expensive, mainly due to the costs associated with contact search. Another drawback is that, compared to implicit methods, little literature is available on contact modelling for these solvers. The commonly encountered ones are the penalty-force and the Lagrange-multiplier method of Taylor and Flanagan [136], and Heinstein et al. [57], among the force-based methods [153], and kinematic contacts that rely on a direct correction of displacements and are very efficient, but are only capable of modelling contacts between deformable and rigid bodies [50]. All of these are typically implemented as node-segment contact algorithms only capable of detecting penetration of mesh nodes into surfaces, which requires two detection passes to achieve some degree of separation of the two surfaces in contact, and even with those two passes mesh edges are still free to intersect. Node-segment methods must rely on denser meshes to avoid the latter type of mesh intersection, which in turn entails more and computationally costlier time steps.

The algorithm presented in this chapter is implemented in the general purpose contact modelling component of NiftySim, whose code structure and usage was briefly explained in Section 4.5.8. It attempts to carry over some of the developments made in the context of implicit contact modelling algorithms to explicit methods, such as being able to process contacts in a single pass, as with segment-segment methods [124], provided the meshes in contact have a similar resolution. Spatial smoothing, which attempts to alleviate the stability issues caused by sudden changes in the direction of contact forces arising from the use of coarse, piece-wise linear contact surfaces has found widespread adoption in implicit methods [154, 115]. Further stability improvement is achieved by gradually slowing down approaching contact surfaces in close proximity, thus adding temporal smoothing to the method.

Another major area of focus in this chapter is the reduction of contact-search costs through BVHs with novel, time-saving update and self-collision detection heuristics. For self-collision detection, the surface-normal bounding-cone heuristics developed by Volino and Magnenat-Thalmann [148] are used. New formulas for the computation of the bounding cones, via Provot's recursive algorithm [114], are introduced. Another novel aspect is how the self-collision criterion is deeply integrated in determining the topology of the BVH and the decision on when to update BVH subtrees. The BVH updating algorithm is specialised for the typically small time-steps of explicit methods, in that it comprises a method for the characterisation of the deformation the simulation geometry has undergone, and identification of areas of negligible deformation and rigid motion, as well as updating of the BVH of the latter parts by means of rigid geometric transforms.

The proposed method is further notable due to its versatility; it allows for modelling of contacts between the surfaces of solid meshes, self-collisions, contacts between solid and membrane meshes, and deformable bodies interacting with moving or fixed rigid ones, and a new, simple friction model is also available.

This chapter is organised as follows: After a quick overview of related previous work (Section 5.2), Section 5.3.1 contains a very short summary of the underlying TLED algorithm, and an introduction of the related notation used in this chapter. This is followed by a detailed discussion of the contact modelling pipeline that is subdivided into a relatively short part describing the contact surface data structures (Section 5.3.3) and two larger sub-sections, the first of which deals with the contact search (Section 5.3.4). Novel modifications to the self-collision detection method, and the new BVH creation and update strategies are then discussed. The contact model is developed in Section 5.3.5. It starts with a discussion of penetration response forces for node-facet and edge-edge collisions, followed by a discussion of the temporal smoothing and the friction model. Algorithmic adaptations required for a CUDA implementation are discussed in Section 5.3.6. In Section 5.4 the practical usage of NiftySim's contact modelling facilities is illustrated by means of a basic example inspired by liver-surgery image guidance. In Section 5.5, the BVH algorithms are validated in the order in which they are presented by

comparison to some alternatives that swap out some of the novel aspects for simpler or more established methods, on mostly synthetic test cases. The contact model is validated in terms of energy and momentum conservation, and plausibility of the trajectories resulting from impacts (Section 5.5.4). In Section 5.5.5, the friction model is validated on a benchmark problem with an analytical ground truth taken from the literature. Finally, a demonstration of the entire pipeline's performance on two image-guidance problems is provided (Section 5.5.6), including that of the GPU implementation (Section 5.5.7).

## 5.2 Related Work

Classically, the algorithms for FEM contact modelling are node-segment approaches [153]. These segments follow the basic pattern described in Section 4.3.3, i.e. one of the two surfaces in contact is assigned the role of the *slave* surface, the other is called the *master* surface. The only type of mesh inter-penetration node-segment approaches can resolve are those of the master surface by slave nodes, which in turn necessitates two contact search and resolution steps with alternating master-slave roles for every time step. The underlying principle of mesh intersection handling with node-segment methods is to project slave nodes onto the nearest facet of the master surface, and check the sign of the difference between the slave node position and its projection with respect to the master surface normal. A contact response in the direction of the master surface normal is then applied. Since most FEM elements are only C0 continuous, this approach can also lead to sudden jumps in the direction of the response experienced by a node sliding over the master surface, in turn leading to instability of the algorithm. Smoothing node-segment methods, such as the one by Wriggers and Krstulović-Opara [154] based on cubic Bézier polynomials, were introduced to remedy this issue.

Segment-segment contact algorithms were devised to overcome the need for two passes with the node-segment approach [124]. Mortar elements, originally developed for coupling non-conforming meshes, were introduced to the field of contact modelling to also overcome the various mathematical limitations of the early node-segment methods, mainly stability problems with implicit methods arising from non-satisfaction of the Babuska-Brezzi condition. In these methods, the contacting meshes are pushed apart by a contact pressure that is interpolated over the mortar mesh. The work of Puso and Laurensen [115] introduced a mortar method for 3D and large deformations.

An interesting method suitable for any FEM or similar algorithm that assembles stiffness matrices was developed by Duriez et al. [40]. Their contact model based on Signorini's law was primarily designed with haptics in mind, and computes contact forces from the constitutive model of the bodies in contact.

An alternative to both node-segment and segment-segment methods, based on intersection volumes, was devised by Heidelberger et al. [55], and significantly extended by Allard et al. [5]. By employing layered depth images (LDI) for intersection volume computation, they effectively solved collision detection and response calculation using the same method. However, the

method is limited in its application to volumetric meshes.

A notable development in the area of matrix-free explicit FEM algorithms came with the Lagrange multiplier-based method employed in PRONTO 3D by Taylor and Flanagan [136], and later extended by Heinstein et al. [57]. These methods, like the - probably most widely adopted for explicit FEM - penalty method, resolve mesh intersection by applying forces to the offending nodes. Unlike with penalty methods that contain an arbitrary non-physical parameter, the forces with the Lagrange multiplier method arise directly from the discretised equilibrium equation and lead to an immediate resolution of any mesh intersection and do not affect the admissible time step size [15]. Cirak and West [28] devised a method for simulating impact contacts with explicit solvers, based on an elaborate decomposition of contact responses into mesh inter-penetration responses and momentum exchanges. In terms of scope their work, resembles the pipeline presented here, with its ability to simulate membrane and solid mesh contacts, frictional as well as frictionless contacts, and handling of both edge-edge and node-facet collisions. Algorithmically there are notable differences, though: Their resolution of mesh inter-penetration was based on a unilateral projection of slave nodes onto the master surface, the consequences of which on the energy balance of the simulated system they tried to minimise by establishing an equation system incorporating both penetratration responses and momenta, thus requiring the assembly and inversion of a matrix equation system in every time step in which contacts are detected.

The range of contact search algorithms proposed for FEM contact modelling is as wide as that of methods for their solution. Spatial hashing [57], implicit surface descriptions [37] and image-space techniques [5, 55], have all been shown to be efficient choices for detecting contacts between deformable objects. A bounding volume hierarchy (BVH) based method is proposed here. These methods are very versatile and used in a wide range of applications, such as cloth modelling [96], robot motion planning [49], ray tracing [75], and FEM contact modelling [110, 155]. What makes them interesting for the application with the relatively small time steps required with explicit FEM solvers is the ability to take a more localised, selective approach to collision detection and exploitation of temporal coherence. A further advantage of employing a BVH is that it can also be used for other problems arising in surgical image guidance, such as fast point location for point-set registration purposes with spatial algorithms, like ICP.

Early developments in the field of BVHs were limited to rigid or even static problems but, since the 1990s, there has been a growing interest in collision detection for simulation of deformable bodies [86]. A key development came in 1994 with Volino and Magnenat-Thalmann's method [148] for efficient self-collision detection. They established two conditions under which a piece of simulated cloth could self-intersect: either the surface is folded onto itself, i.e. it has surface normals pointing in opposite directions, or there are intersecting boundary edges. Larsson and Akenine-Möller [82] devised a hybrid bottom-up/top-down BVH strategy for detecting

collisions between deformable bodies. Its purpose was to reduce the number of bounding volume (BV) updates by only updating down to leaf level those parts of the bodies' BVHs that overlap. The method, however, was not adapted for self-collision detection. They later [83] described a method for dynamically creating BVHs for triangle soups, particularly suitable for such resulting from fracturing of objects. They also developed a variant of their algorithm incorporating a sweep and prune sort of all simulation primitives suitable for detecting self-collisions.

## 5.3 Methods

### 5.3.1 Total Lagrangian Explicit Dynamics

As seen in Chapter 4, the TLED class of FEM solvers are matrix-free solvers relying on explicit central-difference time integration. They have enjoyed some success in the simulation of soft-tissue biomechanics thanks to their ability to simulate large deformations, the relative ease with which complex material models can be implemented, and, not least, the possibility for very elegant parallel implementations [98, 141, 139].

The discretised equilibrium equations of TLED, neglecting damping terms, read:

$$\frac{1}{\Delta t^2}\boldsymbol{M}\left(\boldsymbol{U}^{(t+\Delta t)} - 2\boldsymbol{U}^{(t)} + \boldsymbol{U}^{(t-\Delta t)}\right) + \boldsymbol{R}^{\mathrm{int}^{(t)}} = \boldsymbol{R}^{\mathrm{ext}^{(t)}} \tag{5.1}$$

where $\boldsymbol{U}^{(t+\Delta t)}$, $\boldsymbol{U}^{(t)}$, $\boldsymbol{U}^{(t-\Delta t)}$ denote the next, current, and previous time-step displacements, respectively, $\Delta t$ is the time step size, $\boldsymbol{R}^{\mathrm{ext}}$ is the external load vector, and $\boldsymbol{M}$ is the lumped (diagonal) mass matrix. The term $\boldsymbol{R}^{\mathrm{int}}$ represents the internal forces of the current configuration. The evaluation of the latter term does not involve the assembly of a stiffness matrix. Instead internal forces are computed directly per element as a function of the displacement solution, and subsequently accumulated for all nodes. In the experiments in this chapter, the internal forces are modelled with the neo-Hookean material model and shell-element internal forces are computed with the EBST shell triangle of Flores and Oñate [44], that was described in Chapter 4.

### 5.3.2 Contact Algorithm Overview

The algorithm is of a predictor-corrector type that first evolves the displacements with the standard TLED algorithm without any regard to contacts, then identifies intersecting or very close geometry and applies forces to correct the situation. The contact modelling pipeline comprises three major groups of routines and data structures: The *contact surfaces* which contain the geometry that is searched for collisions and some additional data required for contact-force application, the *BVHs* employed in contact search, and finally the *contact-force computation* algorithms.

A pseudo-code overview of the algorithm including the TLED-related computations is given in Algorithm 5.1.

**Algorithm 5.1** Time-stepping algorithm with all contact modelling and TLED operations.

$t \leftarrow t + \Delta t$ {Beginning of time-step}

$\boldsymbol{R}^{\text{ext}} \leftarrow \texttt{UpdateExternalForces}\,(t)$

$\boldsymbol{R}^{\text{int}} \leftarrow \texttt{UpdateInternalForces}\left(\boldsymbol{U}^{(t)}\right)$

$\boldsymbol{U}^{(p)} \leftarrow \texttt{UpdateDisplacements}(\boldsymbol{U}^{(t)}, \boldsymbol{R}^{\text{int}}, \boldsymbol{R}^{\text{ext}})$ {Compute displacement prediction with eq. (5.1)}

$C_{\text{def}}^{(t)} \leftarrow \texttt{Update}\left(C_{\text{def}}^{(t-\Delta t)}, \boldsymbol{U}^{(p)}\right)$ {Update deformable geometry contact surface $C_{\text{def}}$}

$\texttt{BVH}_{\text{def}}^{(t)} \leftarrow \texttt{UpdateBVH}\left(\texttt{BVH}_{\text{def}}^{(t-\Delta t)}, C_{\text{def}}^{(t)}\right)$ {Update def. geometry BVH with Alg. 5.2}

$C_{dd} \leftarrow \texttt{FindCollisions}\left(C_{\text{def}}^{(t)}, \texttt{BVH}_{\text{def}}^{(t)}\right)$ {Find deformable geometry contacts}

$\boldsymbol{F_c} \leftarrow \sum_c^{C_{dd}} \texttt{ComputeForces}(c, C_{\text{def}}^{(t)})$

$\boldsymbol{F_c} \leftarrow \texttt{ConsolidateForces}(\boldsymbol{F_c}, C_{dd})$ {Correct for redundant constraints}

$\boldsymbol{R}^{\text{ext}} \leftarrow \boldsymbol{R}^{\text{ext}} + \boldsymbol{F_c}$

$C_{\text{movRig}}^{(t)} \leftarrow \texttt{UpdateContactSurface}\left(C_{\text{movRig}}^{t-\Delta t}, t\right)$ {Update moving rigid geometry $C_{\text{movRig}}^{(t)}$}

$\texttt{BVH}_{\text{movRig}}^{(t)} \leftarrow \texttt{UpdateBVH}\left(\texttt{BVH}_{\text{movRig}}^{t-\Delta t}, C_{\text{movRig}}^{(t)}\right)$ {Update moving rigid-body BVHs}

$C_{dr} \leftarrow \texttt{FindCollisions}\left(C_{\text{def}}^{(t)}, C_{\text{rig}}, C_{\text{movRig}}^{(t)}, \texttt{BVH}_{\text{def}}^{(t)}, \texttt{BVH}_{\text{rig}}, \texttt{BVH}_{\text{movRig}}^{(t)}\right)$ {Find deformable-rigid contacts}

$\boldsymbol{F_c} \leftarrow \sum_c^{C_{dr}} \texttt{ComputeForces}(c, C_{\text{def}}^{(t)})$

$\boldsymbol{F_c} \leftarrow \texttt{ConsolidateForces}(\boldsymbol{F_c}, C_{dr})$

$\boldsymbol{R}^{\text{ext}} \leftarrow \boldsymbol{R}^{\text{ext}} + \boldsymbol{F_c}$

$\boldsymbol{U}^{t+\Delta t} \leftarrow \texttt{UpdateDisplacements}(\boldsymbol{U}^{(t)}, \boldsymbol{R}^{\text{int}}, \boldsymbol{R}^{\text{ext}})$ {Compute next time-step displacement}

### 5.3.3 Contact Surfaces

Contact surfaces are central data structures in the proposed contact modelling algorithm. They consist of the geometric primitives – triangles are employed in the subsequent experiments and some of the explanations – which are tested for collisions. They also provide extended geometric information required in contact search, such as surface normals and projection operators.

The contact surfaces associated with fixed rigid geometry are static data structures for which all normals, projection operators required for contact search, and force calculation are precomputed. Moving rigid contact surface data structures are identical to their spatially fixed counterparts, apart from possessing an update routine that applies the appropriate translation and/or rotation to the precomputed normals and operators.

The most important type is the deformable contact surface obtained by extracting the surface facets from the simulation solid mesh. Lazy evaluation with caches is employed for normals, and other contact search related data such as projection operators, which allows the algorithm to limit their re-computation to regions that are in contact with or close proximity to other geometry. Contact forces which are calculated for a contact surface node are applied to the corresponding FEM node via an index lookup table that is constructed together with the surface mesh.

If the simulation contains membranes, these elements are included in the same contact surface object as the solid mesh surface facets. To account for the thickness of the membrane, two contact primitives are introduced for every membrane element, one for the top and one for the bottom. The nodes associated with these membrane contact primitives are obtained by offsetting the membrane nodes by half the thickness of the membrane in the direction of the normal and its opposite, for top and bottom respectively, yielding the sandwich structure visible in Fig. 5.20. By having the entries in the contact-force index lookup table point to the same FEM membrane node for the top and bottom node, it is ensured that contact forces are correctly incorporated in the global force vector.

### 5.3.4 Contact Search

The proposed algorithm finds geometric primitives that are intersecting or in very close proximity through bounding volume hierarchies. BVHs are built by recursively partioning the simulation geometry into geometrically simple volumes. Such a partioning is illustrated in Fig. 5.3. These data structures accelerate the contact search in two ways: 1) The search can be performed by traversing a tree-type data structure, hence the search for one specific geometric primitive can be accomplished with a runtime complexity that is logarithmic in the number of geometric primitives. 2) The geometry of the BVs is normally chosen such that performing intersection tests is computationally cheaper than directly performing the tests on the contained geometry.

In the following, the algorithm is explained for BVHs that have a binary-tree structure and axis-aligned bounding boxes (AABB) are used for illustrations. However, the presented

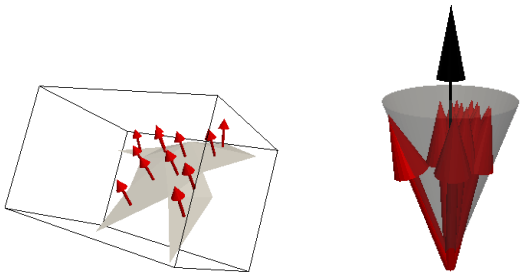


Figure 5.1: Surface-normal bounding cones for self-collision detection. Left: a patch of connected geometry primitives defining a cone with the corresponding surface normals (red) and its AABB. Right: the corresponding cone, the normals it bounds (red) and the cone axis (black).

methods are not limited to this BVH-type.

At the leaf level, the BVs bound one primitive each such that the primitive's vertices at the start of the time step as well as at the end of the predictor step are fully contained within it. The leaf BVs are also fitted with a margin $\epsilon_{BV}$ which is uniform throughout the BVH and defaults to $\frac{1}{100}(h_{\max} + h_{\min})$ in the implementation, with $h_{\max}$, $h_{\min}$ being the maximum and minimum initial-configuration surface facet diameters. The purpose of this margin is to allow for some geometry deformation without the need to refit the bounding volume and to allow for the detection of primitives in close proximity.

All deformable geometry is contained in one BVH, rigid contact surfaces, moving or fixed, each have each their own BVH.

#### 5.3.4.1 Self-Collisions and Surface-Normal Bounding Cones

The surface-normal bounding cones (NBC) are a means for identifying BVH subtrees containing connected geometry that is folded onto itself, i.e., has normals pointing in opposite directions, and thus potentially self-colliding [148]. Every BV in the BVH has one NBC associated with it. The surface normals of all primitives in a surface patch contained in the BV $\mathcal{B}_{\text{patch}}$ are inserted in such a bounding cone, so that its opening angle satisfies:

$$\alpha_{\text{VMT}} \geq \max_{i,j \in \mathcal{B}_{\text{patch}}} \angle\left(\boldsymbol{n_i}, \boldsymbol{n_j}\right) \tag{5.2}$$

where $\alpha_{\text{VMT}}$ is the NBC opening angle. One such cone is illustrated in Fig. 5.1.

Since we mostly deal with solid elements, self-collisions resulting from intersecting mesh boundaries as described by Mezger [96] are not considered, and checks for the presence of potential self-collisions in a surface patch can then be very efficiently performed by checking the opening angle of the associated NBC:

$$\alpha_{\text{VMT}} \geq \tau_{\text{VMT}}, \quad \tau_{\text{VMT}} \leq \pi \tag{5.3}$$

where $\tau_{\text{VMT}}$ the threshold above which self-collision tests are performed. The computation of this quantity $\alpha_{\text{VMT}}$ is done recursively as part of the BVH update with a method similar to that proposed by Provot [114], starting with the facet normal of the bounded primitive and an

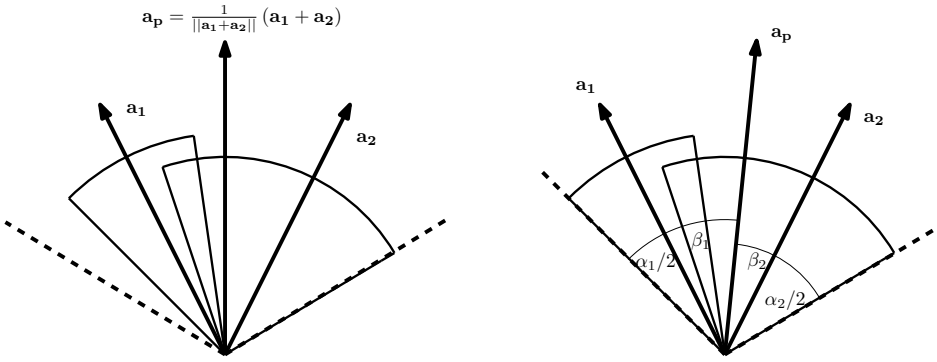



Figure 5.2: Provot's NBC (left) and narrowest NBC (right). Child cones drawn with solid lines, parent NBC with dashed lines.

opening angle $\alpha_{\text{VMT}} = 0$, at leaf level, and creating NBCs for interior nodes by merging of the cones of their children. Due to the recursive computation of the opening angle, it can exceed 180°, and opening angles of 180° or greater indicate a possible self-collision. For safety reasons, a threshold $\tau_{\text{VMT}}$ slightly below 180° is normally chosen.

Unlike Provot's algorithm that adopts for the parent cone's axis the average of the child cones' axes and then computes the opening angles such that the child cones are fully contained, the proposed approach computes the narrowest possible NBC from both the child cones' axes ($\boldsymbol{a_1}$ and $\boldsymbol{a_2}$) and opening angles ($\alpha_1$ and $\alpha_2$), as illustrated in Fig. 5.2. The computation of the parent axis $\boldsymbol{a_p}$ is accomplished via two weights $w_1 > 0$, $w_2 > 0$ whose formulas can be derived as follows: If $\beta$ denotes the angle between the two child axes and $\beta_1$, $\beta_2$ the respective angles between their's and the parent's axis, the optimum opening angle of the parent NBC $\alpha_p$ satisfies:

$$\alpha_p = 2(\beta_1 + \alpha_1/2) = 2(\beta_2 + \alpha_2/2) \tag{5.4}$$

Using this and the definition of $\beta_1$, $\beta_2$, it is obtained

$$\begin{aligned} \beta &= \beta_1 + \beta_2 = \arccos(\boldsymbol{a_2}^{\mathrm{T}}\boldsymbol{a_1}) \\ \beta_1 &= (2\beta - (\alpha_1 - \alpha_2))/4, \quad \beta_2 = \beta - \beta_1 \end{aligned} \tag{5.5}$$

That in turn is used to obtain the desired weights $w_1$, $w_2$ and subsequently the parent axis $\boldsymbol{a_p}$:

$$\begin{aligned} w_1 &= \cos(\beta_1) - e\frac{\cos(\beta_2)-e\cos(\beta_1)}{1-e^2}, \quad \text{where } e := \boldsymbol{a_1}^{\mathrm{T}}\boldsymbol{a_2} \\ w_2 &= \frac{\cos(\beta_2)-e\cos(\beta_1)}{1-e^2} \\ \boldsymbol{a_p} &= \frac{1}{\|w_1\boldsymbol{a_1}+w_2\boldsymbol{a_2}\|}(w_1\boldsymbol{a_1} + w_2\boldsymbol{a_2}) \end{aligned} \tag{5.6}$$

The algorithm only performs these calculations after first checking whether one of the child cones is fully contained in the other. If so, the containing cone is adopted as the parent cone. It is therefore guaranteed that all quantities appearing in (5.6) lie within the valid range.

#### 5.3.4.2 BVH Generation

Since the number of BVH subtrees that need to be tested for self-collisions is determined by the NBCs, the same are used to influence how the BVH is generated and updated, so as to reduce

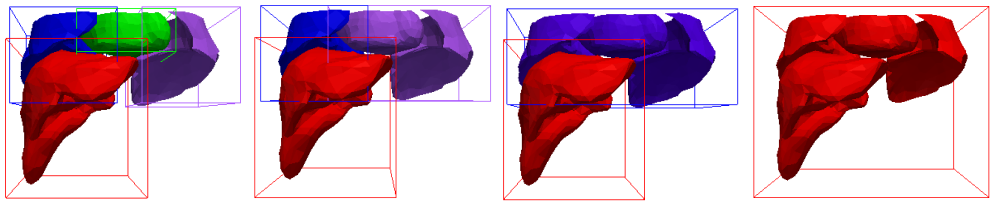


Figure 5.3: Illustration of the bottom-up BV merging process. The leftmost picture shows the initial state when the AABBs contain only connected geometry. The next picture shows the state after the two boxes yielding the smallest parent box have been merged. The last picture (rightmost) shows the BVH root bounding all geometry.

the number of BV intersection tests and updates. The generation process comprises two main stages. First, disconnected geometry is identified and the top part of the BVH is created from the boxes bounding these clusters. This process is conducted bottom-up (illustrated in Fig. 5.3 with meshes courtesy of IRCAD[1]). The second stage is the top-down division of the boxes bounding the connected primitives. In order to be able to apply the NBC self-collision criterion, the division process makes sure that the primitives contained in the newly created BVs remain connected. The cost function governing the assignment of primitives to child BVs, (5.7), consists of two quantities to be minimised: the volume of the resulting BV and the opening angle of the NBC.

$$\mathcal{B}^{n+1}_{\text{child}_i} = \mathcal{B}^{n}_{\text{child}_i} \cup \left\{ \arg \min_{T \in \mathcal{B}_{\text{parent}}} V(\mathcal{B}^{n}_{\text{child}_i} \cup \{T\})(1 + c \cdot \alpha_{\text{VMT}}(\mathcal{B}^{n}_{\text{child}_i} \cup \{T\})) \right\} \tag{5.7}$$

$\mathcal{B}_{\text{child}_i}$, $\mathcal{B}_{\text{parent}}$ denote the primitive sets bounded by the new children and the parent BV being split, respectively. $T \in \mathcal{B}_{\text{parent}}$ is any unassigned primitive from the parent-BV set, $V(\mathcal{B})$ is the volume of the BV bounding the primitive set $\mathcal{B}$, $\alpha_{\text{VMT}}(\mathcal{B})$ the opening angle of its NBC. To be able to mix volumes and angles, the constant $c$ is introduced, for which $2/\pi$ was determined to be a good value. The child-primitive sets $\mathcal{B}_{\text{child}_i}$ are initialised with the two primitives in the parent whose centroids are the farthest apart.

Higher-order BVHs are constructed by performing two or more of the recursive partioning and merging steps, described above, within one iteration of the construction loop.

#### 5.3.4.3 BVH Updating

The BVH updating algorithm only refits bounding volumes to accommodate the deformation undergone by the geometry during a time step; it does not make any changes to the BVH topology. This selective updating is achieved by means of *update nodes* (UN). The UNs are defined as subtree roots in which the deformation undergone by the bounded geometry is quantified to assess the need for an update of the respective BVH subtree before the next collision detection pass. An update of the subtree is required, if 1) there are potential self-collisions in the subtree, or 2) the geometry has moved so much that the bounds of the subtree's BVs are no longer valid. The top part of the BVH, i.e. the part above the UN, is updated in

[1] http://www.ircad.fr/softwares/3Dircadb/3Dircadb.php?lng=en

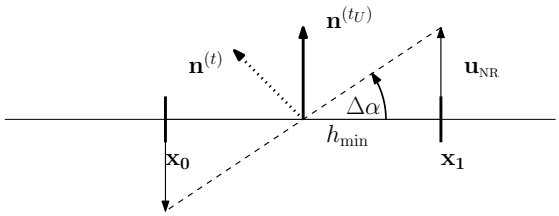



Figure 5.4: In 2D: Situation leading to the largest possible change to the primitive normal for a non-rigid deformation of known magnitude $\|\boldsymbol{u}_{\mathrm{NR}}\|$. The nodes of the example primitive have moved in opposite directions and perpendicularly to the primitive's previous plane.

every time step.

In order to evaluate criterion 1), a bound on the *non-rigid* deformation the geometry can undergo without causing an expansion of the NBC opening angle $\alpha_{\mathrm{VMT}}$ beyond the threshold $\tau_{\mathrm{VMT}}$ is required. The bound introduced here, can be derived with a 2D sketch (Fig. 5.4): Assuming the maximum of the non-rigid displacement $\boldsymbol{u}_{\mathrm{NR}}$ over the bounded nodes is known, taking the smallest primitive diameter $h_{\min}$ in the set of primitives bounded by the UN, it is found that the biggest change to the primitive's normal (and thus potentially all NBCs in which it is contained) occurs if the primitives vertices both move by $\max\|\boldsymbol{u}_{\mathrm{NR}}\|$ in opposite directions and perpendicularly to the plane of the primitive. The angular change to the normal $\Delta\alpha$ arising from such non-rigid deformation is

$$\Delta\alpha = \arctan\frac{\max\|\boldsymbol{u}_{\mathrm{NR}}\|}{h_{\min}/2} \tag{5.8}$$

The next step is to solve for the desired threshold $\tau_{\mathrm{NR}}$ on the non-rigid displacement magnitude $\|\boldsymbol{u}_{\mathrm{NR}}\|$ by setting $\Delta\alpha$ to $(\tau_{\mathrm{VMT}} - \alpha_{\mathrm{VMT}}^{(t_U)})/2$. This yields the following formula:

$$\tau_{\mathrm{NR}} = \frac{h_{\min}}{2}\tan\frac{\tau_{\mathrm{VMT}} - \alpha_{\mathrm{VMT}}^{(t_U)}}{2} \tag{5.9}$$

Finding the the minimum primitive diameter $h_{\min}$ can be done recursively as part of the subtree update and at virtually no cost. The computation of the actual non-rigid displacement magnitude $\|\boldsymbol{u}_{\mathrm{NR}}\|$ requires Procrustes analysis and is significantly more costly. However, the information obtained in the Procrustes analysis, can be used to very cheaply update the subtree if update criterion 2) is satisfied but not 1), and the BV type supports such rigid body transforms, e.g., oriented bounding boxes [46]. This gives rise to the update algorithm, Algorithm 5.2. The term $\epsilon_r$, appearing in the pseudo-code, will be explained in Section 5.3.5.

The UN role is assigned to BVs at the time of BVH creation. Since subtrees potentially containing self collisions always have to be updated, it makes sense to place the update nodes well below a point in the tree where $\alpha_{\mathrm{VMT}} > \tau_{\mathrm{VMT}}$. The algorithm for the placing is a greedy one, initialising the set of UNs with the leafs of the BVH. In every iteration, it picks the two nodes whose parent has the narrowest NBC opening angle, and replaces them with their parent in the intermediate set of update nodes. This procedure continues until the minimum value

**Algorithm 5.2** BVH update algorithm

**for all** $\text{bv}_{\text{sub}} \in$ {Update nodes} **do**
  **if** $\alpha_{\text{VMT}}(\text{bv}_{\text{sub}}) \geq \tau_{\text{VMT}}$) **then**
    {Subtree has potential self-collisions}
    `RefitTopDown`($\text{bv}_{\text{sub}}$)
    `RefitBottomUp`($\text{bv}_{\text{sub}}$)
    $\boldsymbol{x_i}^{(t_U)} \leftarrow \boldsymbol{x_i}^{(t)}$ {Update reference nodes}
  **else**
    $\boldsymbol{c}^{(t)} \leftarrow \frac{1}{|\mathcal{N}(\text{bv}_{\text{sub}})|} \sum_{i \in \mathcal{N}(\text{bv}_{\text{sub}})} \boldsymbol{x_i}^{(t)}$ {Compute new centroid of bounded mesh nodes}
    $\boldsymbol{t} \leftarrow \boldsymbol{c}^{(t)} - \boldsymbol{c}^{(t_U)}$ {Compute subtree translation}
    $\max \|\boldsymbol{u}_{\text{NT}}\| \leftarrow \max_{i \in \mathcal{N}(\text{bv}_{\text{sub}})} \|\boldsymbol{x_i}^{(t)} - \boldsymbol{x_i}^{(t_U)} - \boldsymbol{t}\|$ {Compute maximum non-translational motion}
    **if** $\max \|\boldsymbol{u}_{\text{NT}}\| < \tau_{\text{NR}}$ **and** $\max \|\boldsymbol{u}_{\text{NT}}\| + \|\boldsymbol{t}\| < \epsilon_{BV} - 2\epsilon_r$ **then**
      {Subtree inactive; motion insignificant, no update}
      `RefitBottomUp`($\text{bv}_{\text{sub}}$)
      **continue**
    **end if**
    **if** $\max \|\boldsymbol{u}_{\text{NT}}\| < \tau_{\text{NR}}$ **and** $\max \|\boldsymbol{u}_{\text{NT}}\| < \epsilon_{BV} - 2\epsilon_r$ **then**
      {Translational motion dominant}
      `ApplyTranslationTopDown`($\text{bv}_{\text{sub}}$, $\boldsymbol{t}$)
      $\boldsymbol{x_i}^{(t_U)} \leftarrow \boldsymbol{x_i}^{(t_U)} + \boldsymbol{t}$ {Translate reference nodes}
      `RefitBottomUp`($\text{bv}_{\text{sub}}$)
      **continue**
    **end if**
    $R \leftarrow$ `Procrustes`$(\boldsymbol{x_i}^{(t_U)} - \boldsymbol{c}^{(t_U)}, \boldsymbol{x_i}^{(t)} - \boldsymbol{c}^{(t)})$ {Compute subtree rotation with Procrustes analysis}
    $\max \|\boldsymbol{u}_{\text{NR}}\| \leftarrow \max_{i \in \mathcal{N}(\text{bv}_{\text{sub}})} \|\boldsymbol{x_i}^{(t)} - \boldsymbol{c}^{(t)} - R(\boldsymbol{x_i}^{(t_U)} - \boldsymbol{c}^{(t_U)})\|$ {Compute max. non-rigid deformation}
    **if** `DoesSupportRotation`($\text{bv}_{\text{sub}}$) **and** $\max \|\boldsymbol{u}_{\text{NR}}\| < \tau_{\text{NR}}$ **and** $\max \|\boldsymbol{u}_{\text{NR}}\| < \epsilon_{BV} - 2\epsilon_r$ **then**
      {Rigid motion}
      `ApplyTransformTopDown`($\text{bv}_{\text{sub}}$, $R$, $\boldsymbol{t}$)
      $\boldsymbol{x_i}^{(t_U)} \leftarrow R(\boldsymbol{x_i}^{(t_U)} - \boldsymbol{c}^{(t_U)}) + \boldsymbol{c}^{(t)}$ {Transform reference nodes}
    **else**
      `RefitTopDown`($\text{bv}_{\text{sub}}$)
      $\boldsymbol{x_i}^{(t_U)} \leftarrow \boldsymbol{x_i}^{(t)}$ {Update reference nodes}
    **end if**
    `RefitBottomUp`($\text{bv}_{\text{sub}}$)
  **end if**
**end for**

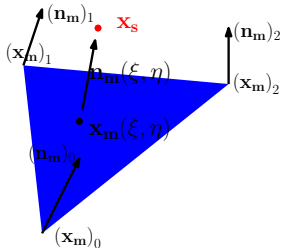



Figure 5.5: Illustration of the slave-to-master projection with the master facet drawn in blue and the slave node shown as a red dot.

of $\alpha_{\text{VMT}}$ of the set of parent BVs of the current UN exceeds a threshold, set to $\frac{1}{2}\tau_{\text{VMT}}$ in the implementation.

#### 5.3.4.4 Collision Detection

The broad-phase collision detection is performed by recursively checking BVH (sub-) trees against each other until the bottom of the two BV trees is reached, i.e. the BVs bounding individual geometric primitives [96]. For self-collision detection, the children of any BVH node where Eq. (5.3) holds true need to be checked against each other. The subsequent primitive-primitive test consists of one test for vertices against facets and one for edges against edges. The node-facet test starts with the computation of an initial projection of the predicted position of the slave vertex $\boldsymbol{x_s}$ onto the master surface and a corresponding gap value, $\left(\tilde{\xi}, \tilde{\eta}, \tilde{g}\right)$. This initial projection is obtained with Möller and Trumbore's method [102] with the minor modification of employing normalised facet normals instead of unnormalised one. This particular method is only applicable with triangular surface discretisations.

If the initial-guess projection $(\tilde{\xi}, \tilde{\eta})$ lies within the bounds of the master facet and the initial guess for the gap value $|\tilde{g}|$ is sufficiently close to the previously nearest projection, it is improved with an iterative procedure that employs *C0*-continuous master facet normals $\boldsymbol{n_m}(\xi, \eta)$, computed from the vertex normals obtained by averaging the normals of the incident facets. This yields the final penetration depth (gap function value) $g$ and projection $\boldsymbol{x_m}(\xi, \eta)$:

$$\boldsymbol{x_m}(\xi,\eta) := \sum_{i \in \{\text{master facet vertices}\}} b_i(\xi,\eta)\boldsymbol{x_i} \quad \Rightarrow \quad g := \boldsymbol{n_m}^{\mathrm{T}}(\xi,\eta)(\boldsymbol{x_s} - \boldsymbol{x_m}(\xi,\eta)) \tag{5.10}$$

Where $\boldsymbol{x_i}$ denotes the coordinates of the $i$-th master facet vertex, and $b_i(\xi, \eta)$ is the corresponding standard 2D linear shape function. An illustration of these quantities and their relationships is given in Fig. 5.5. The vertex normals underlying $\boldsymbol{n_m}$ are computed from

$$(\boldsymbol{n_m})_i = \frac{1}{\| \sum_{j \in \mathcal{T}_i} \alpha_{j,i} \, (\tilde{n}_m)_j \|} \sum_{j \in \mathcal{T}_i} \alpha_{j,i} \, (\tilde{n}_m)_j \tag{5.11}$$

Where $\mathcal{T}_i$ is the set of primitives incident on the master-surface node $i$, $\alpha_{j,i}$ the opening angle of the primitive $j$ at said node, and $(\tilde{n}_m)_j$ the unnormalised, facet normal of the $j$-th primitive. (5.11) is a commonly used approximation in computer graphics; alternative formulations for vertex normals can be found in [147].

The projection $\boldsymbol{x_m}(\xi, \eta)$ is the virtual master-surface node based on which the contact forces are computed (Section 5.3.5). For each slave node only the nearest projection onto a master facet is stored.

The edge-edge collision detection is performed by determining for the potentially colliding edge-edge pairs turned up by the broad-phase search the points on the two edges with the smallest distance in the predictor configuration. This yields the parameters $q$ and $r \in [0, 1]$ that represent the position of the closest point on the slave and the master edge, respectively. The difference between those closest points is subsequently projected onto the master surface normal, resulting in the gap function for the edge-edge case:

$$g = \boldsymbol{n_m}^{\mathrm{T}}(r)(\boldsymbol{x_s}^{(t)}(q) - \boldsymbol{x_m}^{(t)}(r)) \tag{5.12}$$

### 5.3.5 Contact-Force Calculations

Two types of contact forces are distinguished: penetration responses and gap rate-proportional forces. The former come into effect if a predictor displacement configuration leads to intersections of master and slave surfaces. The latter slow down approaching master and slave surfaces before they can intersect. The derivation of the force formulations started from the work of Heinstein et al. [57]. Modifications on the side of penetration responses include the distribution of forces over the master surface, the extension to the edge-edge collision case, the momentum-preserving gap-partitioning factor formulation, and the force consolidation algorithm. The gap rate-proportional forces that in practice constitute the bulk of the contact forces are a new formulation. Another major objective of the presented formulation is to, as far as possible, enforce contact constraints one-by-one and avoid the introduction of any matrix formulations in the force computation step, so as to keep the required communication between workers processing individual contacts in parallel implementations at a minimum.

#### 5.3.5.1 Penetration Response Calculation

Penetrations are mathematically characterised by $g < 0$, i.e. situations where the slave node lies behind the master-surface facet with respect to the master surface normal direction. The penetration response force formulas arise directly from (5.1) with the requirement of immediate resolution of any mesh intersection and, for *node-facet* penetrations, are given by

$$\begin{aligned} \boldsymbol{f_s} &= -\boldsymbol{n}(\xi, \eta)\beta_s \tfrac{m_s g}{\Delta t^2} \\ (\boldsymbol{f_m})_i &= \boldsymbol{n}(\xi, \eta)\beta_m \tfrac{m_i g \gamma_i(\xi,\eta)}{\Delta t^2}, \quad i \in \text{master facet} \end{aligned} \tag{5.13}$$

$\boldsymbol{f_s}$ is the force applied to the penetrating slave node, $(\boldsymbol{f}_m)_i$ denotes the force applied to one of the three vertices of the penetrated master-surface primitive. $m_s, m_i$ are the masses of the slave node and master-facet nodes, respectively.

The factor $\gamma_i$

$$\gamma_i(\xi, \eta) := \frac{b_i(\xi, \eta)}{\sum_{j \in \{\text{master facet vertices}\}} b_j(\xi, \eta)^2}, \quad i \in \text{master facet} \tag{5.14}$$

distributes the gap function value over the master facet such that at the position $\boldsymbol{x_m}$ the fraction of the gap assigned to the master surface is recovered [85]:

$$\beta_m g \quad = \sum_{i \in \{\text{master facet vertices}\}} b_i(\xi, \eta)\gamma_i(\xi, \eta)\beta_m g \tag{5.15}$$

The gap partitioning factor $\beta$, appearing in (5.13), controls how the response is split between master and slave surfaces. For contacts between rigid and deformable objects it is set to 0 and 1 respectively. For inter-penetration of deformable bodies it holds a value in $]0, 1[$ that is computed from the masses of the nodes involved in the contact:

$$\beta_s = \frac{m_m}{m_s + m_m}, \quad \beta_m = 1 - \beta_s = \frac{m_s}{m_s + m_m} \tag{5.16}$$

For the purpose of gap partitioning, the mass of the virtual master node $m_m$ is computed with linear interpolation from the corresponding facet-vertex masses.

With these definitions, it holds at the point of contact on the master surface:

$$\begin{aligned} \boldsymbol{f_m} \quad &= \sum_{i \in \{\text{master facet vertices}\}} b_i(\xi, \eta)(\boldsymbol{f_m})_i \\ &= -\boldsymbol{f_s} = -\lambda \boldsymbol{n} \end{aligned} \tag{5.17}$$

where $\lambda$ is the Lagrange-multiplier for the constraint.

*Edge-edge penetration responses* employ the same rationale that underlies eq. (5.13), except that there are now two slave nodes and only two master nodes, and the $\gamma$ factor (5.14) is now computed with 1D shape functions instead of 2D ones.

$$\begin{aligned} (\boldsymbol{f_s})_i \quad &= -\boldsymbol{n_m}(r)\beta_s \tfrac{m_i \gamma_i(q) g}{\Delta t^2}, \quad i \in \text{slave edge} \\ (\boldsymbol{f_m})_i \quad &= \boldsymbol{n_m}(r)\beta_m \tfrac{m_i \gamma_i(r) g}{\Delta t^2}, \quad i \in \text{master edge} \end{aligned} \tag{5.18}$$

### 5.3.5.2 Gap Rate-Proportional Forces

The gap-rate proportional forces are employed to achieve a degree of temporal smoothing in the contact forces and thus to improve stability. They come into effect when $0 \leq g < \epsilon_r$ and the *gap rate*, the relative velocity between slave and master, in master normal direction, is negative:

$$\begin{aligned} &\boldsymbol{n_m}^{\mathrm{T}}(\boldsymbol{v_s} - \boldsymbol{v_m}(\xi, \eta)) < 0 \\ &\boldsymbol{v_s} := (\boldsymbol{x_s}^{(t)} - \boldsymbol{x_s}^{(t-\Delta t)})/\Delta t, \quad \boldsymbol{v_m}(\xi, \eta) := (\boldsymbol{x_m}^{(t)}(\xi, \eta) - \boldsymbol{x_m}^{(t-\Delta t)}(\xi, \eta))/\Delta t \end{aligned} \tag{5.19}$$

The constant $\epsilon_r = \frac{5}{100 \cdot 2}\epsilon_{BV}$ is chosen such that any node at distance $\epsilon_r$ from a master facet still lies within the safety margin of the BV and so close that any effects of the force applications are not visible in the final configuration.

The force required for velocity matching of the slave node and the virtual master node are derived from the forward-Euler increment of the velocity and momentum conservation as follows:

$$\begin{aligned} \boldsymbol{n_m}^{\mathrm{T}}\left[(\boldsymbol{v_s} - \boldsymbol{v_m}) + (\boldsymbol{\Delta v_s} - \boldsymbol{\Delta v_m})\right] \quad &= 0 \\ \boldsymbol{\Delta v_s} = \tfrac{\Delta t}{m_s}\boldsymbol{n_m}\dot{\lambda}, \quad \boldsymbol{\Delta v_m} \quad &= -\sum_{i \in \text{master facet}} b_i \tfrac{\Delta t}{m_i}\gamma_i \boldsymbol{n_m}\dot{\lambda} \end{aligned} \tag{5.20}$$

This gives rise to the following formula for the force's magnitude $\dot{\lambda}$

$$\dot{\lambda} = -\frac{\boldsymbol{n_m}^{\mathrm{T}}(\boldsymbol{v_s} - \boldsymbol{v_m})}{\Delta t(1/m_s + \sum_i b_i \gamma_i / m_i)} \tag{5.21}$$

The applied force gradually increases as the distance between the surfaces decreases, and full velocity-matching is performed when there is zero distance between the slave node and its projection onto the master surface

$$\boldsymbol{f_s} = (1 - g/\epsilon_r)\boldsymbol{n}(\xi, \eta)\dot{\lambda}, \quad (\boldsymbol{f}_m)_i = -(1 - g/\epsilon_r)\boldsymbol{n}(\xi, \eta)\dot{\lambda}\gamma_i \tag{5.22}$$

The edge-edge contact formula for $\dot{\lambda}$ reads:

$$\dot{\lambda} = -\frac{\boldsymbol{n_m}^{\mathrm{T}}(\boldsymbol{v_s} - \boldsymbol{v_m})}{\Delta t\left(\sum_{i\in\text{master edge}} b_i\gamma_i/m_i + \sum_{i\in\text{slave edge}} b_i\gamma_i/m_i\right)} \tag{5.23}$$

#### 5.3.5.3 Friction

Equation (5.21) can be used in Coulomb's model to simulate friction by substituting the relative tangential velocity for the gap rate. Modelling friction only requires keeping track of the active constraints in a given time step and the associated normal forces, and application of the forces computed from

$$\boldsymbol{f} = \begin{cases} -\lambda_T \boldsymbol{\Delta v_T}/\|\boldsymbol{\Delta v_T}\|, \text{ if } \lambda_T < \mu\|\boldsymbol{f_N}\| \\ \mu\|\boldsymbol{f_N}\|\boldsymbol{\Delta v_T}/\|\boldsymbol{\Delta v_T}\|, \text{ otherwise} \end{cases} \tag{5.24}$$

with $\mu$ being the friction coefficient, $\boldsymbol{f_N}$ the normal forces applied to the corresponding slave node, and

$$\begin{aligned} \boldsymbol{\Delta v_T} &:= (\boldsymbol{v_s} - \boldsymbol{v_m}) - \boldsymbol{n_m}^{\mathrm{T}}(\boldsymbol{v_s} - \boldsymbol{v_m}) \cdot \boldsymbol{n_m} \\ \lambda_T &:= \frac{\|\boldsymbol{\Delta v_T}\|}{\Delta t(1/m_s + b_i\gamma_i/m_i)} \end{aligned} \tag{5.25}$$

#### 5.3.5.4 Contact-Force Consolidation

Since nodes can be involved in multiple contacts, e.g. multiple edge-edge contacts or master-facet vertices being part of multiple node-facet contacts, the contact forces must be consolidated. This can be relatively easily accomplished if the indices subject contact constraints are being kept track of as the responses are computed. The chosen option is that of computing for every node with an active contact constraint the mean direction of the contact forces and applying the maximum projection over all response forces in that direction. Algorithm 5.3 contains a pseudo-code description of the consolidation algorithm.

**Algorithm 5.3** Algorithm for handling of redundant constraints.

**for all** $n \in$ {nodes with contacts} **do**

$\overline{\boldsymbol{f}} = \sum_{\boldsymbol{f}\in\{\text{contact forces applied to } n\}} \boldsymbol{f}/\|\sum_{\boldsymbol{f}\in\{\text{contact forces applied to } n\}} \boldsymbol{f}\|$ {Compute (weighted) mean direction of contact forces}

$f_n \leftarrow \max_{\boldsymbol{f}\in\{\text{contact forces applied to } n\}} \boldsymbol{f}^{\mathrm{T}}\overline{\boldsymbol{f}}$ {Compute maximum projection along mean direction.}

$\boldsymbol{f_n} \leftarrow f_n\overline{\boldsymbol{f}}$

**end for**

With the right data structures, this algorithm can be implemented with an $O(N_{\text{node-facet}} + N_{\text{edge-edge}})$ runtime complexity, where $N_{\text{node-facet}}$ and $N_{\text{edge-edge}}$ are the number of node-facet and edge-edge contacts, respectively.

### 5.3.6 GPU Implementation

Since branching and random memory accesses are significantly more costly on GPUs, certain modifications need to be made to the proposed pipeline to efficiently run it on GPUs. Minor optimisations made in the CPU implementation, such as caching of surface normals and surface projection operators, do not have the expected performance benefits and frequently lead to unelegant code and even performance penalties. However, a number of more fundamental changes to the pipeline described above have been made in the GPU implementation presently in NiftySim, and are discussed in this section. Some more technical aspects of the implementation of the GPU contact modelling pipeline are outlined, as well.

#### 5.3.6.1 Contact Search

The contact search is still performed based on BVHs, however, instead of the pair-wise descent of the CPU implementation, the GPU implementation tests all leafs of the slave BVH (sub-)tree against the root of the master hierarchy, and then proceeds by iteratively filtering out slave BVs that are not intersecting with the master BVH and descend in the master BVH. This is motivated by the fact that an implementation based on pairwise descent only leads to very few active threads per block, or requires elaborate load balancing [84]. Per iteration of the algorithm two kernels are invoked: 1) an intersection test, testing all master/slave BV pairs currently in the list for BV intersections; 2) a descent kernel, replacing every remaining BV pair after the intersection test with a series of pairs containing the same slave BV and the children of the master BV, if there are any. If the master BV is a leaf, no replacement takes place. The algorithm terminates when there are no more new BV pairs, i.e., all master BVs in the list are leafs, or after a number of iterations corresponding to the maximum depth of the master BVH. The BV pairs remaining after the last intersection test form the starting point for the narrow phase search. Since the number of BV pairs can potentially grow as the algorithm descends in the master BVH, a readback of the number of output pairs of the second stage is required and potentially a resizing of the output buffer. By anticipating the maximum possible number of pairs resulting in the next iteration based on the last number read back, and using CUDA's stream mechanism this readback can be performed concurrently with the current-iteration descent.

The narrow-phase contact search requires a series of distinct kernels to be invoked, also, to perform the following tasks:

- Determine for all intersecting leaf-BV pairs all potentially colliding slave-node/master-facet and all edge-edge pairs.
- For all node-facet pairs: Project all nodes onto the corresponding master facet, and filter out any pairs with nodes that are too far in positive direction from the master facet or whose projection lies far outside the master facet.
- For all edge-edge pairs: Compute the parameters of the contact point on both the slave and the master edge, and filter out any non-intersecting edges.

- For all remaining node-facet pairs: Compute projections with C0-surface normals (Fig. (5.5)), and repeat the checks for collision.
- For all remaining edge-edge pairs: Compute the signed distance between the contact points and check for contact (i.e. $g < \epsilon_r$)
- Sort all remaining node-facet/edge-edge pairs and remove any duplicates.

Node-facet and edge-edge narrow-phase collision tests can be performed concurrently using CUDA streams.

#### 5.3.6.2 BVH Refitting

While the algorithm described in Section 5.3.4.3 could be implemented on the GPU quite efficiently, and some of its components such as the deformation analysis are likely to greatly benefit from the parallelism awarded by GPUs, the extra kernel invokations and the readback of the GPU memory holding the result of the motion analysis, more so, can only be amortised with very dense meshes. Therefore, the efficient exhaustive refitting of BVHs was prioritised in the implementation. A method without any need for device to host memory copies can be achieved if the dependences for any given BV are computed offline and an iterative updating of all BVs without un-updated child BVs is performed, giving rise to Algorithm 5.4.

**Algorithm 5.4** GPU BVH refitting algorithm

($N_{\text{levels}}$, `ListBasePtr`, `mesh`) ← Input: Number of update loop iterations/levels in BVH, BVH-level BV indices, surface mesh
$N_{\text{BVH-leafs}}$ ← `RetrieveLevelSize`(1) {Retrieve the number of leaf BVs in the hierarchy.}
`InvokeKernel`(`UpdateBVHLeafs`(`ListBasePtr`, $N_{\text{BVH-leafs}}$, `mesh`)) {Refit the leaf BVs to the current geometric configuration.}
`ListPtr` ← `ListBasePtr` + $N_{\text{leaf-BVs}}$
**for** $l = 2$ **to** $N_{\text{levels}}$ **do**
  $N_{\text{level-BVs}}$ ← `RetrieveLevelSize`($l$) {Determine size of current BV level.}
  `InvokeKernel`(`UpdateInteriorBVHLevel`(`ListPtr`, $N_{\text{level-BVs}}$))
  `ListPtr` ← `ListPtr` + $N_{\text{level-BVs}}$ {Progress to next BVH level}
**end for**

The BVH-levels updated in the iterations of the main loop in Algorithm 5.4 can be computed with Algorithm 5.5.

The number kernel invokations required for BVH refitting can be reduced by using one kernel to update the top part of the hierarchy once the level size is lower or equal to the kernel blocksize, since, from that point on, a loop containing a synchronisation barrier can fully update the following levels of the hierarchy without causing any race conditions.

This approach is agnostic to the BVH topology and the BVH being a full hierarchy or a BVH subtree. Further, by substituing the leaf update routine in Algorithm 5.4 it could also be

**Algorithm 5.5** Algorithm for the pre-computation of hierarchy level-wise BV dependences.

```
IndexList ← GetSubtreeLeafBVIndices(bvh, HierarchyRootIndex)
IsUpdated ← (0, ··· , 0) {Mark all BVs in the hiearchy as un-visited}
VisitNext ← ∅
LevelBounds ← (0)
for all i ∈ IndexList do
  IsUpdated_i ← 1 {Mark leafs as visited}
  VisitNext ← VisitNext ∪ {GetParentBV(i)} {Insert all BVs one level up from leafs as
  candidates for next BVH level.}
end for
while VisitNext ≠ ∅ do
  VisitedLast ← ∅
  LevelBounds ← (LevelBounds, |IndexList|) {Insert the index offset of the current hiearchy
  level in the index bounds list}
  VisitNow ← VisitNext
  for all i ∈ VisitNow do
    if CanUpdate(i, IsUpdated) then
      VisitedLast ← VisitedLast ∪ {i}
      IndexList ← (IndexList, i)
      if GetParentBV(i) ≠ HiearchyRootIndex then
        VisitNext ← VisitNext∪{GetParentBV(i)} {Since the current BV i has been visited,
        its parent can potentially be updated too in the next iteration, provided its other
        children have been updated.}
      end if
    end if
  end for
  IsUpdated(VisitedLast) ← 1 {Mark all BVs visited in this iteration as updated, and not
  inside the inner loop since it could lead to race conditions in Alg. 5.4}
  VisitNext ← VisitNext − VisitedLast
end while
LevelBounds ← (LevelBounds, |IndexList|)
IndexList ← (IndexList, HiearchyRootIndex)
LevelBounds ← (LevelBounds, |IndexList|)
```

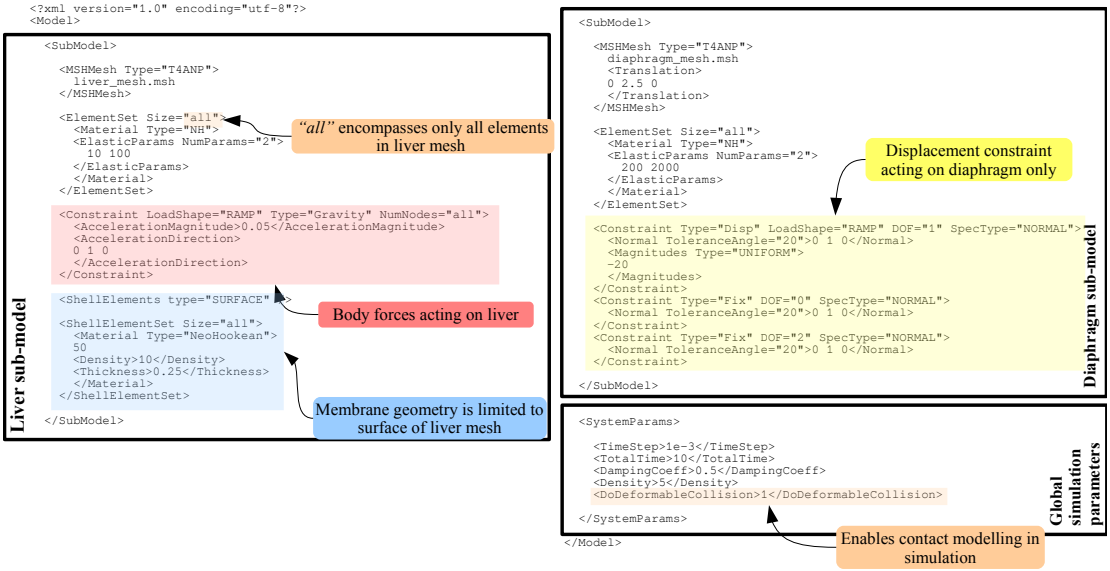



Figure 5.6: XML definition of a simulation of a liver-diaphragm contact.

used to perform the bottom-up update of the top-part of the BVH required with the algorithm described in Section 5.3.4.3.

### 5.3.6.3 Contact Force Computation

The computation of contact forces and their consolidation is comparatively close to the CPU version of the algorithm. There are two kernels that compute for all node-facet and edge-edge pairs identified by the contact search, respectively, the forces based on the procedures described in Section 5.3.5, after checking the gap function value. These kernels output for every contact a structure consisting of a force vector and the index of the node to which the force is to be applied. After the force lists have been sorted and merged, a last kernel implementing the force consolidation algorithm Algorithm 5.3 then performs the consolidation based on the sorted contact force list.

## 5.4 Usage in NiftySim

Continuing the examples from Chapter 4, the mesh-based contact modelling pipeline allows for the addition inter-body contacts. In liver IGS applications, one contact of interest is that between the liver and the diaphragm. In NiftySim, given separate meshes for these anatomical structures, this contact can be modelled with the `SubModel` XML construct and a `DoDeformable-Collision` element in the simulation's `SystemParams` block. Fig. 5.6 shows the XML definition of the example.

In the example, the diaphragm is pushed towards the liver by a displacement BC applied to its top, while the liver is propelled upward by body forces so as to prevent it from being pushed off into empty space by the diaphragm. As stated in section 4.5.3, sub-models are used to recursively define models, which means that boundary conditions defined within a sub-model are limited in their scope to the sub-model's geometry. In concrete terms, this means

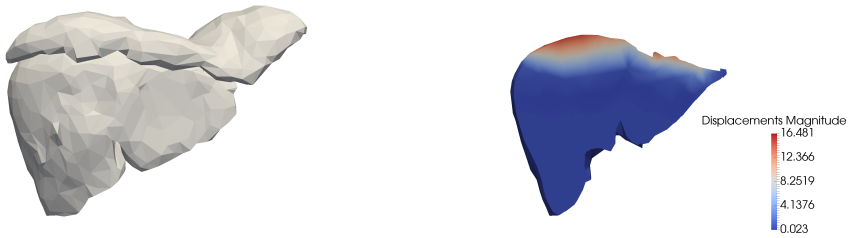



Figure 5.7: Left: inital configuration of the liver-diaphragm contact simulation defined in Fig. 5.6. Right: cross-section of the liver's final configuration.

that the geometric definition for the displacement BC defined in the diaphragm sub-model is not applied to any liver mesh facets. Similarly, the gravity constraint defined on the liver and the wrapping of the geometry in a membrane are limited to the liver sub-model, and no body forces are acting on the diaphragm geometry nor are there any membrane elements associated with its surface facets.

If post-processing of the individual bodies involved in the simulation is desired, the `-export-submeshes` switch is useful. In Fig. 5.7 this facility in conjunction with ParaView was used to produce a cross-section view of the liver's final configuration.

The second example, shown in Fig. 5.8, is a simulation of the gripping of the tip of the left lobe with surgical forceps. This is accomplished using the rigid-deformable contact modelling facilities which are accessible in XML through the `ContactSurface` tag. The example makes extensive use of NiftySim's mesh-manipulation features to create the two sides of the forceps via translation and rotation of geometry loaded from the same mesh file. While these transforms defined inside the `VTKSurface` element are applied directly after loading of the geometry, the translations defined inside the `Motion` element is motion (closing of the forceps) performed by the rigid geometry over the course of the simulation, in a linear fashion.

Fig. 5.9 shows the initial and final configurations of the simulation visualised through NiftySim's built-in visualisation facilities.

## 5.5 Experiments

The objective in this section is to show the inherent advantages of the individual heuristics introduced in this chapter over a number of alternatives. Most of the subsequent evaluations are based on a single-threaded C++ implementation of the algorithm described in the previous section. The FEM calculations were done with NiftySim's CPU solver. The timings were obtained on a workstation equipped with an Intel Core i7 2600k CPU, an nVidia geForce GTX 680 GPU, and 8GB of RAM by surrounding individual parts of the code representing the major stages of the contact modelling pipeline with calls to the C `clock` function and accumulation.

```
<?xml version="1.0" encoding="utf-8"?>
<Model>

  <MSHMesh Type="T4ANP">
    liver_mesh.msh
  </MSHMesh>

  <ElementSet Size="all">
    <Material Type="NH">
      <ElasticParams NumParams="2">
     10 500
      </ElasticParams>
    </Material>
  </ElementSet>

  <ContactSurface>
    <VTKSurface Type="T3">
      forceps.vtk

      <Translation>
     -375 45 330
      </Translation>
    </VTKSurface>

    <Motion Type="Translation">
      0 -20 0
    </Motion>

  </ContactSurface>

  <ContactSurface>
    <VTKSurface Type="T3">
      forceps.vtk

      <Rotation>
     0 0 3 180 0 0
      </Rotation>

      <Translation>
     -375 -35 330
      </Translation>
    </VTKSurface>

    <Motion Type="Translation">
      0 40 0
    </Motion>
  </ContactSurface>

  <SystemParams>
    <TimeStep>1e-3</TimeStep>
    <TotalTime>5</TotalTime>
    <DampingCoeff>0.5</DampingCoeff>
    <Density>1</Density>
  </SystemParams>

</Model>
```

Read geometry from triangular VTK PolyData file

In-simulation motion of rigid geometry

Pre-simulation rotation of VTK surface by 180° about x-axis and centre (0, 0, 3)

Pre-simulation translation of rigid geometry

Rigid contact surface definition

Figure 5.8: XML definition of a simulation of a liver-forceps contact.

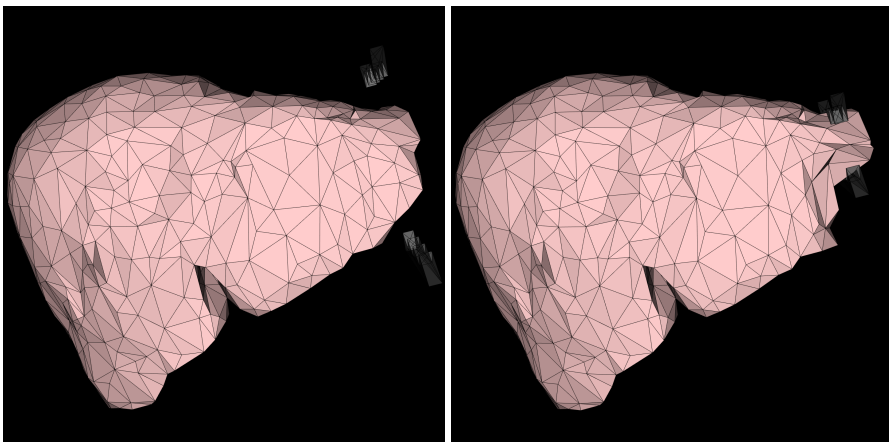


Figure 5.9: Left: inital configuration of the liver-forceps contact simulation defined in Fig. 5.8, with the two rigid surfaces appearing translucent-grey. Right: the corresponding final configuration.

Zipper: "Zipper" geometry twisted and compressed through force boundary conditions ($F_x = \pm 30,\ F_y = \pm 5,\ F_z = -10$). Initial configuration shown in left, final configuration shown in right picture.
Dimensions: $35 \times 7 \times 12.5$.
Mesh: 712 surf. facets, 358 surf. nodes, 1587 solid el's, 485 solid nodes.
Material: $G = 10$, $K = 40$, $\rho = 100$.
Total time: $T = 100$.
Time step size: $\Delta t = 0.075$.

C: C shape being closed by application of displacement boundary conditions to upper part of geometry ($u_z = -3.5$).
Dimensions: $22.5 \times 10 \times 21.5$.
Mesh: 960 surf. facets, 482 surf. nodes, 2320 solid el's, 667 solid nodes.
Material: $G = 10$, $K = 40$, $\rho = 1$.
Total time: $T = 2$.
Time step size: $\Delta t = 0.001$.

Figure 5.10: Simulation geometry and settings used in validation of the proposed cone formula.

### 5.5.1 Self-Collision Detection Cones

The first two experiments are aimed at quantifying the effects of the formula for computation of the NBCs described in Section 5.3.4.1. To this end, it is compared to the method of Provot [114]. The geometry and simulation settings of the simulations are given in Fig. 5.10. Due to the artificial nature of the geometry, units have been omitted, but all parameter values can be assumed to be specified in compatible units, e.g. m, s, Pa, etc. The geometry was generated with AutoCAD[2] and solid meshing was done with GMSH. Most of the experiments are once run with a binary AABB hierarchy (AABB2) and once with a 4-nary BVH (AABB4).

The results in terms of the average number of BVH subtrees that need checking for self-collisions per time step, BV refittings, and the total time spent updating BVHs and searching for contacts, for these two simulations are given in Table 5.1.

A reduction of the order of 40% in the number of BVH subtrees that require testing for self-collisions (first column in Table 5.1) is achieved with the proposed formula in these two test cases. Further, since the presented BVH refitting and construction algorithms take into account the NBC opening angle, the number of BVs that are refitted is about 10% lower with the proposed NBC computation method (second column in Table 5.1). The latter effect can mostly make up for the slight increase in computational costs that comes with the new formulation. These findings are consistent across the two considered BVH orders.

[2] http://usa.autodesk.com/autocad/

(a) Proposed NBC method

| Experiment | Avg. N. of BVH subtrees with potential self-collisions | Avg. N. of refitted BVs / time step (tot. N. of BVs) | Contact search total time (s) | BVH update total time (s) |
|---|---|---|---|---|
| C AABB2 | 32.9 | 216.2 (1919) | 4.35 | 0.129 |
| C AABB4 | 17.4 | 101.4 (1410) | 5.01 | 0.117 |
| Zipper AABB2 | 50.3 | 229.4 (1423) | 1.43 | 0.0936 |
| Zipper AABB4 | 22.9 | 167.3 (1065) | 1.49 | 0.1 |

(b) Provot's method

| Experiment | Avg. N. of BVH subtrees with potential self-collisions | Avg. N. of refitted BVs / time step (tot. N. of BVs) | Contact search total time (s) | BVH update total time (s) |
|---|---|---|---|---|
| C AABB2 | 57.5 | 234.5 (1919) | 5.04 | 0.113 |
| C AABB4 | 28.5 | 121.3 (1410) | 5.49 | 0.103 |
| Zipper AABB2 | 87.7 | 254 (1423) | 1.62 | 0.0823 |
| Zipper AABB4 | 45.2 | 220.7 (1065) | 1.72 | 0.0898 |

Table 5.1: Results from the comparison of the proposed cone computation method with that of Provot.

### 5.5.2 BVH Refitting Strategy

The second set of experiments deals with the evaluation of the proposed BVH updating strategy. Most of the geometry used in these experiments is again artificial and created with Meshlab[3], except the test case containing a liver and diaphragm whose geometry was extracted from volunteer MRI data with Slicer 3D[4]. The "C" test case from the previous section is also used in these experiments. The new test simulations are summarised in Fig. 5.11.

The results for these experiments can be found in Table 5.2. No results are available for Larsson and Akenine-Möller's method on the "rigid bar" test case, since the method is not defined for rigid-deformable contacts. Similarly, while technically it can be easily extended to applications in self-collision detection, its inventors never intended for it to be used in that way, and its performance is very poor and provides little insight in this context. Therefore, there are no results for the "C" test case in Table 5.2(b), either.

The first observation that can be made from these results is that the BVH updating strategy presented in Section 5.3.4.3 leads to a general reduction in BV refitting and, ultimately, in overall computation time, over both exhaustive refitting and Larsson and Akenine-Möller's update strategy. This effect is more pronounced on higher resolution meshes and problems not involving self-collisions. The latter is most likely due to the dominant effect of contact search on the overall computation time of self-collision problems. On the larger problems, the reduction in BVH refitting costs over exhaustive refitting approaches one order of magnitude with the proposed method, despite the not insignificant computational overhead introduced by the deformation analysis.

### 5.5.3 Scaling

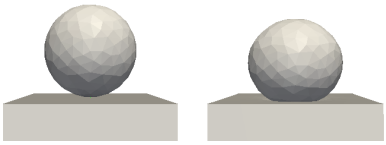
The aim here is to show the behaviour of the NBC computation formula (5.6) and BVH update method Algorithm 5.2 with increasing mesh resolution. To this end, the Zipper self-collision test case is taken and the mesh refined in 5 steps. The Zipper test case was chosen for this experiment due to the relatively large deformation and because all deformation stems from force application, there is hence no bias towards translational movement of geometry which might give the proposed method an unfair advantage.

The first experiment looks at the NBCs. The same simulation is run once with Provot's method and once with the new method, and the average number of BVH subtrees that need checking for self-collisions and the average time required for contact modelling operations per time step are recorded. The results for binary and 4-nary AABB hierarchies can be found in Fig. 5.12. The timing values include the time required for all BVH and contact surface-update, as well as collision-detection and response computations.

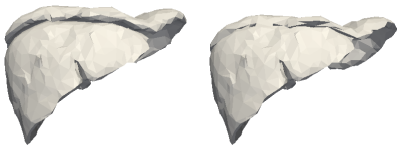
There is a clear divergence in the number of sub-trees that need checking for self-collisions between the two methods, with noticeably lower computation times for the proposed method

[3] http://meshlab.sourceforge.net/

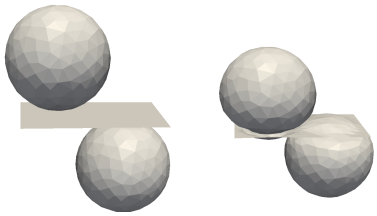
[4] http://www.slicer.org/



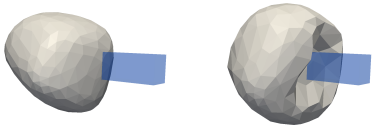
Ball-slab: Ball being pushed against a deformable slab through displacement boundary conditions (vertically by $u_z = -0.4$).
Dimensions: Ball radius = 1.2, slab: $3.15 \times 3.15 \times 1.2$.
Mesh: 1500 surf. facets, 754 surf. nodes, 3284 solid el's, 981 solid nodes.
Material: $G = 100$, $K = 400$, $\rho = 20$.
Time: $T = 2$, $\Delta t = 0.001$.



Diaphragm/liver: Diaphragm pushed against liver by displacement boundary conditions ($u_z = -0.2$).
Mesh: 2400 surf. facets, 1202 surf. nodes, 4921 solid el's, 1448 solid nodes.
Material: $G = 15$, $K = 50$, $\rho = 200$.
Time: $T = 1$, $\Delta t = 0.0025$



Two spheres: Two spheres pushed against a membrane via force constraints ($F_z = \pm 1$), leading to two-sided contact. Only membrane mid-surface shown.
Dimensions: sphere radius = 0.7, membrane dimensions = $2 \times 1 \times 0.15$.
Mesh: $2 \times 700$ sphere surf. facets, $2 \times 252$ sphere surf. nodes, $2 \times 789$ solid el's, $2 \times 278$ solid nodes, 400 membr. el's, 231 membr. nodes.
Material: $G_{\text{sphere}} = 200$, $K_{\text{sphere}} = 600$, $G_{\text{membrane}} = 1200$, $\rho_{\text{sphere}} = 25$, $\rho_{\text{membrane}} = 200$; membrane is incompressible.
Time: $T = 0.5$, $\Delta t = 10^{-4}$



Rigid bar: An irregularly shaped body (grey) is pushed via displacement boundary conditions ($u_x = 0.5$) on its back against a rigid bar (semi-transparent, blue).
Dimensions: $0.91 \times 1 \times 1.34$ (deformable body AABB), $0.5 \times 0.25 \times 0.25$ (rigid body)
Mesh: 600 surf. facets, 302 surf. nodes, 1184 solid el's, 359 solid nodes, 120 rigid facets, 83 rigid nodes.
Material: $G = 0.05$, $K = 0.1$, $\rho = 0.5$
Time: $T = 2$, $\Delta t = 0.0005$

Figure 5.11: Experiments used in BVH update-strategy validation.

(a) Proposed update strategy

| Experiment | Avg. number of refitted BVs / time step (tot. N. of BVs) | BVH update: avg. time (ms) / time step | Total computation time (s) |
|---|---|---|---|
| Ball-slab AABB2 | 111.9 (2999) | 0.0433 | 2.91 |
| Ball-slab AABB4 | 50.1 (2265) | 0.0407 | 4.02 |
| C AABB2 | 216.2 (1919) | 0.0675 | 5.73 |
| C AABB4 | 101.4 (1410) | 0.0606 | 6.34 |
| Liver-diaphragm AABB2 | 659.7 (4799) | 0.202 | 1.65 |
| Liver-diaphragm AABB4 | 412.4 (3659) | 0.196 | 1.67 |
| Two spheres AABB2 | 160.5 (3599) | 0.0488 | 5.81 |
| Rigid bar AABB2 | 75.3 (1199) | 0.0229 | 1.15 |
| Rigid bar AABB4 | 44.7 (904) | 0.0228 | 1.14 |

(b) Larsson and Akenine-Möller's update strategy

| Experiment | Avg. number of refitted BVs / time step (tot. N. of BVs) | BVH update: avg. time (ms) / time step | Total computation time (s) |
|---|---|---|---|
| Ball-slab AABB2 | 898.9 (2999) | 0.129 | 3.11 |
| Ball-slab AABB4 | 581.9 (2265) | 0.0973 | 4.14 |
| Liver-diaphragm AABB2 | 2729.6 (4799) | 0.430 | 1.77 |
| Liver-diaphragm AABB4 | 2035.7 (3659) | 0.348 | 1.67 |
| Two spheres AABB2 | 1039.1 (3599) | 0.143 | 6.31 |

(c) Exhaustive refitting

| Experiment | Avg. number of refitted BVs / time step (tot. N. of BVs) | BVH update: avg. time (ms) / time step | Total computation time (s) |
|---|---|---|---|
| Ball-slab AABB2 | 2999 (2999) | 0.301 | 3.49 |
| Ball-slab AABB4 | 2265 (2265) | 0.257 | 4.84 |
| C AABB2 | 1919 (1919) | 0.214 | 6.17 |
| C AABB4 | 1410 (1410) | 0.184 | 6.66 |
| Liver-diaphragm AABB2 | 4799 (4799) | 0.657 | 1.87 |
| Liver-diaphragm AABB4 | 3659 (3659) | 0.542 | 1.76 |
| Two spheres AABB2 | 3599 (3599) | 0.408 | 7.61 |
| Rigid bar AABB2 | 1199 (1199) | 0.146 | 1.65 |
| Rigid bar AABB4 | 904 (904) | 0.125 | 1.56 |

Table 5.2: Results of the comparison of BVH update strategies.

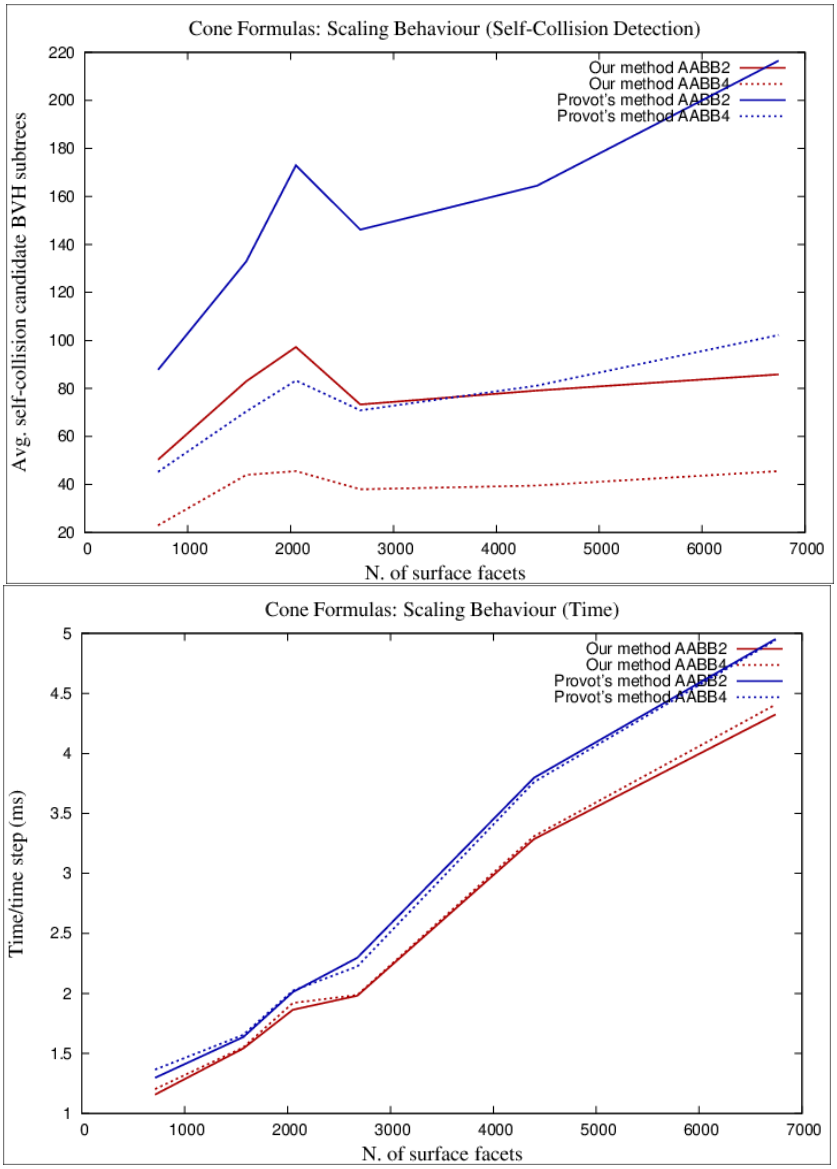



Figure 5.12: Comparison of the proposed method of NBC computation to that of Provot. Top: number of self-collision candidate subtrees plotted against mesh resolution. Bottom: corresponding contact modelling computation time per time step.

at higher mesh resolutions. That the divergence in the time required for contact modelling operations, in turn dominated by the contact search, does not diverge stronger can likely be explained with the subtree pairs that need checking with both types of NBCs having their roots higher up in the hierarchy and thus being more time-consuming to traverse.

The second experiment, looking at the proposed BVH update strategy, is mostly identical except what is recorded is the number of refitted BVs and the average per time-step BVH refitting costs in milliseconds. The results for exhaustive refitting are included for reference. The results for binary and 4-nary AABB hierarchies can be found in Fig. 5.13.

The most striking result is, as the number of surface primitives increases almost ten-fold, the number of updated BVs remains almost constant with the proposed strategy. This can be explained with the UN being placed purely according to geometric criteria. Meanwhile, the costs of exhaustively refitting the BVH approaches 50% of the total contact-modelling costs observed for the proposed algorithm in Fig. 5.12. From these plots it can also be seen that the savings in terms of overall computation time with AABB4 over AABB2 and exhaustive refitting, observable in Table 5.2, can be attributed only to the fewer refitted BVs with the higher order BVH.

### 5.5.4 Contact Forces

The next two experiments look at the quality of the contact forces by considering two geometrically well defined benchmark problems. The first experiment simulates a Newton's cradle consisting of 4 elastic spheres (Fig. 5.14). All spheres are the same size ($r = 1$), identically discretised (1004 nodes, 5101 tetrahedra), and have the same material parameters ($G = 1000$, $K = 4000$, $\rho = 10$). The leftmost ball is given an initial velocity of $\boldsymbol{v} = (0.1, 0, 0)^{\mathrm{T}}$, no other boundary conditions are applied. The simulation comprises 100,000 time steps of $\Delta t = 10^{-4}$ each for a total time of $T = 10$. The second one simulates the breaking of a billiard rack. Again, the system consists of 4 balls, one of which is given an initial velocity of $v_x = 0.15$ (Fig. 5.14), and identical material parameters are used for all 4 balls in the experiment: $G = K = 1$, $\rho = 5$. The total simulated time is $T = 20$ with 20,000 time steps. To assess the energy conservation, the time integration had to be performed with NiftySim's explicit Newmark time-ODE solver and without damping, since central difference integration, even without significant damping and in the absence of any contacts or other constraints, proved to be incapable of holding a fixed kinetic energy.

The ball-centre (average node positions) trajectories and the corresponding energy balance of the system are given in Fig. 5.15 and Fig. 5.16 for the Newton's cradle and the billiard rack experiment, respectively. The strain energies in the plots are the sum of the internal energies of all elements in the simulation and the kinetic energies are computed through the inner product defined by the lumped mass matrix:

$$E_{\mathrm{kin}} = \frac{1}{2\Delta t^2}(\boldsymbol{U}^{(t)} - \boldsymbol{U}^{(t-1)})^{\mathrm{T}}\boldsymbol{M}(\boldsymbol{U}^{(t)} - \boldsymbol{U}^{(t-1)}) \tag{5.26}$$

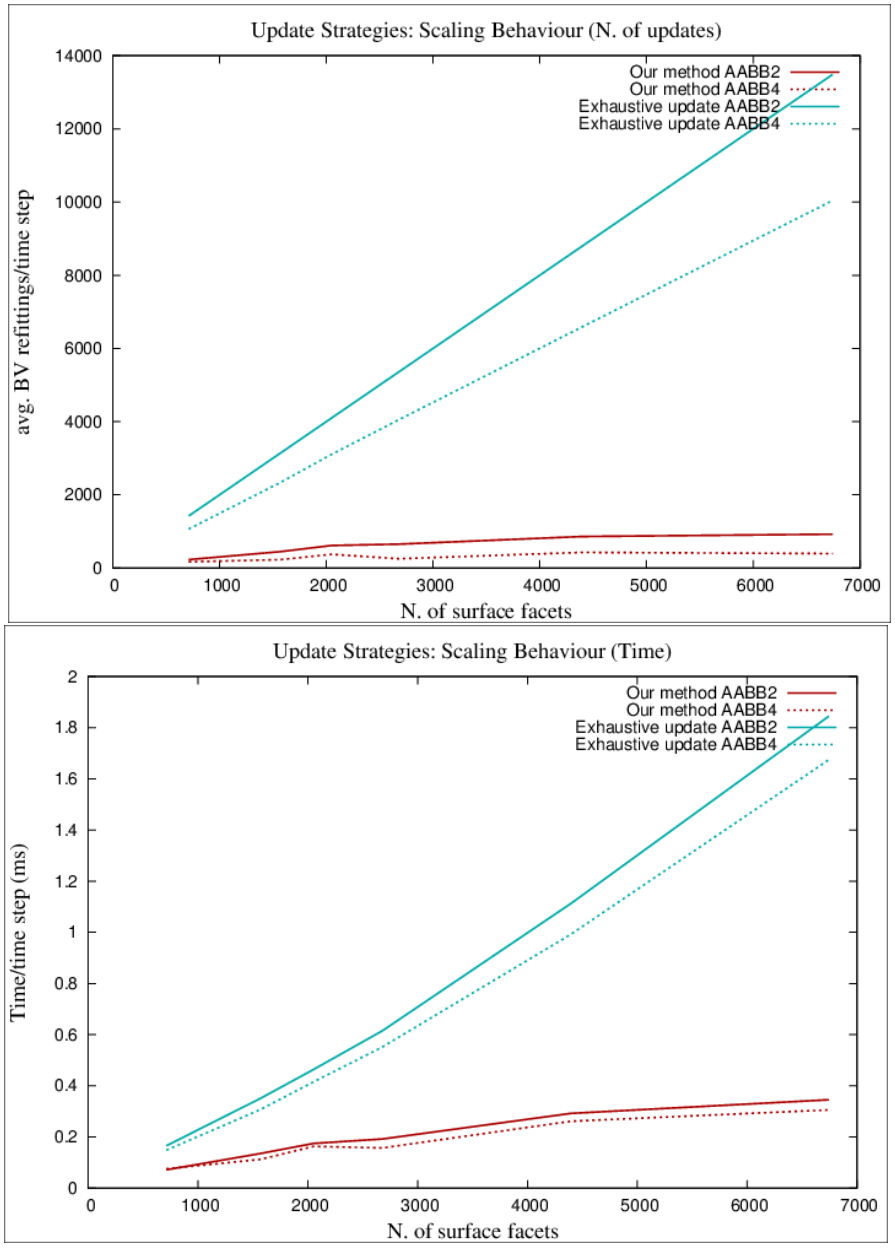



Figure 5.13: Update strategy scaling behaviour: proposed method, and exhaustive refitting. Top: average number of refitted BVs / time step. Bottom: average BVH update time / time step.

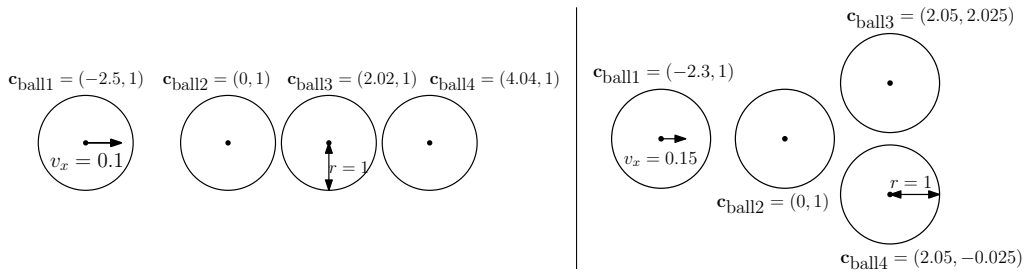



Figure 5.14: Left: Newton's cradle with 4 balls; experiment initial configuration. Right: experimental setup: breaking of a 3-ball billiard rack.

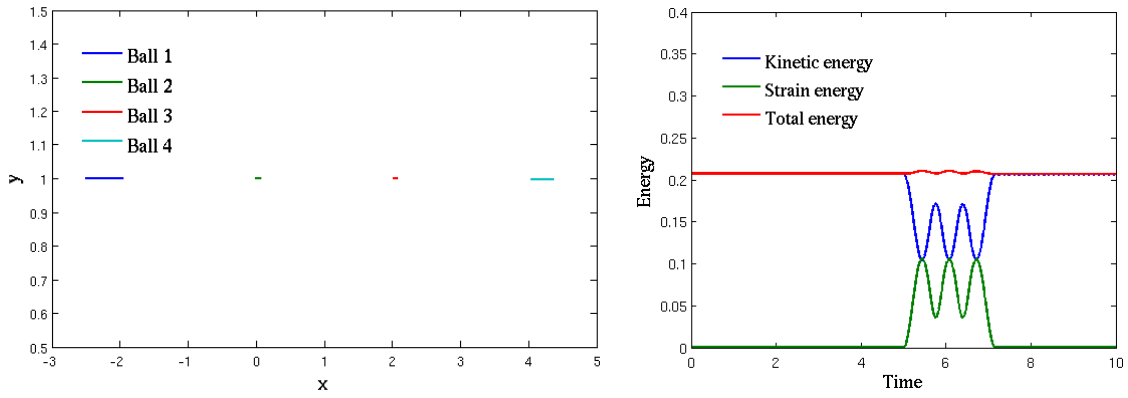



Figure 5.15: Left: ball-centre x-y-plane trajectories for the Newton's cradle experiment. Right: plot of kinetic, strain, and total energy v. time for the Newton's cradle experiment.

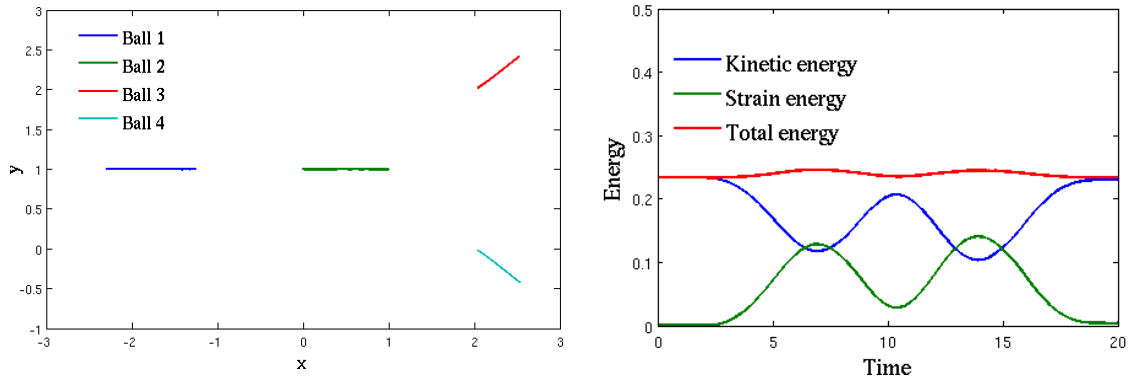



Figure 5.16: Left: ball-centre x-y-plane trajectories for the billiard experiment. Right: plot of kinetic, strain, and total energy against time for the billiard experiment.

In the Newton's cradle experiment, the ball-centre trajectories are all straight lines parallel to the $x$ axis. The standard deviations of the ball trajectories in the $y$ and $z$ directions satisfy $\sigma < 1.7{\cdot}10^{-4}$. The mean velocity of ball 4 at the end of the simulation is $v = (0.0996, 2.3{\cdot}10^{-4}, 1.2{\cdot}10^{-4})$. The total energy at the end of the simulation is 99.7% of the initial energy. In the billiard experiment, the trajectories of ball 1 and 2 form straight lines with standard deviations in the $y$ and $z$ directions satisfying $\sigma < 4.35 \cdot 10^{-4}$. For balls 3 and 4, symmetric trajectories with slopes $\pm 0.82$ are found. The respective deviation of the trajectories from that line are $\sigma_{\text{ball3}} = 3.3 \cdot 10^{-3}$ and $\sigma_{\text{ball4}} = 3.3 \cdot 10^{-3}$. The sum of all energy in the system at the end of the simulation is equal to the initial kinetic energy of ball 1. Calculated using a point mass approximation, the kinetic energy of balls 3 and 4 at the end of the simulation amounts to 96% of the initial kinetic energy of ball 1, and 1.7% is stored as strain energy in the balls.

In both experiments, the momentum and kinetic energy is almost fully transferred to the last balls in the chain. The method's ability to conserve momentum can be easily discerned in the Newton's cradle experiment, where the initial velocity of ball 1 is recovered in ball 4, at the end of the experiment. In the second experiment, the expected symmetric, linear trajectories for

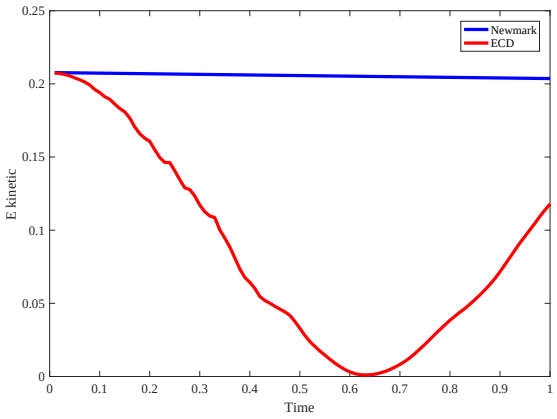



Figure 5.17: Plot of kinetic energy over time with explicit central difference (ECD) and Newmark time stepping, in a simulation without any contacts.

ball 3 and 4 are obtained. There is a small kink at the start of both trajectories, most likely caused by a short phase in the experiment during which strain energy stored in ball 2 is converted back into kinetic energy and transferred to the last two balls and during which the three balls remain in continuous contact. There is a minor increase in the total energy during phases of conversion of kinetic energy into deformation. Through experiments, the temporal smoothing heuristic and the time discretisation were excluded as the cause, but there was a correlation with the mesh density; a reduction in the number of elements per ball from 5101 to 1289 led to an increase of the highest peak from 105% the initial energy to 109%, in the billiard experiment. Further, by replacing the standard linear tetrahedral element with the average nodal pressure one, the peak could be reduced to 103%. In any case, since this excess energy is absorbed just as smoothly as it arises and its magnitude in all experiments is in the single digits of per cents, it can be assumed to be harmless.

Considering that the experiments had to be run with explicit Newmark time integration, which the contact model wasn't explicitly designed for, these results are very satisfactory.

The unusual behaviour observed when imposing an initial velocity with ECD time stepping is depicted in Fig. 5.17. The experiment used to generate the plot in Fig. 5.17 contained one body accelerated to a fixed velocity of $v_x = 0.1$ over the course of the first 10 time steps of the simulation (out of a total of 10,000). The simulation contained no other boundary conditions, nor any other bodies. In the case of ECD time stepping, the acceleration was achieved by imposing a fixed increment $\boldsymbol{\Delta U}$ on the $\boldsymbol{U}^{(n-1)}$, $\boldsymbol{U}^{(n)}$ global displacement vectors. A small velocity damping coefficient of $\alpha = 0.01$ had to be inserted to keep the simulation from diverging with ECD time stepping. The same damping was used with Newmark time stepping, as well, to ensure comparability of the results.

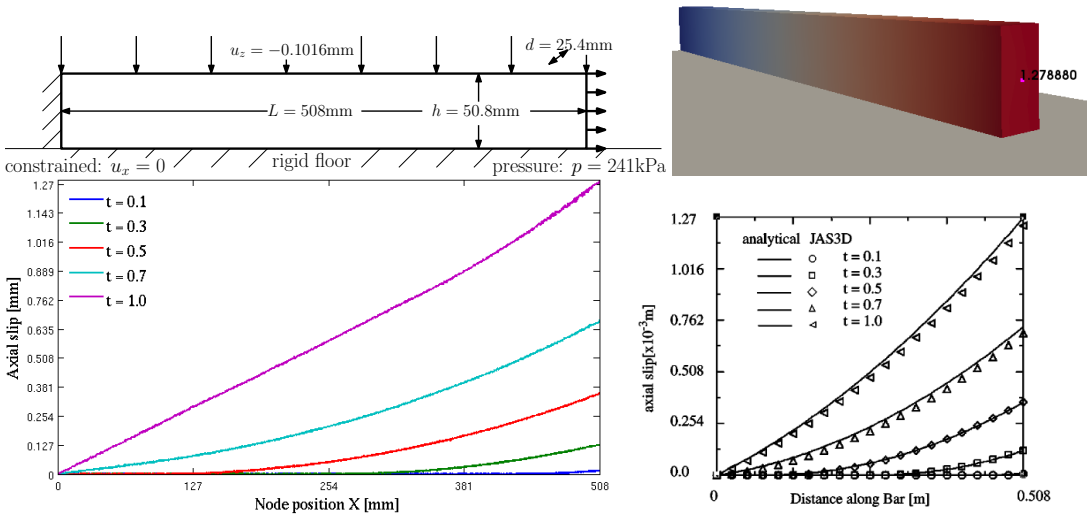



Figure 5.18: Top left: experimental setup of the friction experiment. Top right: result configuration of friction experiment with $u_x$ colour mapped and exact $u_x$ value at centre of front face. Bottom left: plot of accumulated axial slip against nodal $x$ coordinate. Bottom right: corresponding plot of axial slip found with JAS3D along with analytical solution, courtesy of ref. [56].

### 5.5.5 Friction

The following experiment is taken from ref. [56]. It is a quasi-static friction problem with an analytical solution consisting of a bar being pushed down against a rigid surface, then dragged along the same via an outward pressure applied to its front face. The geometry of the experiment and location of boundary conditions is depicted in Fig. 5.18. The bar geometry is discretised with an unstructured mesh consisting of 1678 nodes and 8419 tetrahedra. The floor consists of 400 triangles and 231 nodes in a regular grid. The material of the bar is characterised by a Young's modulus of $E = 68.947$MPa and a Poisson ratio of $\nu = 0$. The displacement boundary condition is ramped up during the first 1% of the simulation time, the pressure is applied subsequently and linearly ramped up over the 99% remaining time. The friction coefficient between the bar and the rigid surface is $\mu = 0.1$.

The final configuration is shown in Fig. 5.18. Fig. 5.18 also shows a plot of the accumulated axial slip as a function of the corresponding point's $x$ coordinate, at multiple time points in turn corresponding to different applied pressures. At most time points, the solutions for the tip displacement agrees with the analytic solution up to 5%, a larger error of 8.2% occurs at $t = 0.7$ after all but the constrained nodes have started slipping. At the end of the simulation, the measured peak displacement is $u_x = 1.287$mm, which is within 2% of the analytical solution. If the displacement threshold is set to 0.005mm, the same value of 317.5mm is recovered for the

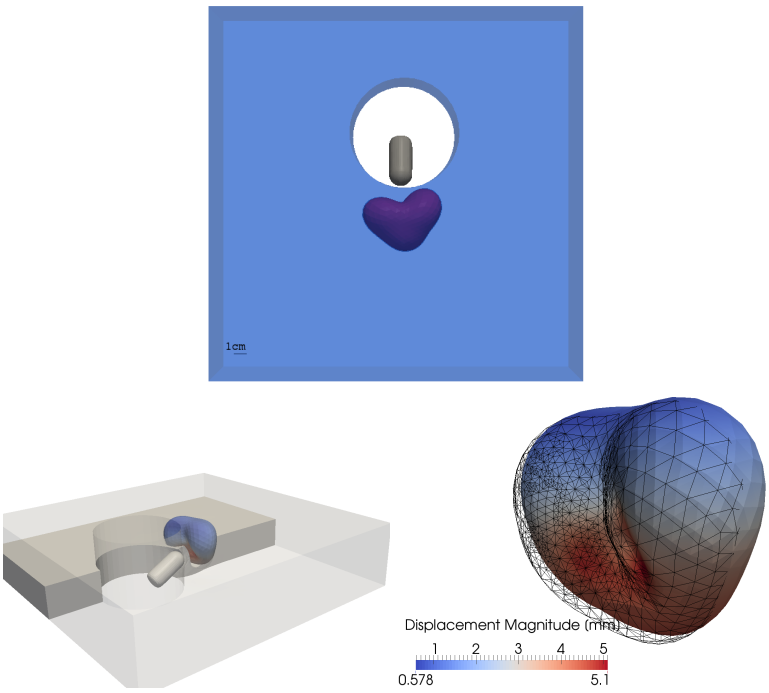



Figure 5.19: Top: initial-configuration geometry of the prostate example. Prostate shown in purple, surrounding tissue in blue, TRUS probe mesh in grey. Bottom left: cutaway view of simulation final configuration. Bottom right: prostate final-configuration posterior 3/4 view with initial configuration overlaid (wireframe).

slip/stick boundary at $t = 0.3$, as found by Heinstein and Laursen [56].

Given the relative simplicity of the friction model, the numerical result shows good agreement with the analytical solution. This result also provides evidence that the contact model is able to accurately deal with sustained contacts.

### 5.5.6 Examples from Image-Guidance

In this section two quasi-static image-guidance examples of FEM contact modelling based on actual patient data are presented with the primary aim of demonstrating the proposed algorithm's performance on high-resolution meshes encountered in TLED's main application area. The code is sequential and uses binary AABB hierarchies for contact search. The first one is a reconstruction of the deformation caused to the prostate by the transrectal ultrasound (TRUS) probe used in guidance of needle biopsy and ablation procedures of prostate cancer. Being able to determine this deformation is crucial for the registration of the interventionally acquired TRUS images to MR images acquired prior to the procedure [63]. The data used in this example comes courtesy of Dr. Y. Hu.

The anatomical meshes were generated from a 320x320x15-voxel MR image with a 0.8x0.8x4mm resolution with experimental, semi-automatic segmentation software and ANSYS[5]. The deformable geometry of the simulation consists of two unconnected parts: the

[5] http://www.ansys.com/

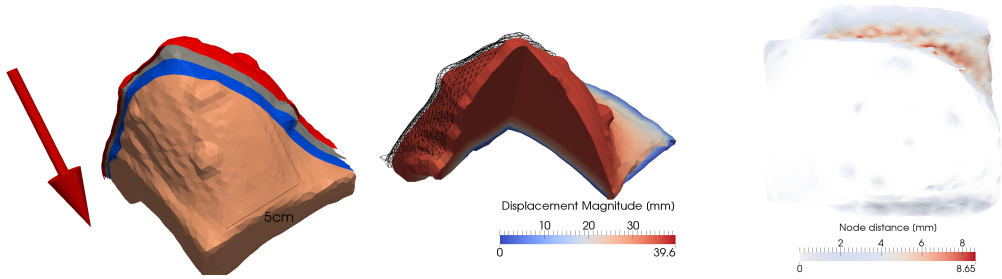



Figure 5.20: Left: initial configuration of the breast simulation with the skin contact assembly partially peeled away for better visibility, showing the outer contact surface (red), midplane (grey, not used in contact modelling), and the bottom part of the membrane contact assembly (blue), and the breast solid mesh (beige). The red arrow indicates the direction of gravity. Centre: inferior-medial view of solid mesh and skin mid-surface final configurations with the skin mid-surface shown as wireframe mesh. Right: frontal view of skinned breast simulation final result with colour and opacity mapped distances to the result of the simulation without skin.

prostate consisting of 22,705 tetrahedra and 4425 nodes (purple in Fig. 5.19), and a block representing the surrounding tissue and the rectum consisting of 64,316 elements and 13,159 nodes (semi-transparent, blue in Fig. 5.19). The TRUS probe mesh was created in Meshlab and comprises 4886 triangular elements and 2445 nodes. Hu et al. [62] randomly sampled their material parameters from ranges given by $E \in [5, 150]$kPa, $\nu \in [0.3, 0.4999]$ for the prostate components and $E \in [5, 100]$kPa and $\nu \in [0.25, 0.499]$ for the surrounding tissue, and used an inhomogeneous model for the prostate and the surrounding tissue. The parameter values chosen for this purely demonstrational simulation are identical for the prostate and the other tissue and set to $G = 1.8$kPa, $K = 6.9$kPa. The front and back face of the block are fixed in all spatial directions via displacement boundary conditions. The probe is translated by $(-0, -11.5, 5)$ mm from its initial to final position, in a linear motion. The simulation runs for a total of 1000 time steps of $\Delta t = 10^{-3}s$, each.

The second example is motivated by the registration of preoperatively acquired prone MR images used for planning of breast conserving cancer surgery to intra-operatively acquired supine MR images [52]. Sliding between the skin and underlying tissues has been observed, but not properly quantified. Having a method that allows for the modelling of this behaviour could therefore be used in future biomechanics-based registration algorithms for this type of application. The data was provided by Dr. L. Han. In this example, the skin is modelled with a separate membrane mesh. The solid mesh comprises 37,613 tetrahedral elements and 10,614 nodes, and was generated from a 256x512x32-voxel MR image with a 0.7x0.7x3.7mm resolution with experimental segmentation software and TetGen. The membrane mesh was generated by extraction of the surface of the solid mesh, offsetting by 3mm, and performing a manual segmentation; it has 4425 elements, 2283 nodes. The thickness of the shell elements is

| | Prostate | Breast |
|---|---|---|
| BVH & contact surface update | 0.3s | 2.87s |
| Contact search | 45.55s | 119.71s |
| Contact-force computation | 0.35s | 0.09s |
| Other FEM operations | 40.63s | 57.58s |
| Total computation time | 86.83s | 181.06s |

Table 5.3: Computation times for the breast and prostate image guidance examples broken down into the major stages of the contact-modelling pipeline

set to 5mm leaving a 0.5mm gap between the two meshes, that is quickly closed by the applied gravity forces. The chest-wall side of the solid mesh is fully fixed. The skin mesh is only held in the superior, lateral corner. The solid mesh is modelled as homogeneous transversely isotropic neo-Hookean [52] with $G = 3.57$kPa, $K = 16.67$kPa, $\eta = 37.71$kPa[6] and the preferential direction coinciding with the ventro-dorsal axis. The skin's material parameters are $E = 25$kPa, $\nu = 0.4$ for the membrane component, $E = 5$kPa, $\nu = 0.25$ for the bending stiffness. The simulation comprises 2500 time steps, representing 2.5 seconds. For reference, an otherwise identical second simulation is run without the skin mesh to better quantify the effects of the skinning. A colour map of the distance between the two results can be seen in Fig. 5.20.

The deformation in the first experiment (Fig. 5.19) is, as can be expected, quite localised with the TRUS probe penetrating into the block by about 1.1cm. In the process, the prostate is primarily rotated, but also slightly bent with respect to its apex-base axis with the base being displaced by about 4mm. Two small dents made by the surrounding tissue displaced by the probe can also be seen on the prostate's posterior surface, where the peak displacement magnitude reaches 5.1mm. That the non-rigid deformation of the prostate isn't larger can probably be attributed to the mesh consisting of two parts that can slide relative to each other.

The deformation of the solid mesh in the breast example (Fig. 5.20) is primarily a compression in ventro-dorsal direction combined with a shift of a sizeable portion of the mass in inferior and medial direction. The skin mesh is well held in place by the former despite there only being displacement boundary conditions on one corner of the mesh. The mean solid-mesh node distance between the results of the simulations with and without skin is 0.28mm with a maximum of 8.61mm. Most of the large magnitude interaction between the the two meshes appears to happen in the area surrounding the breast, although there is some evidence of a constraining of the solid mesh's expansion in the plane of the chest wall. Further, due to the proximity of the skin and the solid mesh, roughly half of both the solid mesh and skin contact surface can be assumed to be subject to contact constraints, or at least be turned up as collision candidates by the broad-phase contact search, for most of the duration of the simulation which is a larger fraction than in most simulations. Thus, it can be assumed that particularly the

[6] $\eta$ controls the stiffness in the preferred material direction, details can be found in the NiftySim user manual.

| | AABB2 | AABB4 |
|---|---|---|
| BVH refitting | 0.14s | 0.08s |
| Contact surface update | 0.04s | 0.04s |
| Broad-phase contact search | 5.74s | 6.34s |
| Narrow-phase contact search | 9.86s | 9.84s |
| Contact force computation | 0.7s | 0.7s |
| Other operations | 0.85s | 0.85s |
| Total computation time | 17.33s | 17.85s |

Table 5.4: GPU timings for the prostate example obtained with AABB2 and AABB4 bounding volumes.

contact search costs are higher in this simulation than in most simulations with a comparable mesh resolution.

In both experiments, the contact search by far dominates the contact modelling computational costs, as is shown in Table 5.3. This is to be expected due to the large number of time steps that have to be performed with explicit time integration, compared to implicit time integration. The sum of the timings obtained for all contact modelling related operations is in both cases of the same magnitude as the time required for the basic FEM modelling which, due to the low computational costs associated with the matrix-free approach of TLED, is quite challenging in its own right.

### 5.5.7 GPU Performance

This experiment reuses the prostate geometry and simulation settings from the previous section (5.5.6), but this time the simulation is run on the GPU. In this context, it is also worth investigating whether the use of AABB4 bounding volumes is faster than binary hierarchies, since the number of loop iterations of both the GPU contact search (Section 5.3.6.1) and the BVH updating algorithm (Section 5.3.6.2) is goverened by the BVH depth. Therefore, in Table 5.4 timings for two simulations are provided, an AABB2 and an AABB4 based one. The AABB2 hierarchies comprise 23,107 BVs and 9,771 BVs for the deformable and the rigid geometry, respectively. The depth of the deformable-geometry BVH is 19 levels, the rigid-geometry one is 14 levels deep. With AABB4 BVs, the BV count and depth of the deformable-geometry BVH drop to 17210 BVs and 10 levels, and the rigid-geometry BVH has 7371 BVs and 7 levels.

There is an approximate $\times 5$ total simulation speed-up achieved over a CPU execution. The timings obtained with the two different hierachy types for all parts of the pipeline, but the broad-phase contact search and the BVH refitting are practically identical, as one would expect. With both BV types, the dominant cost is the narrow-phase search followed by the broad-phase search costs. The contact-force computation, BVH refitting, and geometry update costs remain a negligible part of the total simulation costs.

In conclusion, it can be said that running the pipeline on the GPU has significant perfor-

mance benefits, albeit mainly due to the savings associated with the FEM-related operations. The dominant costs are by far the contact search costs, but running the contact search pipeline on the GPU is still a lot cheaper than running it on the CPU, with a circa ×3 speed-up. It can also be assumed that the contact search costs as a proportion of the total simulation costs drop with finer meshed geometry, not only because of surface/volume ratio considerations, but also because of the fixed costs associated with CUDA kernels and many stages of the pipeline not having sufficient items to process to fill even one CUDA data block. As with the CPU implementation, using quarternary BVHs does not accelerate the simulation.

## 5.6 Conclusions

Methods suitable for detecting and handling of contacts arising in explicit FEM simulation of a range of scenarios, have been presented in this chapter: deformable geometry self-collisions, contacts between deformable solid and membrane meshes, and deformable geometry and a range of rigid geometry. The contact search portion of the presented pipeline is optimised for the typically small time-steps one has with explicit time integration in that it keeps the number of BV refittings low by identifying the parts of the BVH where containment of the geometry is ensured and self-collisions can be excluded. The success of this strategy can be seen in the consistently low numbers of BV refittings.

Further, in this chapter, an improved formula for the computation of the surface-normal bounding cones used for self-collision detection, via Provot's recursive algorithm, has been presented. I have demonstrated that the proposed formula leads to a marked reduction in the number of BVH subtrees that must be checked for self-collisions, compared to the formula originally proposed by Provot.

On the contact modelling side, a robust general-purpose method that can deal with both geometric, as well as temporal singularities via smoothing, has been presented. Mathematically, the contact-force smoothing with respect to space is done by means of linearly interpolated surface normals, and with respect to time, by means of linearly increasing gap-rate proportional forces. Despite these stability improving modifications, the results obtained with the proposed contact model remain consistent with physics and accurate as has been shown with transient impact and quasi-static, resting-contact friction experiments.

It has also been shown that the entire proposed contact-modelling pipeline can be executed on the CPU within a time frame that is of the same order of magnitude as the time required for standard TLED computations, in real-world image-guidance applications. While the CPU pipeline is sufficiently fast for most image guidance tasks, algorithms requiring the running of a large number of simulations, such as pipelines performing material parameter optimisations, are likely best implemented based on the GPU implementation of the pipeline, which offers a significant speed-up running large simulations. The observed timings of the GPU contact modelling code, as a proportion of the total simulation execution time, are much higher than those on the CPU. However, GPU implementations have significant fixed costs associated with

them, e.g. due to fixed kernel sizes, and therefore it is reasonable to assume that the speed up is going to increase and the relative cost of contact modelling decrease as the simulation complexity increases.

## Chapter 6

# Conclusions and Future Work

In this thesis, I have presented my work on the NiftySim FEM software package, with a particular emphasis on my work on contact modelling, and on an algorithm and framework for the estimation of liver deformation due to pneumoperitoneum that uses some of that general purpose soft-tissue biomechanics modelling software. I have also explained how these contributions fit into a computer pipeline for image-guidance of laparoscopic liver surgery.

## 6.1 Chapter Summaries

*In chapter 1*, a discussion of the major medical motivation for performing laparoscopic liver resection surgery was provided, as was an overview over the clinical workflow of such procedures and an explanation of the inherent technical challenges. The latter are mainly the deformation caused to the liver and its anatomical surroundings by the necessary gas insufflation of the abdomen compared to the CT images acquired pre-operatively for planning; the limited field of view available through laparoscopic video and ultrasound, with the video image quality frequently being further degraded by blood staining of the lens and smoke formation; and the inability of the surgeon to palpate the tissue for location of lesions. I also provided an outline of the envisaged image-guidance system that motivated my research.

*In chapter 2*, I have reviewed the literature on all major components that go into an image-guidance system for laparoscopic surgery, with a particular emphasis on biomechanically-informed pipelines. In conclusion, it can be said that simulation of the tissue biomechanics is frequently and successfully used where only a part of the intra-operative target organ is visible and an extrapolation of the displacements determined for that portion of the surface to the rest of the organs volume must be performed. However, doing so necessitates a) recovery of the organ surface from intra-operatively acquired images; b) a means to establish correspondences between the intra-operative and the pre-operative organ surface. One challenge related to the correspondence finding that was discussed at some length in the chapter was that of coming up with a high-quality intialiser for the correspondence finding algorithm. This initialisation is needed because the algorithms used for correspondence finding are frequently derivatives of the ICP algorithm and/or share its tendency to converge to local minima of the distance

cost function. That discussion also provided the main motivation for the pipeline discussed in chapter 3.

*In chapter 3*, an algorithm for the estimation of the deformation caused to the liver due to pneumoperitoneum was presented. The algorithm computes this estimate as a weighted average of known displacements stored in a database, along with the data required to establish point correspondences between the training liver geometries and the patient liver. Point-wise computed similarities between unseen and training-subject livers provide the weights used in displacement computation. A biomechanical simulation is run at the end of the pipeline which at this point in time is mainly a safeguard against any non-physical deformation arising from the computation of the estimate as a point-wise weighted average of training displacements.

Since the estimates are based on weighted averages of known deformations, the method struggles with outliers, but in experiments with porcine image data, the algorithm did provide good estimates where the unseen subject was representative of the training data with RMS TREs in the 5.82–11.47mm range. The pipeline also comprises an algorithm for point-correspondence finding through augmented spectral embeddings that has a much wider applicability in initialisation of intensity-based image registration. That pipeline was independently validated, and satisfactory RMS TREs of 2.96–12.04mm were obtained; omitting one subject with severely motion degraded image data, the range of RMS errors reduces to 2.96–3.88mm. Further, the whole pipeline is not liver or image-guidance specific and, with different similarity metrics, it could be used for other applications where anatomical structures are subjected to loads that are comparable from one patient to another, or e.g. for simulating pneumoperitoneum in surgical simulation.

*In chapter 4*, the NiftySim software package for simulation of soft-tissue biomechanics was described. The FEM solver is based on the TLED framework which has been shown to be a very fast option for the accurate simulation of the mechanical behaviour of soft tissues, that are goverened by complex constitutive models. NiftySim is primarily aimed at running quasi-static simulations for medical image registration purposes. As such, it provides the facilities required for efficiently simulating most scenarios encountered in imaging of soft tissues, and it has an easy to use XML-based input format for simulation definitions, that along with its stand-alone solver application, allows for convenient manual prototyping of simulations. NiftySim provides numerous complex non-linear hyper-elastic and hyper-viscoelastic constitutive models, membranes and shells for simulation of skin and other thin anatomical structures, and is capable of simulating a wide range of contact scenarios.

The software package, originally developed by Dr. Zeike Taylor, has been maintained by me for the last 4 years. During that time, I have made numerous useability improvements to the package, such as adding intuitive methods for defining boundaries through geometric criteria, and also added major new features that build on and go beyond the TLED framework, such as a membrane/shell implementation and a general-purpose contact modelling pipeline.

While no validation per se is contained in the chapter, the software has been successfully used in multiple works of peer-reviewed research - many of which dealt with image guidance, and other researchers have validated the TLED algorithm [99] and NiftySim's implementation thereof [53] against commercial packages.

*In chapter 5*, I have presented a contact modelling pipeline suitable for implementation in matrix-free explicit FEM solvers, particularly such based on the TLED algorithmic framework. The presented contact modelling pipeline comprises a contact search portion based on bounding volume hierarchies and a contact force computation portion based on the explicit Lagrange multiplier method. The most interesting novel aspects of the contact search component are the new formulas for computing surface-normal bounding cones used for self-collision detection and the drastic savings in BVH refitting costs made possible by incorporating the self-collision search criterion in the cost function determining the BVH topology, and a new BVH update algorithm. The BVH update algorithm analyses the deformation of the geometry between refittings of the hierarchy and only updates such parts of the BVH that are no longer properly containing the geometry. The algorithm for the computation of the contact forces is designed for parallel processing of contacts, and includes spatial and temporal smoothing heuristics for improved stability. The spatial smoothing is achieved by linearly interpolating the normals determining the direction of the contact forces. The temporal smoothing is done with forces computed from the gap rate, i.e. the rate of approach of two surfaces, and based on a novel formulation. The incorporation of linearly interpolated normals and edge-edge collisions, while not generally novel, is an extension of the work that provided the starting point for the contact force formulation, as is the method of distribution of contact forces over the master facets and edges. Another standout quality of the algorithm is its versatility; it can simulate collisions of multiple deformable bodies, deformable-geometry self-collisions, deformable geometry colliding with static or moving rigid geometry, and two-sided contacts of 2D surfaces (membranes).

In a detailed validation of all novel components of the pipeline, it was shown that the proposed BVH refitting algorithm can reduce the costs as a percentage of overall simulation costs of BVH refitting to the single digits of per cents or less, thus making it practically negligible. The presented new formulas for computation of the surface-normal bounding cones that are used for self-collision detection lead to a significant reduction in the number of BVH subtrees that must be checked for self-collisions and overall computation time compared to the standard method proposed by Provot [114]. The contact force formulations were shown to have excellent energy and momentum conservation characteristics and allow for an easy and acceptably accurate way of simulating friction. On the CPU, the entire pipeline can be executed within a time frame comparable to the time required for FE modelling, as was demonstrated on two image-guidance examples. In a GPU implementation, the contact modelling costs become dominant, but there is nevertheless a significant speed-up compared to a CPU execution.

## 6.2 Future Work

Possible extensions of the work conducted so far that can be implemented in the relative short term include:

1) Improvement of the pneumoperitoneum simulation by including a state-of-the-art biomechanical model for the liver. The most important of these possible improvements have been listed in the literature review – a distinctive stiffness for the Glisson capsule, simulation of stiffness anisotropy due to the vasculature, simulation of the ligaments through separate geometries, and a use of a more refined viscera model, based on CT image segmentations. With the exception of the vasculature model, all of these improvements can be implemented with the current version of NiftySim without any need for custom FEM code.

Provided a larger database is available, an avenue for the pneumoperitoneum simulation worth investigating is improving the point-wise weighted average of database displacements used for computing the deformation estimate (Eq. (3.13)) with linear regression techniques and introduction of exponential coefficients $\alpha_i$ on the similarity metrics. A starting point for determining the $\alpha_i$ may be given by the formula

$$\log(s(\boldsymbol{u}(\boldsymbol{x_A}), \boldsymbol{u}(\boldsymbol{\phi_{A2B}}(\boldsymbol{x_A})))) = \sum_i \alpha_i \log\left({}^{AB}\tilde{s}_i\right) \tag{6.1}$$

where $s(\boldsymbol{u}(\boldsymbol{x_A}), \boldsymbol{u}(\boldsymbol{\phi_{A2B}}(\boldsymbol{x_A})))$ is a measure of the similarity of the displacement at the point $\boldsymbol{x_A}$ on the surface $A$ and the corresponding displacement on surface $B$, $\boldsymbol{u}(\boldsymbol{\phi_{A2B}}(\boldsymbol{x_A}))$, and the $\tilde{s}_i$ are liver segmentation similarity metrics such as those proposed in Sect. 3.2.7. Finding suitable parameters $\alpha_i$ may then be accomplished by minimising the error function arising from (6.1) subject to the constraint $\alpha_i \geq 0, \forall i$. This approach could be further extended by computing different parameters $\alpha_i$ for the different anatomical regions of the liver, if it turns out that some similarity metrics work significantly better or worse on particular parts of the liver.

2) My non-contact modelling contributions to NiftySim, in the shorter term, could be extended by adding more membrane and shell constitutive models. In the slightly longer term, one could also consider adding beam elements to achieve the aforementioned simulation of anisotropy caused by vasculature, as done by Haouchine et al. [54].

3) The CPU implementation of the contact modelling pipeline can be considered mostly complete, and, thanks to an extensive use of C++ templates, it is also a very extensible and fast code. However, many features of the CPU code have not yet been implemented in the CUDA version; these include: Updating of the self-collision candidate list during the simulation, friction modelling, support for quadrilateral surface facets, and membrane support. Further, there is still a lot of untapped optimisation potential in the GPU code. E.g., by replacing or improving the merge sort implementation that is used for sorting intermediate results using multiple streams.

In this thesis I have only considered an image guidance system that helps the surgeon come up with a mental plan on how to proceed after the liver has been mobilised. In the longer term, there are certainly many interesting problems waiting to be tackled if one wants

to provide image guidance beyond that point, after the transection of the parenchyma has begun, or extend it for patient-specific surgical simulation for intervention-planning. Such problems include the remeshing required to account for the resected parts of the organ, and the modelling of the interactions between the liver tissue and the surgical instruments. Many of the algorithms for intra-operative surface recovery and feature tracking and matching presented in the literature review could likely help with the former problem. The latter problem has partially been addressed in the work of my collaborators and me, via instrument tracking and the modelling of contacts between deformable and moving rigid bodies.

For surgical simulation, the simulation of cutting is a major area that would need investigating. A TLED-specific approach to this problem can be found in [150]. The algorithm retains many of TLED's performance advantages and the original admissible maximum time step size by duplicating and copying the precomputed data of initial-configuration elements that are cut to newly created elements and solely accounting for the cut by placing a virtual node at the point of incission. A slightly older cutting algorithm suitable for explicit FEM solvers can be found in [122]. The latter algorithm simulates cuts by moving the nearest nodes of the mesh onto the cut line, thus preserving the total element count. It subsequently regularises the resulting mesh in the vicinity of the cut with a combination of a mass-spring and particle simulation to avoid badly shaped elements. Finally, the nodes on the cut line are duplicated and the mesh opened.

If online image-guidance or virtual-reality surgical simulation is the aim, one might also want to consider simulating the patient's heartbeat and breathing, e.g., by means of Pratt's algorithm [112], as mentioned in the introduction.

# Bibliography


[1] A Agudo, B Calvo, and J M M Montiel. Finite Element based sequential Bayesian Non-Rigid Structure from Motion. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1418–1425. IEEE, June 2012.

[2] B Ahn and J Kim. Measurement and characterization of soft tissue behavior with surface deformation and force response under large deformations. *Medical image analysis*, 14(2):138–48, April 2010.

[3] C Albitar, P Graebling, and C Doignon. Robust Structured Light Coding for 3D Reconstruction. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–6. IEEE, 2007.

[4] M Allan, P-L Chang, S Ourselin, D J Hawkes, A Sridhar, J Kelly, and D Stoyanov. Image based surgical instrument pose estimation with multi-class labelling and optical flow. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 331–338. Springer, 2015.

[5] J Allard, F Faure, H Courtecuisse, F Falipou, C Duriez, and P G Kry. Volume contact constraints at arbitrary resolution. *ACM Transactions on Graphics*, 29(4):82:1–10, July 2010.

[6] A Ananthakrishnan, V Gogineni, and K Saeian. Epidemiology of primary and secondary liver cancers. *Seminars in interventional radiology*, 23(1):47–63, March 2006.

[7] K S Arun, TS Huang, and S D Blostein. Least-squares fitting of two 3-D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698–700, 1987.

[8] M A Audette, F P Ferrie, and T M Peters. An algorithmic overview of surface registration techniques for medical imaging. *Medical image analysis*, 4(3):201–17, September 2000.

[9] J Bano, A Hostettler, S A Nicolau, S Cotin, C Doignon, H S Wu, M H Huang, L Soler, and J Marescaux. Simulation of pneumoperitoneum for laparoscopic surgery planning. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2012*, 15(Pt 1):91–8, January 2012.

[10] J Bano, A Hostettler, S A Nicolau, C Doignon, and H S Wu. Simulation of the Abdominal Wall and Its Arteries after Pneumoperitoneum for Guidance of Port Positioning in Laparoscopic Surgery. In *8th International Symposium, ISVC 2012*, pages 1–11, 2012.

[11] J Bano, S A Nicolau, A Hostettler, and C Doignon. Registration of Preoperative Liver Model for Laparoscopic Surgery from Intraoperative 3D Acquisition. In H. Liao, C. A. Linte, K. Masamune, T.M. Peters, and G. Zheng, editors, *Augmented Reality Environments for Medical Imaging and Computer-Assisted Interventions*, volume 8090 of *Lecture Notes in Computer Science*, pages 201–210, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[12] K J Bathe. *Finite element procedures*. Prentice Hall, 1996.

[13] M Baumhauer, M Feuerstein, H-P Meinzer, and J Rassweiler. Navigation in endoscopic soft tissue surgery: perspectives and limitations. *Journal of endourology / Endourological Society*, 22(4):751–66, April 2008.

[14] M Belkin and P Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in neural information processing systems*, 14:585–591, 2001.

[15] T Belytschko and M O Neal. Contact-impact by the pinball algorithm with penalty and Lagrangian methods. *International Journal for Numerical Methods in Engineering*, 31(3):547–572, March 1991.

[16] A B Benincasa, L W Clements, S D Herrell, and R L Galloway. Feasibility study for image-guided kidney surgery: Assessment of required intraoperative surface for accurate physical to image space registrations. *Medical Physics*, 35(9):4251, 2008.

[17] P J Besl and N D McKay. Method for registration of 3-D shapes. In Paul S. Schenker, editor, *Proc. SPIE 1611, Sensor Fusion IV: Control Paradigms and Data Structures*, volume 16, pages 586–606, April 1992.

[18] J Bonet, RD Wood, J Mahaney, and P. Heywood. Finite element analysis of air supported membrane structures. *Computer Methods in Applied Mechanics and Engineering*, 190(5-7):579–595, November 2000.

[19] F L Bookstein. Principal warps: thin-plate splines and the decomposition of deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(6):567–585, June 1989.

[20] J Bosman, N Haouchine, J Dequidt, and I Peterlik. The Role of Ligaments: Patient-Specific or Scenario-Specific? In Fernando Bello and Stéphane Cotin, editors, *6th International Symposium, ISBMS 2014*, volume 8789 of *Lecture Notes in Computer Science*, pages 228–232, Cham, 2014. Springer International Publishing.

[21] R Bryant, A Laurent, C Tayar, and D Cherqui. Laparoscopic liver resection-understanding its role in current practice: the Henri Mondor Hospital experience. *Annals of surgery*, 250(1):103–11, July 2009.

[22] T Carter, C Tanner, N Beechey-Newman, D Barratt, and D Hawkes. MR navigated breast surgery: method and initial clinical experience. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2008*, 11(Pt 2):356–63, January 2008.

[23] T J Carter, M Sermesant, D M Cash, D C Barratt, C Tanner, and D J Hawkes. Application of soft tissue modelling to image-guided surgery. *Medical engineering & physics*, 27(10):893–909, December 2005.

[24] D M Cash, M I Miga, S C Glasgow, B M Dawant, L W Clements, Z Cao, R L Galloway, and W C Chapman. Concepts and preliminary data toward the realization of image-guided liver surgery. *Journal of gastrointestinal surgery : official journal of the Society for Surgery of the Alimentary Tract*, 11(7):844–59, July 2007.

[25] D M Cash, T K Sinha, W C Chapman, H Terawaki, B M Dawant, R L Galloway, and M I Miga. Incorporation of a laser range scanner into image-guided liver surgery: Surface acquisition, registration, and tracking. *Medical Physics*, 30(7):1671, 2003.

[26] S Chang, A Laurent, C Tayar, M Karoui, and D Cherqui. Laparoscopy as a routine approach for left lateral sectionectomy. *The British journal of surgery*, 94(1):58–63, January 2007.

[27] H Chui and A Rangarajan. A feature registration framework using mixture models. In *Proceedings IEEE Workshop on Mathematical Methods in Biomedical Image Analysis. MMBIA-2000 (Cat. No.PR00737)*, pages 190–197. IEEE Comput. Soc, 2000.

[28] F Cirak and M West. Decomposition contact response (DCR) for explicit finite element dynamics. *International Journal for Numerical Methods in Engineering*, 64(8):1078–1110, October 2005.

[29] M J Clarkson, D Rueckert, D L G Hill, and D J Hawkes. Using photo-consistency to register 2D optical images of the human face to a 3D surface model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1266–1280, 2001.

[30] M J Clarkson, G Zombori, S Thompson, J Totz, Y Song, M Espak, S Johnsen, D Hawkes, and S Ourselin. The niftk software platform for image-guided interventions: platform overview and niftylink messaging. *International Journal of Computer Assisted Radiology and Surgery*, Published online:1–16, 2014.

[31] L W Clements, W C Chapman, B M Dawant, R L Galloway, and M I Miga. Robust surface registration using salient anatomical features for image-guided liver surgery: Algorithm and validation. *Medical Physics*, 35(6):2528, 2008.

[32] L W Clements, P Dumpuri, W C Chapman, R L Galloway, and M I Miga. Atlas-based method for model updating in image-guided liver surgery. In *Proceedings of SPIE*, volume 6509, pages 650917–650917–12. SPIE, 2007.

[33] A Cooper and T A Aloia. Surgical resection for hepatocellular carcinoma. *Translational Cancer Research*, 2(6):450–459, 2013.

[34] S Cotin, H Delingette, and N Ayache. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions On Visualization and Computer Graphics*, 5(1):62–73, 1999.

[35] K P Croome and M H Yamashita. Laparoscopic vs open hepatic resection for benign and malignant tumors: An updated meta-analysis. *Archives of surgery (Chicago, Ill. : 1960)*, 145(11):1109–18, November 2010.

[36] A J Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1403–1410 vol.2. IEEE, 2003.

[37] J Dequidt, J Lenoir, and S Cotin. Interactive contacts resolution using smooth surface representation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2007*, pages 850–857, 2007.

[38] P Dumpuri, L W Clements, B M Dawant, and M I Miga. Model-updated image-guided liver surgery: Preliminary results using surface characterization. *Progress in biophysics and molecular biology*, 103(2-3):197–207, September 2010.

[39] P Dumpuri, R C Thompson, B M Dawant, A Cao, and M I Miga. An atlas-based method to compensate for brain shift: preliminary results. *Medical image analysis*, 11(2):128–145, April 2007.

[40] C Duriez, F Dubois, A Kheddar, and C Andriot. Realistic haptic rendering of interacting deformable objects in virtual environments. *IEEE transactions on visualization and computer graphics*, 12(1):36–47, 2006.

[41] M Ferrant, B Macq, A Nabavi, and S K Warfield. Deformable Modeling for Characterizing Biomedical Shape Changes. In *Discrete Geometry for Computer Imagery*, pages 235–248, 2000.

[42] M Ferrant, A Nabavi, B Macq, F A Jolesz, R Kikinis, and S K Warfield. Registration of 3-D intraoperative MR images of the brain using a finite-element biomechanical model. *IEEE Transactions on Medical Imaging*, 20(12):1384–97, December 2001.

[43] M Feuerstein, T Mussack, S M Heining, and N Navab. Intraoperative laparoscope augmentation for port placement and resection planning in minimally invasive liver resection. *IEEE Transactions on Medical Imaging*, 27(3):355–69, March 2008.

[44] F G Flores and E Oñate. Improvements in the membrane behaviour of the three node rotation-free BST shell triangle using an assumed strain approach. *Computer Methods in Applied Mechanics and Engineering*, 194(6-8):907–932, February 2005.

[45] O J Garden, M Rees, G J Poston, D Mirza, M Saunders, J Ledermann, J N Primrose, and R W Parks. Guidelines for resection of colorectal cancer liver metastases. *Gut*, 55 Suppl 3:iii1–8, August 2006.

[46] S Gottschalk, M C Lin, and D Manocha. OBBTree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180. ACM, 1996.

[47] O G Grasa, J Civera, and J M M Montiel. EKF monocular SLAM with relocalization for laparoscopic sequences. In *2011 IEEE International Conference on Robotics and Automation*, pages 4816–4821. IEEE, May 2011.

[48] H Gray and W H Lewis. *Anatomy of the Human Body*. Lea & Febiger, 1918.

[49] K Gupta and A P Pobil. *Practical motion planning in robotics: current approaches and future directions*. Wiley, 1998.

[50] J.O. Hallquist. *LS-DYNA Theory Manual*. Livermore, CA: Livermore Software Technology Corporation, 2006.

[51] L Han, J Hipwell, T Mertzanidou, T Carter, M Modat, S Ourselin, and D Hawkes. A hybrid FEM-based method for aligning prone and supine images for image guided breast surgery. In *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 1239–1242. IEEE, March 2011.

[52] L Han, J Hipwell, Z A Taylor, C Tanner, S Ourselin, and D J Hawkes. Fast deformation simulation of breasts using GPU-based dynamic explicit finite element method. *Digital Mammography*, pages 728–735, 2010.

[53] L Han, J H Hipwell, C Tanner, Z Taylor, T Mertzanidou, J Cardoso, S Ourselin, and D J Hawkes. Development of patient-specific biomechanical models for predicting large breast deformation. *Physics in medicine and biology*, 57(2):455–72, January 2012.

[54] N Haouchine, J Dequidt, I Peterlik, E Kerrien, M O Berger, and S Cotin. Image-guided simulation of heterogeneous tissue deformation for augmented reality during hepatic surgery. In *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 199–208. IEEE, October 2013.

[55] B Heidelberger, M Teschner, and M Gross. Detection of collisions and self-collisions using image-space techniques. *Journal of WSCG*, 12(3):145–152, 2004.

[56] M W Heinstein and T A Laursen. An algorithm for the matrix-free solution of quasistatic frictional contact problems. *International Journal for Numerical Methods in Engineering*, 44(9):1205–1226, March 1999.

[57] M W Heinstein, F J Mello, S W Attaway, and T A Laursen. Contact–impact modeling in explicit transient dynamics. *Computer Methods in Applied Mechanics and Engineering*, 187(3-4):621–640, July 2000.

[58] A J Herline, J D Stefansic, J P Debelak, S L Hartmann, CW Pinson, RL Galloway, and WC Chapman. Image-guided surgery - Preliminary feasibility studies of frameless stereotactic liver surgery. *ARCHIVES OF SURGERY*, 134(6):644–649, JUN 1999. 106th Scientific Session of the Western-Sugical-Association, INDIANAPOLIS, INDIANA, NOV 16-18, 1998.

[59] S D Herrell, R L Galloway, and L-M Su. Image-guided robotic surgery: update on research and potential applications in urologic surgery. *Current opinion in urology*, 22(1):47–54, January 2012.

[60] G A Holzapfel. *Nonlinear Solid Mechanics: A Continuum Approach for Engineering*. John Wiley & Sons, Chichester, 2000.

[61] M Hu, G Penney, M Figl, P Edwards, F Bello, R Casula, D Rueckert, and D Hawkes. Reconstruction of a 3D surface from video that is robust to missing data and outliers: Application to minimally invasive surgery using stereo and mono endoscopes. *Medical image analysis*, December 2010.

[62] Y Hu, H U Ahmed, Z Taylor, C Allen, M Emberton, D Hawkes, and D Barratt. MR to ultrasound registration for image-guided prostate interventions. *Medical image analysis*, 16(3):687–703, April 2012.

[63] Y Hu, T J Carter, H U Ahmed, M Emberton, C Allen, D J Hawkes, and D C Barratt. Modelling prostate motion for data fusion during image-guided interventions. *IEEE Transactions on Medical Imaging*, 30(11):1887–900, November 2011.

[64] Y Hu, E J Rijkhorst, R Manber, D Hawkes, and D Barratt. Deformable vessel-based registration using landmark-guided coherent point drift. *Medical Imaging and Augmented Reality*, pages 60–69, 2010.

[65] T J R Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analyses*. Prentice-Hall, Inc, Englewood Cliffs, NJ, 1987.

[66] V Jain, H Zhang, and O Van Kaick. Non-rigid spectral correspondence of triangle meshes. *International Journal of Shape Modeling*, 13(01):101–124, June 2007.

[67] S Johnsen, Z Taylor, S Thompson, M Hu, B Davidson, D Hawkes, and S Ourselin. Explicit FEM Tissue Simulation for Surgical Image-Guidance. In *Proceedings of Living Imaging – LIVIM 2011*, volume Published online, pages 1–10, 2011.

[68] S F Johnsen, Z A Taylor, M Clarkson, S Thompson, M Hu, K Gurusamy, B Davidson, D J Hawkes, and S Ourselin. Explicit contact modeling for surgical computer guidance and simulation. In David R. Holmes III and Kenneth H. Wong, editors, *Proc. SPIE 8316, Medical Imaging 2012: Image-Guided Procedures, Robotic Interventions, and Modeling*, pages 831623–831623–9, February 2012.

[69] S F Johnsen, Z A Taylor, L Han, Y Hu, Clarkson M J, D J Hawkes, and S Ourselin. Detection and Modelling of Contacts in Explicit Finite-Element Simulation of Soft-Tissue Biomechanics. *International Journal of Computer Assisted Radiology and Surgery*, 10:1873–1891, 2015.

[70] S F Johnsen, Z A Taylor, Clarkson M J, J Hipwell, M Modat, B Eiben, L Han, Y Hu, T Mertzanidou, D J Hawkes, and S Ourselin. NiftySim: A GPU-based nonlinear finite element package for simulation of soft-tissue biomechanics. *International Journal of Computer Assisted Radiology and Surgery*, 10:1077–1095, 2015.

[71] S F Johnsen, S Thompson, M J Clarkson, M Modat, Y Song, J Totz, K Gurusamy, B Davidson, Z A Taylor, D J Hawkes, and S Ourselin. Database-Based Estimation of Liver Deformation Under Pneumoperitoneum for Surgical Image-Guidance and Simulation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, 9350:450–458, 2015.

[72] G R Joldes, A Wittek, and K Miller. An efficient hourglass control implementation for the uniform strain hexahedron using the total lagrangian formulation. *Communications in Numerical Methods in Engineering*, 24:1315–1323, 2008.

[73] G R Joldes, A Wittek, and K Miller. Non-locking tetrahedral finite element for surgical simulation. *Communications in Numerical Methods in Engineering*, 25(7):827–836, July 2009.

[74] F A Jolesz, A Nabavi, and R Kikinis. Integration of interventional MRI with computer-assisted surgery. *Journal of magnetic resonance imaging : JMRI*, 13(1):69–77, January 2001.

[75] T L Kay and J T Kajiya. Ray tracing complex scenes. *ACM SIGGRAPH Computer Graphics*, 20:269–278, 1986.

[76] R Kimmel and J A Sethian. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences of the United States of America*, 95(15):8431–5, July 1998.

[77] T P Kingham, S Jayaraman, L W Clements, M A Scherer, J D Stefansic, and W R Jarnagin. Evolution of Image-Guided Liver Surgery: Transition from Open to Laparoscopic Procedures. *Journal of Gastrointestinal Surgery*, 17:1274–1282, 2013.

[78] T Kitasaka, K Mori, and Y Hayashi. Virtual pneumoperitoneum for generating virtual laparoscopic views based on volumetric deformation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2004*, 3:559–567, 2004.

[79] E C H Lai, C N Tang, and M K W Li. Robot-assisted laparoscopic hemi-hepatectomy: technique and surgical outcomes. *International journal of surgery (London, England)*, 10(1):11–5, January 2012.

[80] T Lange, N Papenberg, S Heldmann, J Modersitzki, B Fischer, H Lamecker, and P M Schlag. 3D ultrasound-CT registration of the liver using combined landmark-intensity information. *International journal of computer assisted radiology and surgery*, 4(1):79–88, January 2009.

[81] T Langø, S Vijayan, A Rethy, C Vå penstad, O V Solberg, R Må rvik, G Johnsen, and T N Hernes. Navigated laparoscopic ultrasound in abdominal soft tissue surgery: technological overview and perspectives. *International journal of computer assisted radiology and surgery*, 7(4):585–99, July 2012.

[82] T Larsson and T Akenine-Möller. Collision detection for continuously deforming bodies. *Eurographics 2001*, pages 325–333, 2001.

[83] T Larsson and T Akenine-Möller. A dynamic bounding volume hierarchy for generalized collision detection. *Computers & Graphics*, 30(3):450–459, June 2006.

[84] C Lauterbach, Q Mo, and D Manocha. gProximity: Hierarchical GPU-based Operations for Collision and Distance Queries. *Computer Graphics Forum*, 29(2):419–428, June 2010.

[85] B Lee. *Physically Based Modelling for Topology Modification and Deformation in Surgical Simulation*. PhD thesis, University of Sydney, 2007.

[86] M C Lin and S Gottschalk. Collision detection between geometric models: A survey. In *In Proc. of IMA Conference on Mathematics of Surfaces*, pages 37–56, 1998.

[87] Y Liu. Automatic registration of overlapping 3D point clouds using closest points. *Image and Vision Computing*, 24(7):762–781, July 2006.

[88] J M Llovet. Updated treatment approach to hepatocellular carcinoma. *Journal of gastroenterology*, 40(3):225–35, March 2005.

[89] H Lombaert, J Sporring, and K Siddiqi. Diffeomorphic Spectral Matching of Cortical Surfaces. *Information Processing in Medical Imaging*, pages 376–389, 2013.

[90] B D Lucas and T Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, pages 674–679. Morgan Kaufmann Publishers Inc., 1981.

[91] R M Lupinacci, E S Mello, F F Coelho, J A P Kruger, M V Perini, R S Pinheiro, G M Fonseca, I Cecconello, and P Herman. Prognostic implication of mucinous histology in resected colorectal cancer liver metastases. *Surgery*, 155(6):1062–8, June 2014.

[92] L Maier-Hein, P Mountney, A Bartoli, H Elhawary, D Elson, A Groch, A Kolb, M Rodrigues, J Sorger, S Speidel, and D Stoyanov. Optical techniques for 3D surface reconstruction in computer-assisted laparoscopic surgery. *Medical image analysis*, 17(8):974–96, December 2013.

[93] C R Maurer, G B Aboutanos, B M Dawant, R J Maciunas, and J M Fitzpatrick. Registration of 3-D images using weighted geometrical features. *IEEE Transactions on Medical Imaging*, 15(6):836–849, DEC 1996.

[94] U Meier, O Lopez, C Monserrat, M C Juan, and M Alaniz. Real-time deformable models for surgery simulation: a survey. *Computer Methods and Programs in Biomedicine*, 77:183–197, 2005.

[95] T Mertzanidou, J Hipwell, S Johnsen, L Han, B Eiben, Z Taylor, S Ourselin, H Huisman, R Mann, U Bick, N Karssemeijer, and D Hawkes. MRI to X-ray mammography intensity-based registration with simultaneous optimisation of pose and biomechanical transformation parameters. *Medical Image Analysis*, Published online, 2014.

[96] J Mezger, S Kimmerle, and O Etzmuß. Hierarchical techniques in collision detection for cloth animation. *Journal of WSCG*, 11(2):322–329, 2003.

[97] A M Mharib, A R Ramli, S Mashohor, and R B Mahmood. Survey on liver CT image segmentation methods. *Artificial Intelligence Review*, 37(2):83–95, April 2011.

[98] K Miller, G Joldes, D Lance, and A Wittek. Total Lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation. *Communications in Numerical Methods in Engineering*, 23(2):121–134, 2007.

[99] K Miller, A Wittek, G Joldes, A Horton, T Dutta-Roy, J Berger, and L Morriss. Modelling brain deformations for computer-integrated neurosurgery. *International Journal for Numerical Methods in Biomedical Engineering*, 26(1):117–138, January 2010.

[100] M Modat, P Daga, M J Cardoso, S Ourselin, G R Ridgway, and J Ashburner. Parametric non-rigid registration using a stationary velocity field. In *Proceedings of the Workshop on Mathematical Methods in Biomedical Image Analysis*, pages 145–150, 2012.

[101] M Modat, G R Ridgway, Z A Taylor, M Lehmann, J Barnes, D J Hawkes, N C Fox, and S Ourselin. Fast free-form deformation using graphics processing units. *Computer methods and programs in biomedicine*, 98(3):278–84, June 2010.

[102] T Möller and B Trumbore. Fast, minimum storage ray/triangle intersection. In *ACM SIGGRAPH 2005 Courses*, page 7. ACM, 2005.

[103] C Monserrat, U Meier, M Alcaniz, F Chinesta, and M C Juan. A new approach for the real-time simulation of tissue deformations in surgery simulation. *Computer Methods and Programs in Biomedicine*, 64(2):77–85, 2001.

[104] P Mountney and GZ Yang. Motion compensated slam for image guided surgery. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2010*, 13:496–504, 2010.

[105] A Myronenko and X Song. Point set registration: coherent point drift. *IEEE transactions on pattern analysis and machine intelligence*, 32(12):2262–75, December 2010.

[106] S Nicolau, L Soler, D Mutter, and J Marescaux. Augmented reality in laparoscopic surgical oncology. *Surgical oncology*, 20(3):189–201, September 2011.

[107] NVIDIA Corporation. *NVIDIA CUDA Programming Guide Version 6.5*, 2014.

[108] E Oñate, P Cendoya, and J Miquel. Non-linear explicit dynamic analysis of shells using the BST rotation-free triangle. *Engineering Computations*, 19(6):662–706, 2002.

[109] O Oktay, L Zhang, T Mansi, P Mountney, P Mewes, S Nicolau, L Soler, and C Chef d'Hotel. Biomechanically Driven Registration of Pre-to Intra-Operative 3D Images for Laparoscopic Surgery. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*, pages 1–9, 2013.

[110] M A Otaduy, R Tamstorf, D Steinemann, and M Gross. Implicit Contact Handling for Deformable Objects. *Computer Graphics Forum*, 28(2):559–568, April 2009.

[111] R Plantefeve, I Peterlik, H Courtecuisse, R Trivisonne, J-P Radoux, and S Cotin. Atlas-based transfer of boundary conditions for biomechanical simulation. In Polina Golland, Nobuhiko Hata, Christian Barillot, Joachim Hornegger, and Robert Howe, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2014*, volume 8674 of *Lecture Notes in Computer Science*, pages 33–40, Cham, 2014. Springer International Publishing.

[112] P Pratt. Image guidance and surgery simulation using inverse nonlinear finite element methods. In *Biomedical Simulation - 4th International Symposium, ISBMS 2008*, pages 185–190, 2008.

[113] P Pratt, D Stoyanov, M Visentini-Scarzanella, and G-Z Yang. Dynamic guidance for robotic surgery using image-constrained biomechanical models. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2010*, 13(Pt 1):77–85, January 2010.

[114] X Provot. Collision and self-collision handling in cloth model dedicated to design garments. In *Graphics interface*, volume 97, pages 177–189. Citeseer, 1997.

[115] M Puso. A mortar segment-to-segment frictional contact method for large deformations. *Computer Methods in Applied Mechanics and Engineering*, 193(45-47):4891–4913, November 2004.

[116] J-L Raoul. Natural history of hepatocellular carcinoma and current treatment options. *Seminars in nuclear medicine*, 38(2):S13–8, March 2008.

[117] T P Rauth, P Q Bao, R L Galloway, J Bieszczad, E M Friets, D A Knaus, D B Kynor, and A J Herline. Laparoscopic surface scanning and subsurface targeting: implications for image-guided laparoscopic liver surgery. *Surgery*, 142(2):207–14, August 2007.

[118] E Roan. The effect of Glisson's capsule on the superficial elasticity measurements of the liver. *Journal of biomechanical engineering*, 132(10):104504, October 2010.

[119] A Samani, J Bishop, M J Yaffe, and D B Plewes. Biomechanical 3-D finite element modeling of the human breast using MRI data. *IEEE Transactions on Medical Imaging*, 20(4):271–9, April 2001.

[120] R San José Estépar, C-F Westin, and K G Vosburgh. Towards real time 2D to 3D registration for ultrasound-guided endoscopic and laparoscopic procedures. *International journal of computer assisted radiology and surgery*, 4(6):549–60, November 2009.

[121] F M Sánchez-Margallo, J L Moyano-Cuevas, R Latorre, J Maestre, L Correa, J B Pagador, L F Sánchez-Peralta, J a Sánchez-Margallo, and J Usón-Gargallo. Anatomical changes due to pneumoperitoneum analyzed by MRI: an experimental study in pigs. *Surgical and radiologic anatomy : SRA*, 33(5):389–96, July 2011.

[122] D Serby, M Harders, and G Székely. A new approach to cutting into finite element models. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2001*, pages 425–433, 2001.

[123] J H Siewerdsen, D J Moseley, S Burch, S K Bisland, A Bogaards, B C Wilson, and D A Jaffray. Volume CT with a flat-panel detector on a mobile, isocentric C-arm: Pre-clinical investigation in guidance of minimally invasive surgery. *Medical Physics*, 32(1):241, 2005.

[124] J C Simo. A finite strain beam formulation. The three-dimensional dynamic problem. *Computer methods in applied mechanics and engineering*, 49:55–70, 1985.

[125] O Skrinjar, A Nabavi, and J Duncan. Model-driven brain shift compensation. *Medical image analysis*, 6(4):361–73, December 2002.

[126] O V Solberg, T Langø, G A Tangen, R Mårvik, B Ystgaard, A Rethy, and T A N Hernes. Navigated ultrasound in laparoscopic surgery. *Minimally invasive therapy & allied technologies : MITAT : official journal of the Society for Minimally Invasive Therapy*, 18(1):36–53, January 2009.

[127] Y Song, J Totz, S Thompson, S Johnsen, D Barratt, K Gurusamy, B Davidson, S Ourselin, D Hawkes, and M J Clarkson. Locally Rigid, Vessel Based Registration for Laparoscopic Liver Surgery. volume 10, pages 1951–1961, 2015.

[128] J D Stefansic, A J Herline, Y Shyr, W C Chapman, J M Fitzpatrick, B M Dawant, and R L Galloway. Registration of physical space to laparoscopic image space for use in minimally invasive hepatic surgery. *IEEE Transactions on Medical Imaging*, 19(10):1012–1023, 2000.

[129] D Stoyanov, G P Mylonas, F Deligianni, A Darzi, and G Z Yang. Soft-tissue motion tracking and structure estimation for robotic assisted MIS procedures. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2005*, pages 139–46, January 2005.

[130] D Stoyanov, M V Scarzanella, P Pratt, and G-Z Yang. Real-time stereo reconstruction in robotically assisted minimally invasive surgery. *Medical Image Computing and Computer-Assisted Interventation – MICCAI 2010*, 13(Pt 1):275–82, January 2010.

[131] C Studholme, D L G Hill, and D J Hawkes. An overlap invariant entropy measure of 3D medical image alignment. *Pattern Recognition*, 32(1):71–86, 1999.

[132] L-M Su, B P Vagvolgyi, R Agarwal, C E Reiley, R H Taylor, and G D Hager. Augmented reality during robot-assisted laparoscopic partial nephrectomy: toward real-time 3D-CT to stereoscopic video registration. *Urology*, 73(4):896–900, April 2009.

[133] S Suwelack, S Röhl, S Bodenstedt, D Reichard, R Dillmann, T Dos Santos, L Maier-Hein, M Wagner, J Wünscher, H Kenngott, B P Müller, and S Speidel. Physics-based shape matching for intraoperative image guidance. *Medical physics*, 41(11):111901, November 2014.

[134] G Szekely, C Brechbühler, R Hutter, A Rhomberg, N Ironmonger, and P Schmid. Modelling of soft tissue simulation for laparscopic surgery simulation. *Medical Image Analysis*, 4:57–66, 2000.

[135] T Tan, B Platel, J Van Zelst, L Han, T Mertzanidou, S Johnsen, J Hipwell, R Mann, D Hawkes, and N Karssemeijer. Registration of Automated 3D Breast Ultrasound Views. In Anne Martel, John Hipwell, Julia Schnabel, Mads Nielsen, Martyn Nash, Despina Kontos, and Nico Karssemeijer, editors, *Breast Image Analysis - BIA 2013*, pages 25–33, Nagoya, Japan, 2013.

[136] L M Taylor and D P Flanagan. PRONTO 3D: A three-dimensional transient solid dynamics program. Technical Report SAND87–1912, Sandia National Laboratories, Albuquerque, NM, March 1989.

[137] Z A Taylor, M Cheng, and S Ourselin. Real-time nonlinear finite element analysis for surgical simulation using graphics processing units. In *10th International Conference on Medical Image Computing and Computer Assisted Intervention*, Brisbane, Australia, October 2007.

[138] Z A Taylor, M Cheng, and S Ourselin. High-speed nonlinear finite element analysis for surgical simulation using graphics processing units. *IEEE Transactions on Medical Imaging*, 27(5):650–663, 2008.

[139] Z A Taylor, M Cheng, and S Ourselin. High-speed nonlinear finite element analysis for surgical simulation using graphics processing units. *IEEE Transactions on Medical Imaging*, 27(5):650–663, 2008.

[140] Z A Taylor, O Comas, M Cheng, J Passenger, D J Hawkes, D Atkinson, and S Ourselin. Modelling anisotropic viscoelasticity for real-time soft tissue simulation. In *11th International Conference on Medical Image Computing and Computer Assisted Intervention*, New York, September 2008.

[141] Z A Taylor, O Comas, M Cheng, J Passenger, D J Hawkes, D Atkinson, and S Ourselin. On modelling of anisotropic viscoelasticity for soft tissue simulation: numerical solution and GPU execution. *Medical Image Analysis*, 13(2):234–244, 2009.

[142] Z A Taylor, S Crozier, and S Ourselin. Real-time surgical simulation using reduced order finite element analysis. In *13th International Conference on Medical Image Computing and Computer Assisted Intervention*, Beijing, October 2010.

[143] Z A Taylor, S Crozier, and S Ourselin. A reduced order explicit dynamic finite element algorithm for surgical simulation. *IEEE Transactions on Medical Imaging*, 30(9):1713–21, September 2011.

[144] S Thompson, M Hu, S Johnsen, K Gurusamy, B Davidson, and D Hawkes. Towards Image Guided Laparoscopic Liver Surgery, Defining the System Requirement. In *Proceedings of Living Imaging – LIVIM 2011*, volume Published online, pages 1–9, 2011.

[145] S Thompson, G Penney, P Dasgupta, and D Hawkes. Improved modelling of tool tracking errors by modelling dependent marker errors. *IEEE Transactions on Medical Imaging*, 32(2):165–77, February 2013.

[146] S Thompson, J Totz, Y Song, S Johnsen, D Stoyanov, S Ourselin, K Gurusamy, C Schneider, B Davidson, D Hawkes, and M J Clarkson. Accuracy Validation of an Image Guided Laparoscopy System for Liver Resection. In *Proceedings of SPIE, Medical Imaging 2015*, volume In press, 2015.

[147] P-A Ubach, C Estruch, and J Garcia-Espinosa. On the interpolation of normal vectors for triangle meshes. *International Journal for Numerical Methods in Engineering*, 96(4):247–268, September 2013.

[148] P Volino and N M Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum*, 13(3):155–166, August 1994.

[149] M P Wachowiak, X Wang, A Fenster, and T M Peters. Compact support radial basis functions for soft tissue deformation. In *Proceedings of the 2004 IEEE International Symposium*

*on Biomedical Imaging: From Nano to Macro, Arlington, VA, USA, 15-18 April 2004*, pages 1259–1262, 2004.

[150] H-X Wang, A-M Hao, and D Li. Real-time cutting method for soft tissue based on TLED algorithm. In *2010 2nd International Conference on Computer Engineering and Technology*, pages V3–393–V3–396. IEEE, 2010.

[151] A Wittek, T Hawkins, and K Miller. On the unimportance of constitutive models in computing brain deformation for image-guided surgery. *Biomechanics and modeling in mechanobiology*, 8(1):77–84, February 2009.

[152] A Wittek, G Joldes, M Couton, S K Warfield, and K Miller. Patient-specific non-linear finite element modelling for predicting soft organ deformation in real-time: application to non-rigid neuroimage registration. *Progress in biophysics and molecular biology*, 103(2-3):292–303, December 2010.

[153] P Wriggers. *Computational contact mechanics*. J. Wiley & Sons, 2002.

[154] P Wriggers and L Krstulovič-Opara. On Smooth Finite Element Discretizations for Frictional Contact Problems. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 80(S1):77–80, March 2000.

[155] B Yang and T A Laursen. A contact searching algorithm including bounding volume trees applied to finite sliding mortar formulations. *Computational Mechanics*, 41(2):189–205, September 2006.

[156] T Yasunaga, K Konishi, S Yamaguchi, K Okazaki, J-S Hong, S Ieiri, H Nakashima, K Tanoue, T Fukuyo, and M Hashizume. MR-compatible laparoscope with a distally mounted CCD for MR image-guided surgery. *International Journal of Computer Assisted Radiology and Surgery*, 2(1):11–18, May 2007.

[157] G Y Zhang, A Wittek, G R Joldes, X Jin, and K Miller. A three-dimensional nonlinear meshfree algorithm for simulating mechanical responses of soft tissue. *Engineering Analysis with Boundary Elements*, 42:60–66, 2014.

[158] H Zhang, F Banovac, R Lin, N Glossop, B J Wood, D Lindisch, E Levy, and K Cleary. Electromagnetic tracking for abdominal interventions in computer aided surgery. *Computer aided surgery : official journal of the International Society for Computer Aided Surgery*, 11(3):127–36, May 2006.

[159] R Zhang, P-S Tsai, J E Cryer, and M Shah. Shape-from-shading: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):690–706, 1999.

[160] O C Zienkiewicz, R L Taylor, and J Z Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Elsevier Science, 2005.