Real-Time Modelling and Visualization of Soft Tissue Thermomechanical Behaviour for Radiofrequency Thermal Ablation

A thesis submitted in fulfilment of the requirements for the degree of Masters by Research of Engineering (Mechanical)

Jeremy Hills

BEng (Aero) (Hons I)

BBus (Mgt) (Distinction)

School of Aerospace Mechanical and Manufacturing Engineering

College of Science Engineering and Health

RMIT University

August 2015

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the choose an item is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

Jeremy Hills

15/11/2015

Table of Contents

Abstract

A review of current literature indicates a limited knowledge and documentation of thermomechanical response of soft tissue during Minimally Invasive Surgical (MIS) hyperthermia procedures such as Radiofrequency Thermal Ablation (RFA). Furthermore, current models and simulations have not accounted for the temperature-dependence of the stress-strain behaviour of soft tissue. The only quantified data for temperature-dependent stress-strain relationships in literature is yielded from Xu and Lu (2009) and Xu, Seffen and Lu (2008a). As well as this, hardware-accelerated (by use of Graphics Processing Units (GPUs)) heat transfer simulations of RFA had not been documented prior to commencement of this project, and the first conference paper announcing this achievement was published in April of 2014 following research and implementation by NE Scientific LLC.

A computational three-dimensional (3D) simulated virtual model of liver tissue is developed to establish temperature distributions resulting from single point temperature sources in emulation of the RFA heat treatment procedure. The temperature distribution in the virtual tissue domain is produced by Finite Element (FEM) spatial discretization and Finite Difference (FDM) temporal discretization of the Pennes bioheat transfer equation. The modelled temperature distribution is utilized to determine the degree of transient thermal damage to the virtual tissue based on the Arrhenius Burn Integration. Furthermore, the temperature distribution is used in conjunction with linear thermal expansion to model thermal strains and thermal stresses within the virtual tissue, resulting from the heat source. Novel work is undertaken in producing a thermal stress profile of virtual liver tissue under operational temperatures based on temperature-dependent material stress-strain.

The simulation of tissue bioheat transfer and thermal damage is developed in C++ and the High-Level Shader Language (HLSL) with Microsoft's Direct3D11 Application Programming Interface (API), where the numerical solution process is parallelized and accelerated in performance upon an NVIDIA GTX 770M GPU far beyond its performance upon a single thread/core of an Intel® Core™ i7-4700MQ Central Processing Unit (CPU). A maximum mesh resolution is determined for producing visual real-time post-processing output data based on the GPU accelerated simulation performance. A commercial FEM software package (LISA) is used for determining thermal strain and thermal stress distributions from the temperature distribution data.

Examination of simulation results when comparing tissue thermomechanical response for temperature-independent and temperature-dependent stress-strain relationships indicates a dramatic difference in magnitude and distribution of the thermally-induced stresses within the soft tissue. The implications are that RFA simulations must account for this stress-strain temperature-dependence in order to produce remotely accurate stress and strain distributions (both thermomechanical and mechanical) due to the behavioural response of the collagenous biological soft tissue. Furthermore, GPU acceleration is highly recommended for RFA simulation, as the visual real-time maximum mesh resolution far surpasses that of real-time performed on a single modern CPU core.

Acknowledgements

I would like to express my gratitude to my supervisor Dr. Yongmin Zhong for his encouragement, guidance and assistance during my candidature. The provision he offered me for undertaking a Masters by Research was not only timely, but gave me both opportunity and purpose towards developing a deeper knowledge and understanding of programming; a skill and passion that I treasure and am determined to apply in my future career.

List of Figures

List of Tables

List of Equations

Nomenclature

A	Area (m^2) Material frequency factor (s^{-1})
$[C]$	Capacitance matrix
C_{bl}	Blood specific heat capacity $\left(\frac{J}{kg\cdot°K}\right)$
C_{ti}	Tissue specific heat capacity $\left(\frac{J}{kg\cdot°K}\right)$
E	Elastic/Young's Modulus (kPa)
E_a	Activation energy $\left(\frac{J}{mol}\right)$
$\{f\}$	Forcing vector
G	Shear modulus (kPa)
$[K]$	Stiffness matrix
k_{ti}	Tissue thermal conductivity $\left(\frac{W}{m\cdot°K}\right)$
l	Length (m)
n	Boundary normal vector/component
N	Shape function $(N_i, N_j, N_k, N_m \ldots)$
q	Heat flow/flux $\left(\frac{W}{m^2}\right)$
Q_m	Metabolic heat generation rate $\left(\frac{W}{m^3}\right)$
Q_r	Regional heat source generation rate $\left(\frac{W}{m^3}\right)$
R	Universal Gas Constant $\left(R = 8.314\frac{J}{mol°K}\right)$
$\{R\}$	Thermal load vector
t	Time (s)
Δt	Time step (s)
T_{art}	Arterial temperature (°C)
T_{ti}	Tissue temperature (°C)
$\{T_{ti}\}$	Unknown temperature vector
T_0	Initial temperature (°C)
V	Volume (m^3)
w	Weighting function
W_{bl}	Blood perfusion rate $\left(\frac{kg}{m^3\cdot s}\right)$
x, y, z	Cartesian coordinates
α	Coefficient of linear thermal expansion (°C^{-1})
γ	Shear strain
Γ	Boundary of domain Ω
ε	Normal strain
λ	Elongation
ν	Poisson's Ratio
ρ_{ti}	Tissue density $\left(\frac{kg}{m^3}\right)$
σ	Normal stress (kPa)
τ	Shear stress (kPa)
Ω	Thermal damage Domain
∇^2	Laplace operator

Abbreviations

API	Application Programming Interface
CPU	Central Processing Unit
CSR (CRS)	Compressed Sparse Row (storage)
CUDA	Compute Unified Device Architecture
DoF	Degrees of Freedom
ECM	Extra-Cellular Matrix
FDM	Finite Difference Method
FEM	Finite Element Method
FLOPS	Floating Point Operations Per Second
GPGPU	General Purpose GPU (programming)
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HLSL	High Level Shader Language
RAM	Random Access Memory
RF	Radio-Frequency
RFA	Radio-Frequency (thermal) Ablation
SRV	Shader Resource View
VRAM	Video RAM
UAV	Unordered Access View

1 Introduction

Minimally Invasive Surgeries (MIS) (often referred to as “keyhole” surgeries) are medical procedures performed as favourable alternatives to traditional surgeries which can often involve extensive amounts of cutting, suturing, and moving of organ tissue in order to provide surgeon hand and visual access to the location of treatment. MIS techniques often involve needle and probe-like implements which are inserted subcutaneously, providing an alternative to direct surgeon hand access, and imaging techniques such as ultrasonic imaging are used as an alternative to direct visual access (line of sight). Such surgeries minimize risk and exposure of the patient, and reduce recovery time and bed occupancy, however introduce technical difficulties at the expense of the surgeon. Surgeons perform many complicated manoeuvres and thus prior practice is beneficial, if not vital. Just as pilots can train for flight by using flight simulators, surgeons can train with virtual surgical simulators, even with haptic devices which provide input ability and force feedback output ability. Thus, the need for surgical simulators which provide real-time response behaviour is apparent, and realistic complex physical systems must be discretized to the point of computational execution.

Radiofrequency Thermal Ablation (RFA) is an image-guided (ultrasonic) MIS technique which uses a radiofrequency current to generate heat in tissue, when applied from the tip of a needle-like instrument which is inserted through tissue, to the location of treatment. RFA is applied to the treatment of cancer; more specifically addressed in this research project is liver cancer. The heat generated during the RFA procedure is applied to tumour tissue, and induces cellular necrosis of the tissue, ultimately cooking the cancer to death. This cellular necrosis can be quantified with a mathematical function and an absolute value able to represent complete necrosis. That is, thermal damage can be simulated on top of tissue temperature distribution, determining the degree of damage from tissue temperature and other tissue-specific parameters.

It is common knowledge in engineering that materials typically expand when heated; a term referred to as thermal expansion. A steel bar which is uniformly heated to a sufficient extent will expand and elongate. That same steel bar when braced at both ends in order to inhibit its elongation, will produce forces upon the anchoring mechanisms at each end, and produce a stress distribution within the bar as a result. The bar’s state of stress will be one of axial compression, as it does not have the ability to relax into its elongated state, something it tends towards naturally when heated. This behaviour of thermal expansion is frequently applied to heated materials, and can similarly be applied to soft tissue which is heated during the RFA procedure. This behaviour can be termed as thermomechanical response (i.e. the thermally-induced mechanical response of the material); in this case, soft biological tissue.

Current literature extensively examines the means of discretizing and simulating the transfer of heat through biological tissue, and the tissue’s resultant effect upon heat transfer. Multiple methods of discretization of the tissue domain and of the mathematical heat transfer models (partial differential equations; PDEs) are employed in various combinations in literature, and methods of approximating tissue response to thermal load are expressed in great detail. The Pennes bioheat equation is commonly applied to the virtual soft tissue domain in order to obtain a temperature distribution. The Arrhenius Burn Integration is proposed as a quantitative means of describing the level of thermal damage sustained by soft tissue.

Xu, Lu and Seffen (2008) and Xu and Lu (2009) provide some experimental research results regarding the temperature-dependent stress-strain relationship of collagenous soft tissue. When thermal expansion of heated tissue is accounted for, thermal stress can be calculated from thermal strain either by assuming linear elasticity and simply using an elastic modulus, or by some function of strain and temperature. Thus the internal stress state/distribution can be established within the heated soft tissue.

Comas et al. (2008), Courtecuisse et al. (2010) and Luna (2012) discuss the importance and implementation of the Graphics Processing Unit (GPU) in place of the Central Processing Unit (CPU) for the computation of scientific simulations. Comas et al. (2008) demonstrate a degree of speed improvement of simulation performance of over 50 times in favour of GPU over a similar CPU implementation. This improved performance speed allows an increase in maximum possible mesh resolution for real-time RFA simulation. Prior to commencement of this research, RFA heat transfer had not been accelerated by GPU, and the first implementation of such was published in a 2014 conference by Borsic, Hoffer and Attardo (2014). Even accounting for this development, RFA still has not been hardware accelerated for real-time visual feedback (i.e. $\geq$ 25 Hz/FPS), and thus a gap is available within literature wherein a contribution of a real-time hardware accelerated RFA technique is applicable for breaking new ground.

As well as this gap, in-vivo (in the living organism) data of biological soft tissue mechanical and thermal characteristics and response behaviour is very sparse, since testing of living subjects at the very least requires extensive ethics approvals. As a result, literature on thermomechanical response of liver tissue is also sparse, and the temperature-dependent stress-strain relationship thermomechanical response is undocumented for any case as far as is indicated by literature. Furthermore, simulations of soft tissue mechanical and thermomechanical responses under virtual loads representing realistic conditions (and with realistic tissue parameters) are valuable to provide safe, nonintrusive means of determining realistic tissue behaviour. Such behaviour is beneficial and instructional for surgeons to rehearse prior to actual real-life scenarios/procedures.

Thus novelty exists for GPU-accelerated real-time RFA heat transfer and for simulation of temperature-dependent stress-strain relationship thermomechanical response of biological tissue. This research project delves into these two major facets in order to produce novel and new information and results to expand the field of RFA and other hyperthermia simulations. Furthermore, significant conclusions can be reached based on these results.

Primary objectives for this research project are the following:

- To investigate temperature-dependent thermal parameters and compare temperature distribution against temperature-independent thermal parameters, and where possible, compare the results against literature.
- To model tissue damage from the thermal ablation process.
- To accelerate as much as possible, the simulation of heat transfer and thermal damage in soft tissue due to RFA.
- To produce accurate numerical simulation data of the state of stress induced upon liver tissue by the applied heating during an RFA procedure for cases of temperature-independent and temperature-dependent stress-strain relationships; comparing the two scenarios to determine the magnitude and significance of the difference.
- To validate the obtained simulation results against published data in literature, and establishing convergent behaviour of the system.

In order to achieve these objectives, a GPU-accelerated finite element simulation of bioheat transfer and thermal damage is developed using C++ and the High Level Shader Language (HLSL), as well as the Direct3D11 and WIN32 Application Programming Interfaces (APIs), with the simulation being based on in-vivo literature parameters for human liver tissue where possible, or the nearest applicable parameter data. The simulation is applied across a virtual liver tissue domain discretized by the Finite Element Method (FEM), using 4-noded tetrahedral elements (linear shape functions). Thermal strain distributions are subsequently obtained via commercial FEM software from the simulated temperature distribution data. Calculations based on the thermal state of strain as detailed in the methodology, are applied in order to determine the thermal stress distribution within the soft tissue.

The results show that thermal stresses induced in the tissue during RFA (as calculated based upon thermal expansion) are very small (< 1 kPa), however vary greatly between temperature-dependent and temperature-independent stress-strain relation cases in the temperature-affected tissue volume. This significant variation in stress distribution implies that soft tissue simulation of heat treatment procedures (such as RFA) involving operation instrument loads must account for this vital stress-strain temperature-dependence of the collagenous tissue, in order to produce increasingly realistic tissue response behaviour. Furthermore, GPU acceleration of the simulated RFA heat transfer procedure concurs well with literature, regarding the degree of performance improvement.

2 Literature Review

2.1 Background

In order to proceed to a reasoning for this research, a connection must first be made with the purpose of its point of origin. Therefore to discuss biomechanical models of human tissue, an application of such models (radio-frequency thermal ablation (RFA)) must be introduced. RFA treatment procedures would not exist without the frequent need to treat diseases such as liver cancers, and liver cancers in turn would not be treated if the liver's biology was unaffected by their presence, or if its function was not so vital to sustaining a healthy and living being. It is for this reason that the point of entry of this research is at the level of the liver and its important biological functions.

2.1.1 The Liver

The human body's largest internal organ, the liver is a vital organ located in the upper right of the abdominal cavity; a mass of approximately 1 to 1.5 kg in a human adult. The primary functions of the liver include protein and bile synthesis and secretion, detoxification of drugs and poisonous substances, carbohydrate metabolism, and is also involved in the urea cycle (British Medical Association 1990).

It produces blood plasma proteins such as albumin, complement, coagulation factors and globin. Other proteins such as cholesterol are also produced by the liver to aid the transport of fats around the body. The liver regulates the level of amino acids in the bloodstream, converting them into proteins, glucose, and urea. By absorbing poisonous substances and drugs, the liver alters the chemical structure of these substances to be water soluble, and excretes them as bile. Bile removes the waste chemicals and assists in the breakdown of fats. The liver furthermore controls the forwards and backwards conversions between glucose and glycogen, for storing and accessing stored energy when the body requires it (Kolchanov 2006; University of Rochester Medical Center 2014).

Though large, the liver is capable of continuing its function even after up to 75% of its mass is excised, and can recover by growing back to some extent (British Medical Association 1990).

The liver can be described as a cone shaped organ with a red-brown colour, consisting of two main lobes; left and right. Two smaller lobes; the caudate and quadrate lobes, lie behind the right lobe. The lobes are composed of thousands of small lobules. The main type of lobule is termed the hepatic lobule, or the classical lobule. These are comprised of branches of the hepatic vein, hepatic artery, hepatic portal vein and hepatic bile duct. One can imagine a simplified hepatic lobule as a hexahedral prism, where at each of the six corners of the hexagon, one would find portal triads; groups of three vessels running the length of the prism (hepatic artery, portal vein and bile duct). In the centre of the prism's longitudinal axis, lies the central vein. Hepatocytes (liver cells) lie in an arrayed formation, spreading from the central axis to the sides, and are layered in the axial direction in a stacked formation. Spaces between the hepatocyte layers are filled with networks of sinusoidal capillaries, referred to as sinusoids, which receive mixed portal and arterial blood from the portal triads of the lobule, provide exchange functionality between the blood and hepatocytes, and send used blood to the central vein (Kolchanov 2006).

Several major vessels constitute the network for blood transport through the liver. The hepatic artery carries oxygenated blood from the heart to the liver. The hepatic portal vein carries blood laden with nutrients from the small intestine to the liver. The hepatic artery and portal vein branch into a network of smaller vessels which carry blood to the sinusoids. Central veins from the liver lobules connect together to form hepatic veins, allowing blood to leave the liver.

When it comes to sustaining physical loads, the hepatocytes themselves rely on the liver's extracellular matrix for support. The extracellular matrix (ECM) is a soft connective tissue, which functions as a mechanical framework, providing anchorage of the hepatocytes (Bittar 2004). It is composed of multiple types of collagens as well as non-collagenous components such as elastin, laminin, fibronectin and glycosaminoglycans (Kolchanov 2006; Okazaki et al. 2003).

2.1.2 Liver Cancer

Liver cancer can be in the form of primary or metastatic tumours (secondary liver cancers originate from outside the liver); the primary tumours being either a hepatocellular-carcinoma (or “hepatoma”; originating in the liver cells), or a cholangio-carcinoma (originating in the cells lining the bile ducts). Liver cancers manifest themselves through symptoms of weight loss, fatigue, loss of appetite, nausea and abdominal pain.

Hepatomas are the most common form of primary liver cancers, and one of the most common causes of cancer mortalities worldwide. They are strongly linked to Hepatitis B; common throughout Africa and the Middle and Far East, but less common in Western civilisations (British Medical Association 1990). Metastatic liver cancers are more common than primary ones in Western civilisations such as the UK and the US, often originating from cancers in the stomach, pancreas or large intestines.

Metastatic hepatic tumours generally arise from colorectal cancers (between 30% to 50% of cases), and the rate of survival for such cancers averages at 10 months (Delingette & Ayache 2004).

Cancerous tumours (and cirrhotic regions of the liver) tend to exhibit higher stiffness than healthy tissue (Carter, TJ et al. 2005). This is due to a combination of infiltrating T lymphocytes and inflammation causing transformation of fibroblasts into myofibroblasts, which deposit ECM proteins and growth factors into the affected area. These deposited proteins (collagen and elastin in particular) form new crosslinks which stiffen the ECM structure (Venkatasubramanian 2012). Vasculature is subsequently increased by growth factors, which then increasingly facilitates further tumour invasion. Thus, tumorous tissue exhibits greater stiffness and increased vasculature over healthy tissue.

2.1.3 Treatment of Liver Cancer

Established so far, we can see the value of the liver as a vital organ, its purpose and functions, that blood (and thus heat) may be conducted throughout its structure, that the structural component its self (the extracellular matrix) endows the liver its mechanical properties, and that cancers of the liver are suitably common and require treatment.

Excision of tumours from the liver require the patient to be opened up on the operating table to the degree that surgeons can obtain direct access to the offending tumours. Inherently, operating times are long, involve cutting and suturing, blood loss, and long recovery periods, increasing hospital bed stay and occupancy. By 1990, the main treatment available for primary liver cancers was excision of the tumour; often removing a large portion of the liver. Treatment by anticancer drugs was also common. However, secondary cancers were not treatable (British Medical Association 1990). Common to treatment for primary and metastatic liver cancers today however, is a minimally invasive procedure called radio-frequency ablation (also referred to as thermal ablation, RFA or RITA; Radiofrequency Interstitial Tissue Ablation).

In a video interview, Steven A. Rosenberg, M.D. (Chief of Surgery at the National Cancer Institute in Bethesda, Maryland USA) stated: *"The success of surgical therapy is very much related to the size of the tumour, its location in the liver and its ability to be completely ablated, either by surgery or by other techniques such as radio-frequency ablation. That represents the only curative treatment for patients with liver cancer, be it either primary liver cancer, or liver tumours that are metastatic."*

(Sewell 2010).

2.1.4 Radio-Frequency Thermal Ablation Treatment

RFA does not employ an entirely new technology, but rather the application of the technique its self to percutaneous procedures is still young. The concept was demonstrated on tissue as long ago as 1891, and after a period of dormancy, was developed for liver cancer treatment around 1990-1993 (McGahan & Raalte 2005). During these several years, the procedure was developed based upon the workings of the Bovie knife, which implemented a pulsed/damped or continuous current to either cauterize or cut through tissue respectively. A radio-frequency (RF) current causes ionic excitement of the tissue surrounding the source of the current, generating heat in the tissue its self, and ultimately causing coagulation and cellular necrosis of the surrounding tissue. Variation of the application of RF waves through tissue causes differing responses such that a rapid RF application causes a small region of coagulation and charring (necrosis); the charring (and thus reduction in local tissue water content) in turn reducing the ionic excitations and restricting the affected volume to a small size (McGahan & Raalte 2005).

The modern procedure uses this process of RF application, but from the uninsulated tip of an otherwise insulated percutaneous needle. RF is not transferred to tissue along the insulated length of the needle, and thus interior volumes of tissue can become the operative focus of thermal ablation. The needle is hollow with pronged electrodes that can protrude from the tip in an umbrella-like fashion when activated by the surgeon. These prongs assist in increasing the volume of coagulation and necrosis, such that larger tumours can be ablated. The instrument design can be seen in Figure 2-1.

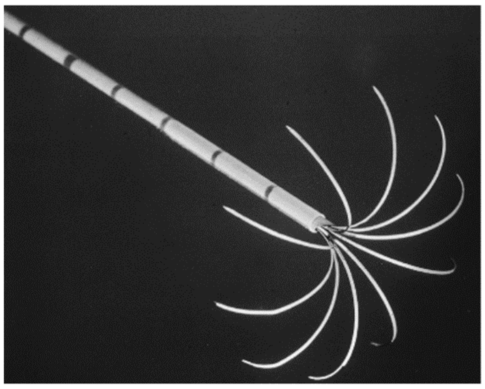


Figure 2-1: RFA Needle and Prongs (McGahan & Raalte 2005).

Tumour ablation can be adversely affected by the presence of large vessels, which act as a heat sink, convecting heat away from the area by heat transfer to the blood flow. Some treatments utilize embolization and occlusion of the main arteries and veins to temporarily decrease the blood flow to the operable region (McGahan & Raalte 2005).

While procedures such as RFA are more optimal than excision for cancer treatment, one must consider the disadvantages to surgical staff performing the procedure. Direct hand access and direct line of sight are no longer available to the surgeon during RFA, as the only percutaneous instrument in the procedure is the RF needle. Minimally invasive procedures such as RFA have thus evolved to become image-guided surgery procedures, utilizing modern technology to convert magnetic resonance imaging (MRI) and computed tomography (CT) data into three-dimensional visualizations that surgeons can rely on for the location of the tumour and the instrument head (Carter, TJ et al. 2005). Preoperative data is obtained to model the patient tissue, and implemented by rigid body transformation, yet actual deformation of the soft tissue between model production and operation is liable to cause a mismatch between visual and operational data, thus a model is needed to accurately reflect such deformations. This is where soft tissue modelling is applied to simulate soft tissue deformations; thermal and mechanical, and provide accurate visual feedback to the surgeons that corresponds with the actual operational deformations of the real tissue (Carter, TJ et al. 2005).

2.2 Medical Simulation

2.2.1 Soft Tissue Simulation

Surgical simulators are the solution to the increased difficulty and level of manual skill required to perform minimally invasive surgery operations such as RFA. Much like flight simulators facilitate the training of pilots, surgical simulators provide an environment where surgeons can practice operations in order to attain the required level of skill, without risk to the patient. Delingette and Ayache (2004) highlight the fact that even obtaining ethics approval to practice in-vivo (in the living organism) operations on live animal subjects is difficult and/or forbidden in specific countries, thus further justifying the need for simulated environments.

(Satava 1996) proposed three categories in which to classify surgical simulators based on the degree of information that they deal with. First generation simulators deal with geometrical data of the anatomy. They facilitate virtual navigation around and through the organs and biological structures, and provide limited interaction between the user and the simulated environment. An example of first generation surgical simulators is the class of virtual endoscopy simulators (Delingette & Ayache 2004). Second generation simulators account for physical properties of the biological tissue in their virtual representations. Thus virtual cutting and suturing manipulations are possible in second generation simulators implementing forces and soft body deformations. Arthroscopic procedures are an example of second generation simulations. Third generation simulators are the most detailed, accounting for physiological responses of the body, coupled with the physical responses. This coupling is complex and difficult to simulate. An example of a third generation simulation is the modelling of mechanical cardiopulmonary interactions.

Delingette and Ayache (2004) describe the common architecture of modern surgical simulators. Most utilize input devices (e.g. haptic input, mouse and keyboard), output devices (e.g. haptic output, screen) and a simulation kernel that controls multiple processes including the soft tissue deformation, visual rendering and the haptic rendering. Since visual and haptic rendering require different update frequencies, these two asynchronous processes can be separated at the expense of greater architectural complexity. Figure 2-2 demonstrates the design of the simulation loop separating the high frequency haptic rendering from the soft tissue deformation and visual rendering. Refer to section 2.10 for further details on these frequencies.

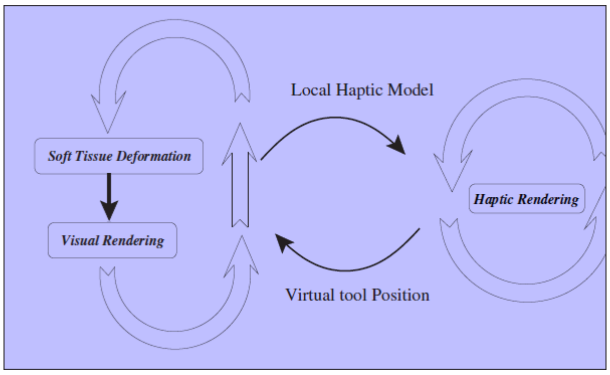


Figure 2-2: Architecture: Two Asynchronous Loops (Delingette & Ayache 2004).

Were the architecturally simplest approach to be taken, the deformation, and visual and haptic rendering would require sequential execution, all within the maximum time required to update the haptic rendering display (i.e. minimum haptic feedback frequency).

Delingette and Ayache (2004) discuss the three main computational method categories used to address soft tissue modelling: direct methods, explicit iterative methods, and semi-implicit iterative methods.

Direct methods solve the static/dynamic equilibrium equations at each iteration; also called quasi-static motion. Pre-computations must be performed in order to achieve reasonable performance for such methods. Since such pre-computations rely on mesh connectivity and geometry, these simulations are not ideal for modelling cutting of the mesh, since cutting changes the mesh connectivity and can invalidate pre-computed data.

Explicit iterative methods are the most commonly utilized methods in literature for soft tissue modelling. These methods employ the use of explicit schemes to calculate deformations as the limit of a converging series in finite time. An explicit scheme will calculate new positions or field data (X_{t+1}) based on the data from the previous iteration (X_t). If the time step used in an explicit scheme is much smaller than the time taken to compute the iteration, the resultant dynamic behaviour of soft tissue can become unrealistic, due to high relaxation time of the scheme. Resultantly, jelly-like behaviour of soft tissue can be observed in cutting simulations.

Implicit and semi-implicit iterative methods are similar, however new field data (X_{t+1}) relies on not only that of the previous iteration (X_t), but field data of the current iteration (X_{t+1}). Implicit methods hence produce linear systems of simultaneous equations, which are computationally costly to solve, hence restricting the number of nodes in a mesh, should rapid or real-time computation be desired. Implicit methods do however have the benefit of greater stability at larger time steps than explicit methods.

2.3 Numerical Simulation

2.3.1 Discretization Methods

The finite element method (FEM) is broadly used in modern industry today, for discretizing arbitrary volumes and geometries into a mesh of volumetric elements that enable simulation of fluid flows, heat transfer and stress analysis amongst others. It is an accurate method when applied correctly, however produces a linear system of simultaneous equations to be solved, thus requiring some form of matrix inversion to determine the unknown field quantities, and increasing computational cost over simpler alternative methods such as Finite Difference Methods, Boundary Element Methods, and others (such as the simple but less accurate Mass-Spring Method). Delingette and Ayache (2004) advise that no FEM software implementations dealt with real-time deformation at the time their research was published.

The Boundary Element Method (BEM) has the advantage over FEM of not requiring a volumetric mesh to be constructed. When the mesh topology is required to be modified by such procedures as cutting however, the change in connectivity makes the method unsuitable. It is well suited for simulating linear elastic isotropic and homogeneous materials.

The Finite Difference Method (FDM) is perhaps one of the most mathematically and computationally simple methods of discretization, and is perfectly suited to homogeneous geometries; situations where the domain is discretized over a structured grid. It is simple to implement in such cases, however is less appropriate for addressing unstructured meshes without utilizing coordinate transformations between the irregular physical and regular computational representations of the mesh. Özisik (1994) discusses that these transformations become increasingly complex towards being impossible for most multidimensional cases. The Boundary Fitted Coordinate Method has evolved from this shortcoming, and utilizes the solutions of partial differential equations over the domain in order to obtain the transformations required to map the physical domain to the regular computational domain. Thus FDM retains its simplicity, yet results in similar equations to FEM in the same scenario. Delingette and Ayache (2004) advise however, that FEM and FDM differ significantly when non-linear elasticity is accounted for, and FDM has not been proven to converge towards the correct solution as FEM does as mesh resolution increases.

The Mass-Spring Method is commonly applied for soft tissue modelling involving small strains and deformations, whereas a majority of operational procedures involve large deformations and strains (Misra, Ramesh & Okamura 2010). RFA is one such procedure involving such strains and displacements during needle insertion and removal. Perhaps during the thermal conduction its self, it would seem that physical instrument motions are minimal, however tissue response to thermal loads can vary as coagulation, necrosis and tissue shrinkage take place, thus as well as its limited accuracy, the mass-spring method is not so suited to modelling tissue response during a RFA procedure. Furthermore, Carter, TJ et al. (2005) mentions that a major drawback of the Mass-Spring Method, is that the mesh parameters have no physical analogue, making it difficult to incorporate constitutive laws.

2.3.2 Numerical Methods

A vast variety of numerical solution methods exist for applying to solve the systems of linear equations, which arise from the implementation of FEM. They are typically categorised as seen in section 2.2.1: direct methods, and iterative methods; explicit and implicit (/ semi-implicit). Direct methods such as Gaussian Elimination, Jordan Elimination, and Gauss-Jordan Elimination are considerably more computationally expensive than iterative methods (Shen, Zhang & Yang 2005a). As mentioned in 2.2.1, pre-computation is typically applied to the model to accelerate the solution process. Iterative solvers such as Jacobi Iteration or Gauss-Seidel Iteration are generally much faster although solution convergence to within a specified margin of error must be addressed.

Both Liszka and Orkisz (1980) and Courtecuisse et al. (2010) implement the Gauss-Seidel iterative technique, however apply a projected variation to accelerate convergence, at the cost of some complexity of the technique. The Gauss-Seidel method, while faster than alternative iterative methods, is still criticized as being slow (Shen, Zhang & Yang 2005c), and is classified as a “stationary” iterative method. The successive over-relaxation method is still regarded as slow when compared to modern projection methods. In fact, literature promotes Krylov subspace methods; particularly a subspace method called GMRES (generalized minimal residual algorithm) (Karaa, Zhang & Yang 2005; Shen, Zhang & Yang 2005a, 2005c).

The conjugate gradient method can accelerate the solution of a system of simultaneous linear equations, where the coefficient matrix is symmetrical (Carter, TJ et al. 2005). Matrix preconditioning such as incomplete LU and Choleski techniques can also be applied to symmetric matrices, prior to their numerical solution.

For the sake of simplicity of implementation for this research project, a simple Gauss-Seidel iterative solver is developed and implemented for solving the system of simultaneous linear equations. It provides further benefits such that iterative methods are much more easily parallelizable by nature, compared to direct methods.

2.3.3 Hardware Acceleration Methods

One of the most demanding requirements for RFA and similar surgical simulations is that feedback is encountered in real-time; a big step to take when some number of hours may be spent on computation time for more comprehensive bio-thermomechanical soft tissue models, particularly those employing FEM (Prakash 2010). Relatively new to the scene, local graphics processing units have been utilized in recent years to accelerate the solution of these models (Comas et al. 2008; Courtecuisse et al. 2010). Frequently taking advantage of matrix preconditioning, highly parallelized operations are sent to the GPU to perform, and solutions can be achieved by up to 53.6 times increased speed as compared to CPU operation (Comas et al. 2008). Figure 2-3 demonstrates the ratio of improved GPU to CPU computation time achieved by Comas et al. (2008).

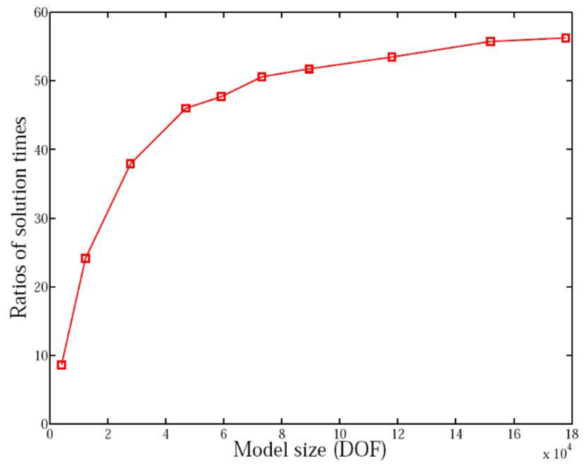


Figure 2-3: Ratio of CPU to GPU solution Times (Comas et al. 2008).

Borsic, Hoffer and Attardo (2014) present the first known published example of GPU-accelerated RFA temperature distributions, accomplished by building a Finite Element simulation with C++, (OpenGL / CUDA) based on the Pennes bioheat equation. (This conference paper was published after the time at which part of this research project's novelty was determined to be GPU accelerated RFA simulation.) In contrast to the simplicity of this research project (single heat point source), a mesh of the umbrella-like pronged RFA electrode was modelled for their simulation. Large time steps (0.25s) were utilized on a detailed mesh in order to simulate a 10-minute procedure in little more than 3 minutes. The update frequency of only 4 Hz does not imply real-time visual feedback of the simulation however. View section 2.10 for more information.

While multi-core CPU architectures are gradually providing an improvement in processing for CPU-based simulations, the current trend is to utilize the GPU to perform numerical solutions in simulations. Courtecuisse et al. (2010) recommend the use of GPU hardware acceleration techniques to undertake mass-spring networked simulations. Comas et al. (2008) utilize the GPU to accelerate a nonlinear FEM simulation of soft tissue modelling, demonstrating the time-saving benefits of the application of this relatively young technique.

Luna (2012) introduces the concept of GPU hardware accelerated computing by its industry/research related title: General Purpose GPU (GPGPU) programming. It is explained that the massive parallelism potential offered by modern GPUs has arisen out of designs purposed for vertex and pixel computations. Typically, GPUs are applied to rendering virtual geometry. GPU operations on vertex positional data involves transforming from local space to world space (local coordinate system to a global coordinate system, via translations, rotations and scaling), from world space to view space (transforming from global coordinates, relative to camera/view position) , and from view space to projection space (from view coordinates to the screen plane). Pixels in areas between vertices are interpolated by the GPU to fill them so as to represent solid surfaces where required, and colour or texture data is interpolated for each pixel of visible surfaces. Such processes must be performed on all vertices and all pixels for the desired resolution, each time the application is required to update the display. Thus GPUs have evolved to contain a multitude of cores; not individually 1:1 as powerful as CPU cores, but with hundreds or thousands of them, in order to efficiently apply the same algorithm/process simultaneously to as much data as possible. Luna (2012) warns however, that GPGPU programming is beneficial only for "data-parallel" algorithms, where a multitude of data requires similar operations performed, otherwise hardware accelerated gains will not be substantial.

The dramatic performance improvement (speed increase) afforded by GPGPU implementation allows larger simulations to be conducted in the same execution time as smaller simulations on a CPU. This translates to FEM simulation in that the number of nodes and elements in a mesh affect the total number of processes required to be completed by the computer per iteration to solve a distribution of some field value such as temperature. When the resolution of a mesh increases; that is, the number of nodes and/or elements increases, then the number of required computational processes per iteration increases for the simulation. The GPU's ability to much more rapidly process similar calculations on nodes and elements through parallelization (thanks to its architecture), means that a greater mesh resolution can be simulated by the GPU in the same time that a mesh of a much smaller resolution might take on a CPU.

2.3.4 The Programmable Graphics Pipeline

The programmable graphics pipeline is a series of programmatically modifiable stages in the overall GPU rendering sequence that is necessary in order to generate an image based on the view of a virtual camera. See Figure 2-4 to view the various stages of the pipeline. Arrows indicate reading operations from, or writing operations to pipeline memory resources stored in Video RAM (VRAM) on the GPU.

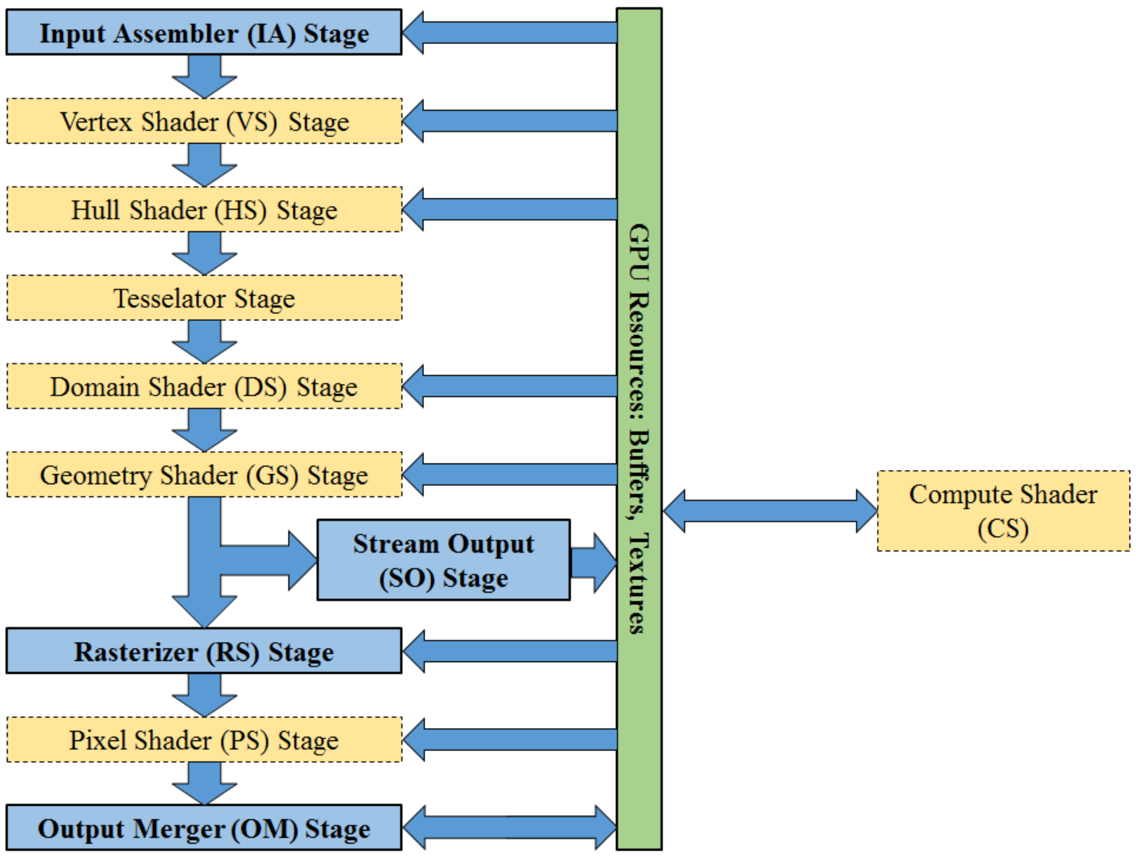


Figure 2-4: Programmable (Rendering) Pipeline

Shaders are programmable components (think of them as GPU programs) of the pipeline that can be modified and manipulated by the programmer, allowing design and implementation of different effects, such as blurring, animations, particle effects, shadows and lighting, and many more. Shaders in Figure 2-4 are shown in yellow (with dotted lines). Note how the compute shader is not part of the rendering pipeline, however sits to the side. It can be utilized separately, such that it can mix with graphical rendering processes or be applied solely to GPGPU programming.

Programming the graphics pipeline is commonly performed using one of two programming libraries which enable communication between the CPU and graphics hardware: OpenGL (Open Graphics Library) and Direct3D. OpenGL is an industry-standard application programming interface (API) which provides cross-platform support between operating systems, functioning on Windows, Mac OS X and Linux systems. Direct3D is a Microsoft proprietary industry standard API for Windows platforms only. OpenGL and Direct3D are utilized in development of commercial 3D software tools and entertainment products. OpenGL's GPGPU programming language equivalent is OpenCL (Open Computing Language), DirectX (10, 11+) exposes DirectCompute and the NVIDIA company has developed CUDA (Compute Unified Device Architecture); all for GPGPU programming by utilizing compute shaders.

CUDA is frequently used in research for GPGPU programming in the literature (Comas et al. 2008; Courtecuisse et al. 2010; Dillenseger & Esneault 2010). CUDA is however, designed specifically for use with NVIDIA graphics hardware architecture, and a system with the same operating system but with an ATI/AMD Radeon GPU for example, will not be compatible for executing a GPGPU simulation programmed using CUDA.

Microsoft DirectCompute is functional on Windows platforms for both NVIDIA and ATI/AMD graphics hardware, utilizing the same code. A DirectCompute API exists for C++ (as it does for OpenCL and CUDA), as well as for the C# language. Due to its extensive use and documentation, as well as the ability to also utilize it for graphical display output (simulation post-processing), the DirectX DirectCompute C++ API will be utilized in this research project, along with the C++ programming language, and the HLSL (High Level Shader Language) to write hardware-accelerated simulations (see section 3.5.2).

2.3.5 Floating-Point Computation

Referring to section 3.5.2, the selected programming language for this project is C++, and the shader programming language is HLSL.

In both languages, floating point numbers (“float”) are represented using 4 bytes (32 bits; 0s/1s). HLSL requires that data structures are 16-byte aligned. This means that HLSL variables are typically 16 bytes, packing 4 floats into a single vector variable (4 floats multiplied by 4 bytes = 16 bytes). This design is optimal for passing variables containing normalized floating point data for red, green, blue and alpha (transparency) colour components, as well as multidimensional vertex coordinate data (x, y, z, plus 1 float padding).

Thus, a floating point vector variable in HLSL can have 2, 3 or 4 float components utilized, however 2, 1 or 0 float components respectively will occupy space as padding. For performance optimization’s sake, it is important to use “float” (treated as scalar), “float2”, “float3” and “float4” where most appropriate in calculations.

Mathematical vector operations in HLSL are optimized for such four-component inputs. Variables such as “float4” (four component vector), and “float4x4” (matrix of 4 rows, 4 columns, float elements) can be directly input into built-in HLSL vector/matrix multiplication functions, and be computed optimally. For example, rather than representing a 4-by-4 matrix by creating an array of 16 floats, and then addressing them each multiple times in order to calculate the matrix determinant, it is optimal to instead use the built-in HLSL function “determinant(M)”, where the parameter passed as “M” is of float4x4 type.

2.3.6 Multi-Core Architecture, Parallelized Processing and Thread Synchronicity

GPUs are architected with multiple cores for highly parallelized computations. The current popular high-range NVIDIA GPU: the GeForce GTX 980, has 2048 CUDA cores, each operating at approximately 1.1 GHz. NVIDIA's latest top-end GPU: the Titan X, has 3072 CUDA cores, each operating at approximately 1.0 GHz. The Titan X is capable of up to 7 TFLOPS (TeraFLOPS; Trillion FLoating-point Operations Per Second).

Cores are grouped in streaming multiprocessors (SM) on board the GPU. Luna (2012) mentions that the past NVIDIA multiprocessor architecture ("Fermi"); last used with the GTX 580 GPU, contained 32 cores per SM, while the current "Maxwell" SM architecture contains 128 cores.

A single thread is a sequence of programming instructions operated upon by a core. The parallelized nature of GPU architecture enables numerous threads to execute simultaneously on separate cores. For example, a hypothetical GPU of 64 cores could execute 64 threads simultaneously. If it were required to double the values held in data array of 128 elements, then it could either complete two threads per core, or be programmed to operate on two array elements per thread.

Threads are managed on GPUs in thread groups. Since cores are grouped in multiprocessors on the GPU, a problem should be broken up such that each multiprocessor operates on its own thread group in order to distribute the load across all cores. It is commonly recommended that multiprocessors be assigned at least two thread groups each, in order to hide latency/stalls (Comas et al. 2008; Luna 2012). Such latencies are introduced when a particular thread group is waiting for access to memory for example, and the latency is hidden by the multiprocessor switching to operating on a different thread group while the previous one waits. Thread groups should be some whole multiple of "warp"/"wavefront" size of the hardware. NVIDIA graphics hardware "warp" sizes are 32 threads, and ATI/AMD graphics hardware "wavefront" sizes are 64 threads (Luna 2012). For a portable application, the logical step is to use a multiple of 64, as it will be compatible with both NVIDIA and ATI/AMD hardware, however for this research project, a thread group size of 32 was employed on NVIDIA hardware, as it performed fastest for the programmed implementation of the simulation.

In order to parallelize a process, data to be processed is first typically arranged in large arrays or texture resources. Textures can be 1D, 2D or 3D, hold a multitude of various data types (i.e. colour components for pixel shader input/output), however they have a maximum size limit of 16384 elements per dimension (width/height/depth). Arrays of data can alternatively be quite flexibly defined and stored in buffers. The method for parallelizing an operation on an array (in a buffer), or a texture, is to distribute the load across the total threads which are to be executed on the GPU. Examining Figure 2-5, the shader can be programmed such that each thread accesses *n* elements of an array, where *n* is typically the array size divided by the number of threads. The array indices accessed for each thread could be incremental, or clumped.

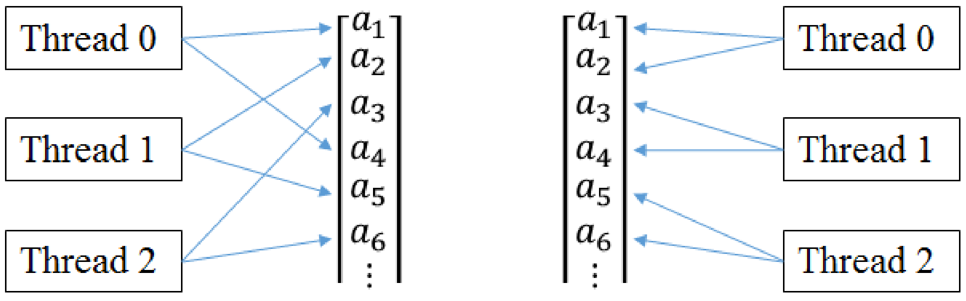


Figure 2-5: Two Examples of Array Indexing

Threads and thread groups can be designed in multidimensional grids of threads/groups to suit the problem type. A grid of threads or thread groups can be 1D, 2D or 3D. The benefit to any particular design of grid applies to how the data is parallelized/accessed. For example, for a 1-dimensional array of data elements, it makes sense for a thread group of 64 threads to be of dimensions 64×1×1, whereas a 2D texture might not only possibly adopt a row-at-a-time behaviour of 64×1×1 threads, but alternatively benefit by 2D thread groups of 8×8×1 such that it can be operated on in a tile-like manner. The grid of thread groups can be implemented in a similar way. Thus the nature of the problem dictates or suggests the implementation of thread and thread group design.

Furthermore, each thread contains an identification number; several in fact. It has an x, y and z ID local to its thread group, and each thread group has its own x, y and z ID. These IDs (integers) can be utilized to compute index locations for accessing or writing to buffer or texture resources. Examining Figure 2-6, a 2D resource (2D array in a buffer or 2D texture) of 16 rows by 22 columns of data is operated on by 3×2×1 thread groups; each of 8×8×1 threads. Once the thread and thread group IDs have been used to determine the index location in the resource, the algorithm in the shader can perform the operations desired. Extraneous threads are those whose IDs cause them to attempt referencing a non-existent index/location in the resource, and can be told not to attempt operations by encapsulating the shader code with a conditional 'if' statement. If the IDs are within a certain range, do the computation. If not, don't.

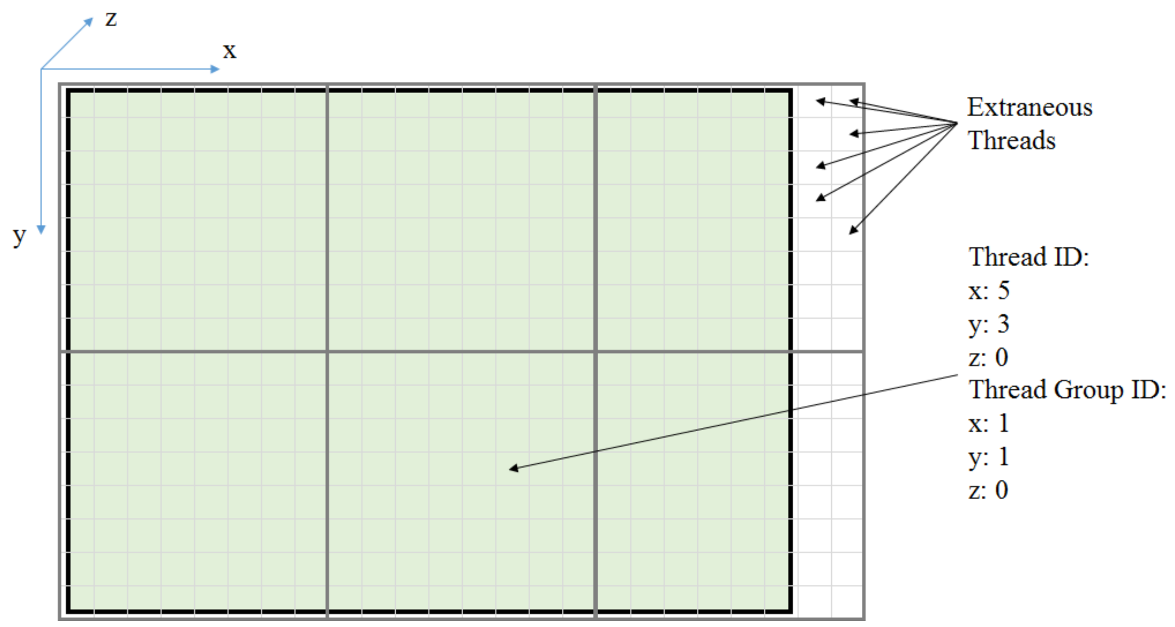


Figure 2-6: Example Thread & Thread Group Grids

While thread group dimensions are specified in the shader code, the dimensions of the grid of thread groups its self is specified on the CPU side in a "dispatch" call. A "dispatch" call is a command that adds the currently set shader to a queue in the GPU to be executed. Dispatching a grid of thread groups causes the shader code to be executed for each thread in each thread group.

Shader resources can be considered as device resources (buffers, textures) or group shared memory. Buffer resources which simply supply information to the shader and which needn't facilitate writing accesses, are described to the shader by Shader Resource Views (SRVs). If a buffer must be written to, then since the shader cannot guarantee exactly how (what order) the buffer will be written to, it is described to the shader by an Unordered Access View (UAV). Group shared memory is much faster for reading/writing (temporarily) than to a device resource via UAV (or SRV), however is limited to 32 kilobytes per group, and excessive consumption can actually hinder performance (Luna 2012).

Synchronization of threads is sometimes necessary for correct operations. There is no guarantee of knowing whether the command reached by a particular thread has yet been reached or already executed on another thread, unless some means of synchronizing threads can be implemented. Three types of synchronization can be employed in HLSL. Group memory barriers, device memory barriers, or ("all") barriers to both types simultaneously (Zink, Pettineo & Hoxley 2011). Group memory barriers ensure that all pending write accesses to group shared memory have been completed before the shader continues. Device memory barriers ensure that all pending write accesses to device resources through UAVs have been completed before the shader continues. Each of the three barrier types can also include a variant that synchronizes all threads in a thread group to the same point before continuing. Global synchronization of all threads in a dispatch is different however. By synchronizing every thread in a dispatch to each other, the parallelized nature of the process would be negatively impacted, so no particular command facilitates this within HLSL. Rather, global thread synchronization is achieved by using multiple shaders and dispatching them in sequence. When one shader has completed, the next will proceed.

One further feature of HLSL is that of atomic functions, specifically "interlocked" atomic functions. Since not even group/device memory barriers can prevent threads from attempting to write a value to the same location at the same time, it may be necessary to enable interlocked capability. For example, suppose two threads simultaneously try to add a number to a particular destination in a resource. Suppose the current value held at the location in the resource is 12.5, thread A seeks to add 1.0, and thread B seeks to add 3.5. Both threads access the current value of the resource and subsequently attempt the same writing command at the same time. The operational result which thread A seeks to write is 13.5, while B's result is 16.0. A writes first to the destination, followed by B, which overwrites A's result. Thus the simultaneous computation of $12.5 + 1.0 + 3.5$ becomes 16.0 instead of the expected 17.0 Enter interlocked atomic functions. These allow write accesses to virtually queue at a destination. However, the interlocked functions in HLSL only support whole integer operations, and have a significant negative impact on performance when used excessively.

2.3.7 Matrix Compressed Sparse Row Storage

Matrix Compressed Sparse Row storage (CSR) is a necessary technique to implement to reduce storage memory for large sparse matrices within software simulations, as well as speeding up row/column operations. Due to a large number of global DoF of the system, the coefficient matrix which must be inverted by iterative methods (see section 2.3.2) will be of dimensions $n \times n$ where n is the number of global DoF. Such matrices are typically sparse, due to the high number of global DoF/nodes in a mesh, and the limited connectivity between nodes in the mesh. This means that computationally, a vast percentage of the matrix stores no useful information (0's), and occupies valuable space in memory (RAM if on CPU, or VRAM if on GPU). Referring to section 2.3.5, it can be seen that should a data structure be stored on VRAM on a GPU using HLSL, the data will be 16-byte aligned, even if each individual unit of data is less than 16 bytes. If aggressive packing is not utilized (4 × 4 byte float structures), then each matrix element could be considered to occupy 16 bytes. For a mesh of 1000 nodes with a single DoF (temperature) per node, the total memory occupied by the coefficient matrix will be $16\ bytes \times 1000 \times 1000 = 16$ MB.

CSR (also referred to as CRS; Compressed Row Storage) utilizes three one-dimensional arrays to store nonzero matrix data.

For example, a sparse matrix:

$$\begin{bmatrix} a_{00} & a_{01} & 0 & 0 & 0 \\ a_{10} & a_{11} & a_{12} & 0 & 0 \\ 0 & a_{21} & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{43} & a_{44} \end{bmatrix}$$

can be stored using CSR as:

$$value = \begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{32} \\ a_{33} \\ a_{34} \\ a_{43} \\ a_{44} \end{bmatrix}, col = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 2 \\ 1 \\ 2 \\ 3 \\ 2 \\ 3 \\ 4 \\ 3 \\ 4 \end{bmatrix}, rowptr = \begin{bmatrix} 0 \\ 2 \\ 5 \\ 8 \\ 11 \end{bmatrix}.$$

Obviously, this actually requires more storage space for a system of a mere 5 global DoF, however for larger systems (i.e. similar to the sparse matrix presented), a system of 7 or more global DoF would begin to see reduced memory costs for saving the sparse matrix in CSR format. For example, for a system of similar connectivity to that of the sparse matrix already presented, 1 MB of 16-byte aligned (non-aggressively packed) memory could store a typical system of 250 global DoF, compared to CSR data for a sparse matrix of a system of 8,929 global DoF. Brute-force (default) storage is of the order $O(n^2)$, while CSR is of the order $O(n)$.

The technique is utilized in the simulation software developed in this research project and is further explained by Saad (2003). An added benefit is gained during Gauss-Seidel iteration over a CSR sparse matrix, as all zero elements are appropriately skipped.

2.4 Current Simulation Procedures

Cook (1995) outlines the procedure for finite element simulations to calculate thermal stresses by the following steps:

- Establish the temperature field on the mesh.
- For each element, restrain all nodal DoF (displacement) of each node, and compute loads applied by the element upon those nodes due to change in temperature.
- Assemble the elements and element loads from the previous step.
- Solve the resultant system of equations for nodal DoF (displacements) and then compute element strains; derived from the element's nodal DoF.
- Compute the element stresses from the strains.
- Superpose initial stresses upon the calculated stresses.

Furthermore, Cook (1995) advises that mechanical loads may be superposed on thermal loads during the assembly phase. Thus, the thermal loads calculated due to a changing temperature distribution over the mesh can be added with instrument mechanical loads to produce the total strains and stresses.

Shen, Zhang and Yang (2005a) undertake the simulation procedure similarly, by first computing the heat transfer process throughout the mesh, using the Pennes bioheat equation. The equation is however, discretized over a regular domain, using the Finite Difference Method. The Finite Difference Method applied is not the usual 7-point central-difference scheme, but rather a 19-point scheme that accounts for the contributions of an increased number of nearby/adjacent nodes in order to increase the accuracy of the outcome. In treating the domain as a porous medium (see section 2.5.5), a thermoporoelastic model is then applied, where stress, strain, pressure and temperature are related. Elastic deformation (displacements) and consequently thermal strains and stresses are calculated using a modified Duhamel-Numann equation.

Karaa, Zhang and Yang (2005) simulate bioheat transfer for hyperthermia conditions such as those found in RFA procedures. They utilize the Pennes bioheat equation for heat transfer and model skin surface temperature distributions under particular conditions, as well as deep tissue heating, displaying a typical temperature distribution for one or three point heat sources, as shown in Figure 2-7 and Figure 2-8. They apply an implicit solver with a very large time step (20 s) for their solution of heat transfer from skin inwards to deep tissue. The figures provided nevertheless show the computed temperature distribution in the tissue at steady state.

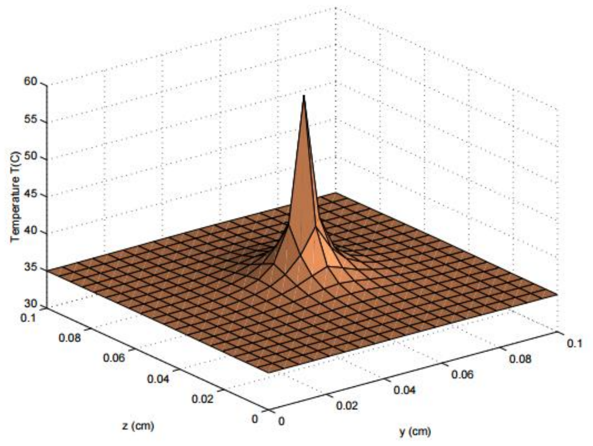


Figure 2-7: Tissue Temperature Distribution – Single Point Source (Karaa, Zhang & Yang 2005)

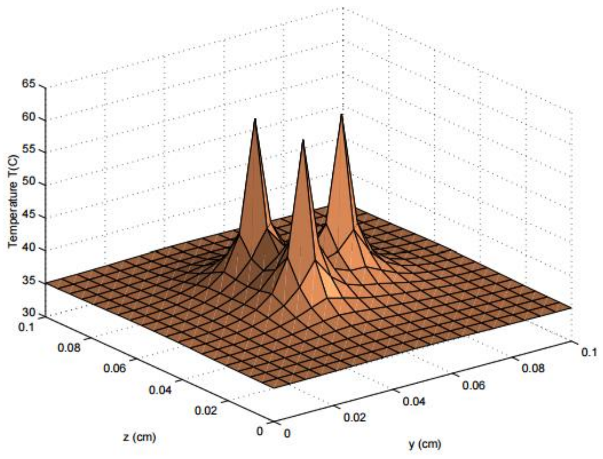


Figure 2-8: Tissue Temperature Distribution – Multiple Point Sources (Karaa, Zhang & Yang 2005)

Terzopoulos, Platt and Fleischer (1991) demonstrate a long-applied use of the mass-spring model to soft tissue deformation, relating temperature to stiffness of the spring units such that tissue deformation response becomes associated with temperature. At some threshold temperature, thermoelastic units have their stiffness set to zero, and become governed by particle forces to interact as a fluid, thus melting into a gel-like state.

Since RFA is increasingly used to treat organ cancers, the importance of producing refined organ models is heightened. Image-guided surgeries such as RFA require detailed and accurate output for their success. When it comes to implementing FEM for organ discretization in a soft tissue model, the computational expense is generally a hindrance, and the use of coarser meshes as well as linear elements are often the focus for performance improvement (Zhong et al. 2012).

2.5 Tissue Thermomechanics

2.5.1 Bio-Thermomechanics and Biomechanics Literature

Tissue bio-thermomechanics can be defined as the response of the tissue under thermomechanical loading, implying potential damage; that is, the thermal denaturation of collagen in the tissue (Xu & Lu 2009). Xu and Lu (2009) investigate the Arrhenius burn integration for modelling tissue damage and necrosis, and also analyses the temperature-dependence of some thermal parameters and of the stress-strain relationship of (pig) skin tissue. Xu, Seffen and Lu (2008b) mention the lack of current literature accounting for temperature-dependence of mechanical properties (skin) under hyperthermia treatments, as well as thermal-mechanical interactions.

Deshan et al. (2007) discuss that one major facet of bio-thermomechanics that is currently lacking in literature, is that of water-related processes in biological tissue. Research regarding water content and processes such as diffusion, evaporation and condensation during heating and thermal denaturation is regarded as sparse, and there is added complexity of involving such parameters in simulation. The Pennes bioheat equation (most widely used) does not account for these behaviours.

Watanabe et al. (2009) evaluate the originality of their work regarding the fact that a temperature-dependent thermo-physical model of a liver is developed, implying that temperature-dependence of thermal parameters in thermal simulations of organ tissue is still relatively new ground.

Delingette and Ayache (2004) suggest that more experimental studies are required before an adequate understanding of liver biomechanical properties are known. The most needed data is that from in-vivo and in-situ analysis, such that derived parameters can be used for furthering realism of soft tissue modelling and surgical simulations. Picinbono, Delingette and Ayache (2003) also discusses the difficulty of validating such simulations without access to raw experimental data. (Also see section 4.5.1.) Furthermore, the source states that quantitative dynamic behaviour is known for very few anatomical structures. Skin, muscle and myocardium are several examples of anatomical structures wherein some quantitative data is known, data of which is demonstrated in (Xu & Lu 2009); see Figure 2-13.

Current analyses in literature mention the lack of known quantitative data of in-vivo tissues and their deformation behaviours, particularly under the loadings applied in actual surgery (Brown, Jeffrey D et al. 2003; Carter, FJ et al. 2001). The quantitative behaviours of in-vivo abdominal organs are rarely documented for minimally invasive surgery, while quantitative tests are frequently performed on ex-corpus organs, or in-vivo on animal subjects, such as pigs and rhesus monkeys (Brown, Jeffrey D et al. 2003), Melvin et al. in (Miller 2000).

While rare, some quantitative data is defined in literature for mechanical characteristics of the human liver (Carter, FJ et al. 2001). Indentation measurements taken during open surgery with ethical clearance and informed patient consent, have yielded in-vivo liver tissue stiffness information which is covered in section 2.5.4.

2.5.2 Elastin & Collagen

As previously discussed, the liver's support structure; the ECM, is primarily composed of collagens, elastin, laminin, fibronectin and glycosaminoglycans.

Fibronectin is a binding protein, and serves to bind other ECM components such as collagen. Laminin; a large crucifix-shaped protein, also functions for ECM binding. Glycosaminoglycans are long chains of molecules that attract water, and thus function in lubricating processes.

Collagen is the major load carrying constituent of soft tissues. It is a protein composed of a triple helix of amino acids, in the form of chains, which interlink with one another to form fibrils and thus, fibres. Multiple types of collagen exist in the body (Type I – Type XIX), however of the collagen content present in the liver ECM (80 mg/g protein), the content composition is mainly of the first six types; the two most dominant being (40-50%) Collagen I, and (40-50%) Collagen III (Okazaki et al. 2003). When collagenous tissue is heated, it can exhibit elastic or plastic deformations based on collagen content, maximum temperature during heating, exposure time, mechanical stress applied during the heating process and aging (Xu, Lu & Seffen 2008). Due to the structure of the Collagen I helix, it is particularly susceptible to a heating induced process called thermal denaturation.

Thermal denaturation occurs when sufficient heat is attained to break intramolecular links in the collagen helix, causing the collagen molecules to slip from one another, and the fibrils to convert from organized crystalline structures to a gel-like state (Xu, Lu & Seffen 2008). Collagen fibres maintain organ function and integrity when in tension, and it is this tension and collagen helix unwinding that causes collagen shrinkage to as little as a third of its original length when denatured (Holzapfel 2000). During the process of RFA, thermal denaturation leads to tissue coagulation at temperatures of about 60°C (Watanabe et al. 2009). When moisture evaporates from the tissue, the tissue necrotises at about 70-80°C. Also note that the water content present in the collagen is strongly associated with the temperature at which thermal denaturation occurs (Xu & Lu 2009).

When tissue bio-thermal properties such as thermal conductivity and specific heat capacity are considered, we find that not only are they not constant, but for specific heat capacity, a highly non-linear variation occurs with temperature. Examining Figure 2-9, it can be observed that the pig flank skin sample when heated, undergoes a rapid rate of thermal energy absorption between 60-66.83°C (Xu & Lu 2009). The excess thermal energy is absorbed by the process of breaking down the collagen structure; that is, denaturation. The same endothermic peak of thermal denaturation is observable in the "linear heating" method results in Figure 2-10, yet applicable to the specific heat of liver tissue. Associating these two graphs, a common peak occurs just above 66°C, where the process accelerates to this point from 60°C. The non-linear relationship between specific heat capacity and temperature could prove valuable to the accuracy of virtual thermomechanical models.

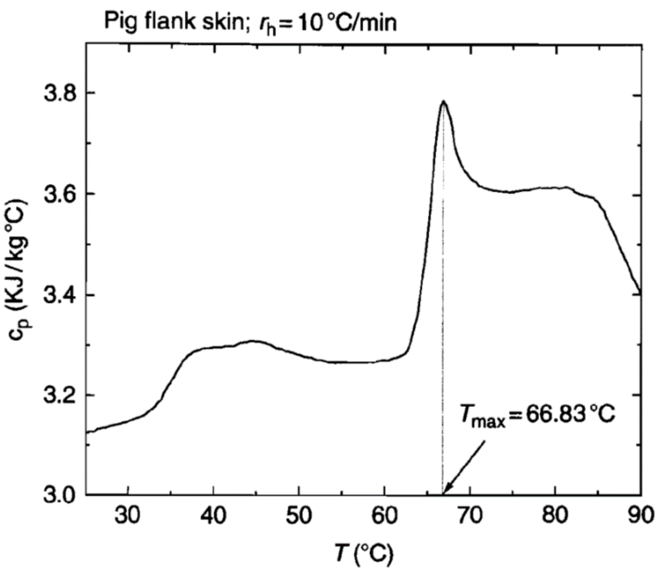


Figure 2-9: Variation of Specific Heat Capacity with Temperature of Pig Skin Tissue (Xu & Lu 2009).

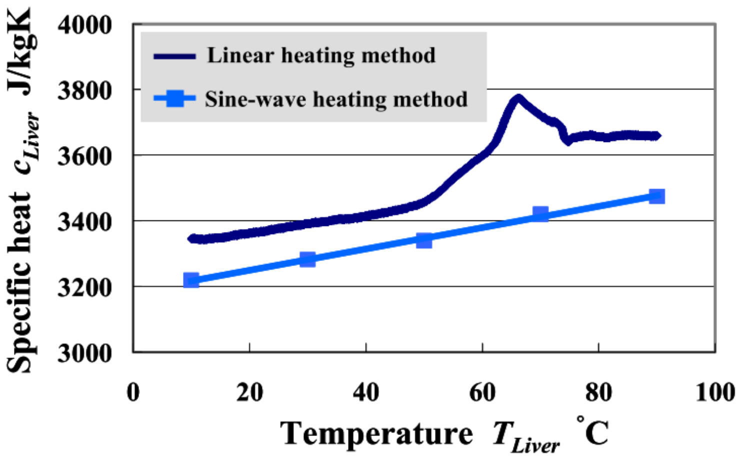


Figure 2-10: Variation of Specific Heat Capacity with Temperature of Hog Liver Tissue (Watanabe et al. 2009).

Note furthermore, the relationship between thermal conductivity and temperature as seen in Figure 2-11. The data points of the four trials and their average, shows a linear relationship of thermal conductivity with temperature. Supposedly, the linear relationship is due to the reduced moisture as temperature increases (Watanabe et al. 2009). The coloured cross sections on the graph represent the physical appearance of the tissue sample at each stage of heating.

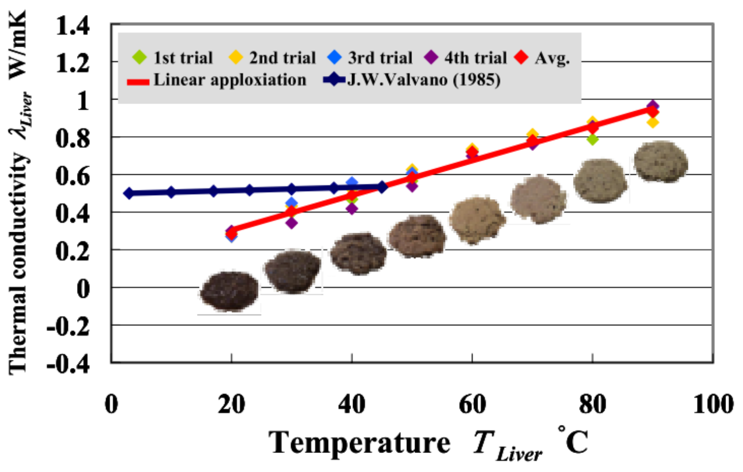


Figure 2-11: Variation of Thermal Conductivity with Temperature of Hog Liver Tissue (Watanabe et al. 2009).

Elastin, in comparison to collagen, is thermally stable. It can retain its mechanical integrity throughout exposure to high temperatures for several hours (Xu, Lu & Seffen 2008). It is described as behaving as a linearly elastic material with relaxation effects much smaller than that of collagen (Holzapfel 2000).

Collagen's response to tensile forces can be broken down into three phases (Holzapfel 2000; Xu & Lu 2009). The first phase is in the absence of loading, where collagen fibres relax in a randomly weaved form, wavy and un-stretched. Low stresses will begin to straighten the collagen, however at this stage, the tissue's resistance to the load is based on elastin's behaviour, and the stress-strain relation is linear. The second phase is the point at which the collagen has straightened enough to roughly conform to the axial direction of the load and commences load-bearing. The stress-strain relationship is non-linear for this phase. At the third phase, the collagen fibres are completely straightened and provide strong resistance to the applied load, stiffening the tissue. The stress-strain relationship here is linear again. The collagen fibres will break when the ultimate tensile strength is reached. Figure 2-13 from Xu and Lu (2009) shows the linear-non-linear-linear relationship of stress vs stretch of the skin samples (subtract 1.0 for strain on the horizontal axis). Note that the mechanical behaviour differs greatly by species. The three phases described by Holzapfel (2000) are quite clear (Figure 2-12). The shallow gradient of the first elastin-dominant phase, the non-linearity of the second phase as the collagen begins to straighten and produce greater resistance, and the linear behaviour of the third phase as the collagen bears the majority of the load.

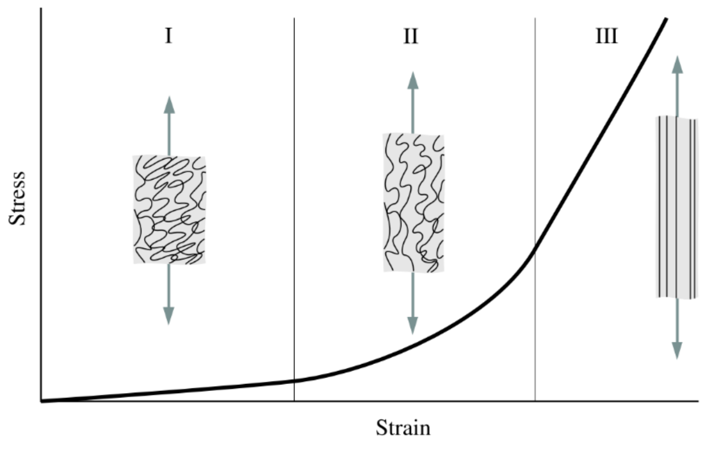


Figure 2-12: Collagen Fibre Morphology (Holzapfel 2000)

When the thermomechanical properties of skin tissue are examined now, as per Figure 2-14 and Figure 2-15, the decrease in stiffness corresponds with the thermal denaturation of collagen with increasing temperatures. The heat and load combine to damage the tissue. As expected, towards 60°C, the sample exhibits a large drop in stiffness, and subsequent smaller drops thereafter, due to thermal denaturation having taken place; the gel-like state of the collagen being unable to bear loads (Xu & Lu 2009). The remaining resistance exhibited by the sample is then primarily due to the elastin, which as discussed, maintains its stiffness despite heating; tending towards perfect linearity from zero strain as the temperature is increased.

When time-dependent mechanical behaviour of tissue is accounted for, tissue is described as having viscoelastic behaviour (Comas et al. 2008; Xu, Seffen & Lu 2008b). That is, it exhibits both viscous and elastic behaviours. Tissue may deform elastically, however over time, stress and strain will vary due to the relaxation of the tissue. Furthermore, other influences can be accounted for, including relaxation factors/coefficients, loading rate, period of loading, preconditioning stress and strain-rate dependence (Carter, TJ et al. 2005; Xu, Seffen & Lu 2008b). Dependencies other than temperature-dependence of stress-strain are not considered in the scope of this research project, however.

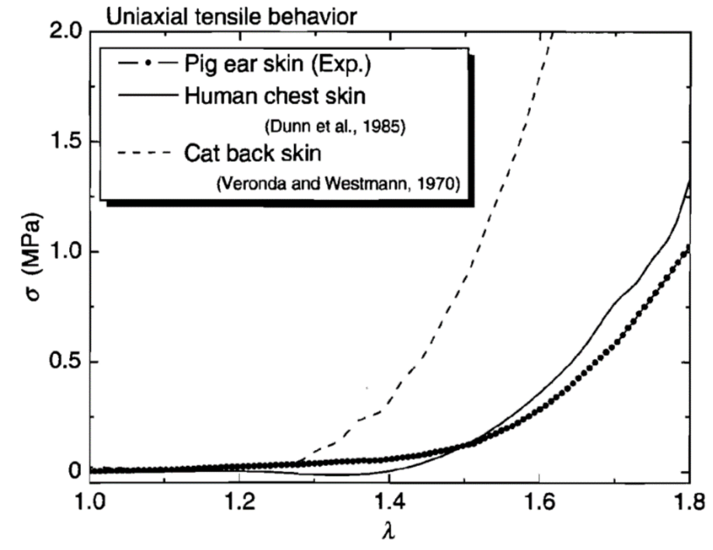


Figure 2-13: Uniaxial Tensile Behaviour of Skin Tissue (Xu & Lu 2009).

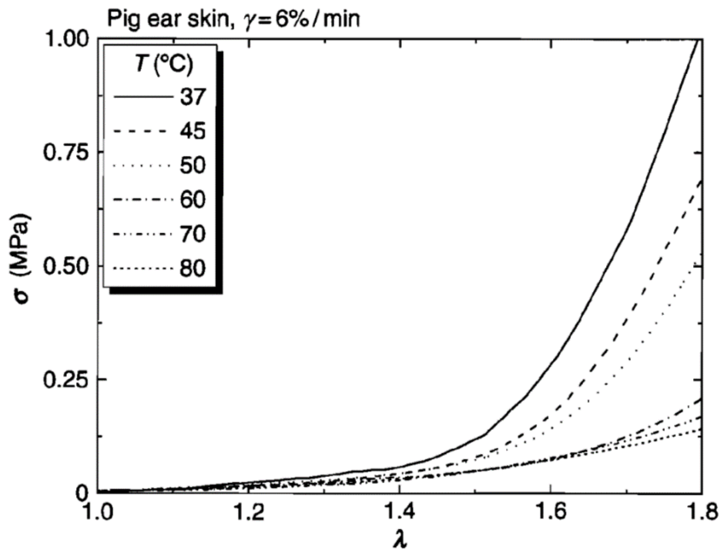


Figure 2-14: Uniaxial Tensile Behaviour of Skin Tissue Under Varying Temperatures (Xu & Lu 2009).

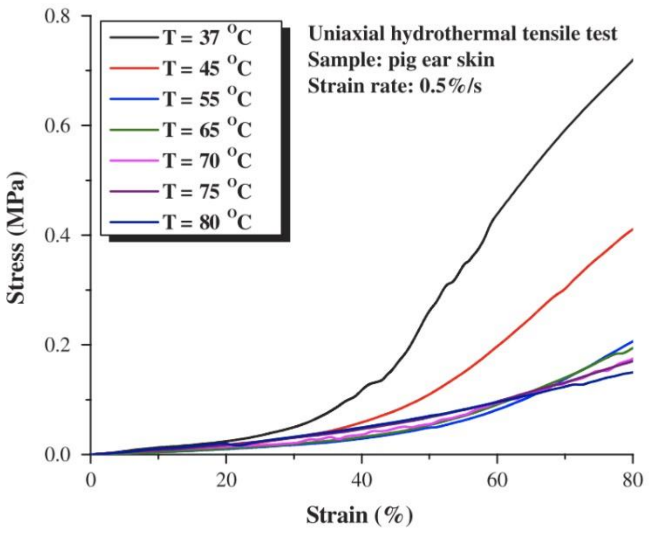


Figure 2-15: Uniaxial Hydrothermal Tensile Behaviour of Skin Tissue Under Varying Temperatures (Xu, Lu & Seffen 2008)

2.5.3 Operational Loading

During the RFA treatment procedure, the affected area of tissue can reach up to 90 °C (Watanabe et al. 2009). Typically, the applied temperature during RFA will be between 50 to 60 °C, and treatment durations will be between 3 and 12 minutes (Prakash 2010). Prakash and Diederich (2012) describe ablations lasting 5 minutes or less as short, and 15 minute ablations as long, while temperatures applied are approximately 54 °C for short ablations, and 50 °C for long ablations.

Surgical operational loads applied via grasping and compressive forces generally do not exceed 26.5 N or a compressive stress of 470 kPa (Rosen et al. 2008). Brown, J. D. et al. (2004) analysed grasping forces and durations of grasping during operations, finding the mean grasping force of 8.52 N +/- 2.77 N, for 2.29 s +/- 1.65 s duration. The maximum force applied was 68.17 N, and the average maximum grasping duration was 13.37 s +/- 11.42 s. This quantitative data was derived from testing of 31 different surgeons of varying skill.

The applied instrument temperature for the simulation developed for this research project will be in the range of 54 to 60 °C for simulating short/instantaneous point ablations, as it adds the benefit of making simulated data faster to achieve.

2.5.4 Temperature-Independent Stress-Strain Relationship

Liu and Bilston (2002) suggests that the stress-strain relationship of bovine liver tissue becomes non-linearly viscoelastic at strains greater than 0.2%, however Picinbono, Delingette and Ayache (2003) advises that "small" displacements and deformations of less than 10% of the mesh size may be considered as linearly elastic. Furthermore, Delingette and Ayache (2004) claims that displacements less than 5% of the mesh size can be considered linearly elastic. Note however, that the model presented by Picinbono, Delingette and Ayache (2003) is designed for large displacements involved in such procedures as cutting and excision.

Carter (as cited in Delingette and Ayache (2004)) indicates that the in-vivo Young's Modulus of the human liver is approximately 170 kPa. The method of measurement of this quantity is not indicated in the text however.

Carter, FJ et al. (2001) give a mean elastic modulus of 270 kPa for the right lobe of a human liver (in-vivo). For a diseased case, the mean elastic modulus was 0.74 MPa. Pig liver in the same source, has a mean elastic modulus of 4.0 MPa, which shows the degree to which in-vivo human and porcine measurements can differ. (Note that this is the simplest value, and differs by value and implementation to dynamic moduli for viscoelastic materials, which involve storage and loss components in elastic and shear moduli.) By using indentation tests, a stress-strain curve was derived by Carter, FJ et al. (2001) for human liver in-vivo; see Figure 2-16 below.

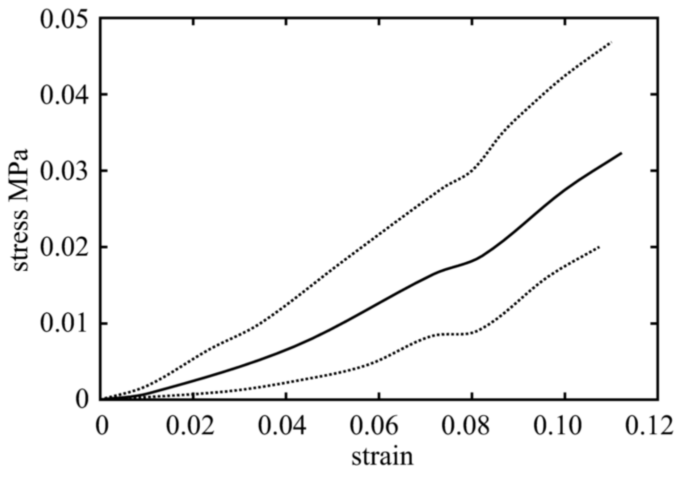


Figure 2-16: Mean Stress-Strain Curve from In-Vivo Indentation Tests of Human Liver (Carter, FJ et al. 2001)

This in-vivo stress-strain data exists only for small strains, and while it may be appropriate for utilizing as the basis for the stress-strain relationship for small thermal stresses in the simulation, no in-vivo data exists for the case of temperature-dependent stress-strain for the human liver. Since the available data for the temperature-dependent stress-strain relationship in section 3.3.3 is derived from pig ear skin (Figure 2-14), the 37 °C curve is utilized for the temperature-independent stress-strain relationship in this research project, as modelled as shown in Table 3.12 (for large strains) and Table 3.13 (for small strains; specific to tissue thermomechanical response in RFA treatment).

Refer to section 3.3.3 in the methodology chapter for a description of modelling the temperature-dependent stress-strain relationship.

2.5.5 Bioheat Transfer Models

Heat transfer in soft tissues can be treated as some combination of thermal conduction, convection via blood perfusion (blood flow through vessels and capillaries in living soft tissue), and metabolic heat production (Zolfaghari & Maerefat 2011). Bioheat transfer models can be classified into several categories: continuum models, vascular models, hybrid models and porous media models (Xu & Lu 2009). Note that hybrid models can be assumed to combine the considerations applied by continuum and vascular models.

- Continuum models consider the effect of blood vessels within tissue as an aggregate global parameter. The most popular continuum model: the Pennes bioheat transfer model, considers heat transfer due to conduction, blood perfusion through the domain, and metabolic heat generation. It regards a global effect of blood perfusion throughout the domain, rather than considering detailed contributions of the embedded vasculature. It assumes that capillary blood flow is isotropic, ignores directionality effects of blood flow, ignores vasculature geometry, and assumes that arteriole supply temperature is equivalent to core body temperature (Zolfaghari & Maerefat 2011). The Wulff continuum model regards blood temperature as being locally in thermal equilibrium with adjacent tissue due to microcirculation. The Klinger continuum model accounts for directional blood flow based on a convection field modelled on the domain's vasculature. The Chen-Holmes continuum model argues that heat transfer occurs primarily in vessels of 50 μm to 500 μm diameter. While the Chen-Holmes model considers subdivisions of tissue and blood, its requirement of knowledge of the vasculature makes it a more complex and advanced continuum model (tending towards a vascular category). Zolfaghari and Maerefat (2011) propose a simplified thermoregulatory bioheat (STB) model that can vary the core body temperature based on environmental and other factors, and accounts for the influence of thermoregulatory mechanisms such as shivering, sweating, and vasodilation/constriction.
- Vascular models consider the impact of blood vessels, focusing on each individual vessel. Zolfaghari and Maerefat (2011) discuss the hierarchy of blood vessels as they extend from the heart; blood exiting via the aorta, travelling to main supply arteries and veins, to primary and secondary arteries, to arterioles, and finally to capillaries. All of these have varying diameters, and thus thermal effects on the tissue in which they are embedded. The aorta has a diameter of 5000 μm, while a typical capillary can be as small as 5 μm in diameter. Thus, consideration of bioheat transfer due to the presence of the vascular network is a priority for some models, such as the Weinbaum-Jiji bioheat model. This model is incredibly detailed and difficult to implement, since it accounts for details such as vessel number density, size and spacing of the local vasculature. The Weinbaum-Jiji bioheat model views much of the bioheat transfer as occurring due to imperfect countercurrent heat exchange between artery-vein pairs (Zolfaghari & Maerefat 2011). In comparison, the Pennes bioheat model approximates the vasculature behaviour into a single perfusion component within its equation.
- Porous media models treat the domain as a fluid-saturated porous medium. The biological domain for a porous media bioheat model is broken down into blood vessels, cells and interstitium. Nakayama and Kuwahara (2008) consider the domain as composed of a fluid phase (blood) and a solid matrix phase (tissue, ECM), employing volume-averaging techniques and Navier-Stokes equations for the fluid phase. Physiological parameters of biological tissues including porosity and specific surface area depend on thermoregulatory mechanisms and environmental factors (Nakayama & Kuwahara 2008).

2.6 Domain Discretization

2.6.1 Defining the Domain

Xu and Lu (2009) discuss that a domain of body tissue can be represented as a hierarchical vascular network embedded in a homogeneous continuum. This description provides the possibility of utilizing either vascular or continuum (or hybrid) models for bioheat transfer. Utilization of a continuum model such as the Pennes bioheat equation to model bioheat transfer in the domain, can be applied if hierarchical vasculature is ignored.

For the sake of modelling, it is simpler to employ an isotropic, homogeneous medium, ignoring vasculature, as in Shen, Zhang and Yang (2005a). For this research project, the same rationale will be employed, supposing only that any temperature-dependent thermal parameters will vary over the domain. The domain its self will not represent the physical organ, but rather the computational domain will be assumed as a block of isotropic tissue with parameters as close to in-vivo human liver characteristics as possible, except where literature data may be insufficient, such as temperature-dependent (and independent) stress-strain relations. (See sections 2.5.4 and 3.3.3.)

The domain is modelled in 3 dimensions in this research project.

2.6.2 Mesh Element Selection

Multiple types of basic elements exist for discretizing domains of different dimensions. Kattan (2008) discusses the following element types in great detail:

Table 2.1: Finite Element Types

Element Name	**Dimensions**
Spring Element	1D
Bar Element	1D
Plane Truss Element	2D
Beam Element	2D
Plane Frame Element	2D
Grid Element	2D
Triangular Element	2D
Quadrilateral Element	2D
Space Truss Element	3D
Space Frame Element	3D
Tetrahedral Element	3D
Hexahedral (Brick) Element	3D

A great deal more varieties exist, such as prismatic and axisymmetric element varieties. These are developed and employed for specific applications; axisymmetric elements applied to modelling pressure vessels for example.

Interpolations between nodes in finite elements can vary. The Finite Element Method utilizes shape functions (a.k.a. “interpolation functions” or “basis functions”) for these interpolations. Shape functions are polynomials, commonly linear or quadratic, although further orders can be implemented when appropriate (cubic, etc). Quadratic interpolations require the addition of an extra node along each edge of the element (the edge midpoint), in order to enable quadratic interpolation (three coefficients in the quadratic equation and three nodes where the field value is determined); see Figure 2-17; (Roland W. Lewis, Perumal Nithiarasu & Seetharamu 2004).

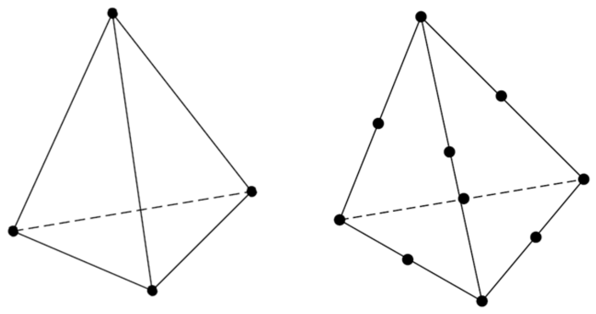


Figure 2-17: Linear and Quadratic Tetrahedral Elements

The addition of extra nodes to each element results in greater accuracy for values interpolated within the element, however drastically increases the amount of computation required, since the number of unknowns to be solved for / DoF, greatly increases.

Delingette and Ayache (2004) discuss that the 4-node linear tetrahedral element has the disadvantage of poor convergence in comparison to the linear hexahedral element. Furthermore, like the aptly-named 2-dimensional constant-strain triangular element (CST) (i.e. the linear triangular element), the 3D linear tetrahedron provides only linear displacement functions, thus the derived strain is constant throughout the element. Cook (1995) advises that such constant-strain elements give good results in regions of the domain where the strain gradient is small, otherwise, it is advised that a mesh of constant-strain elements is repeatedly refined in order to correctly represent sharp strain gradients. However, the loss of accuracy in deformation computations utilizing linear tetrahedral elements are still small in comparison to the "large uncertainty" regarding physical tissue parameter values (Delingette & Ayache 2004).

Delingette and Ayache (2004) state that the rationale for employing linear tetrahedra for 3-dimensional cases (likewise linear triangles for 2D cases), is that geometrically, meshing of irregular domains is much simpler with tetrahedra than with hexahedra. Since anatomical structures are generally irregularly shaped, the tetrahedral element is favoured for meshing a 3D domain.

The liver is curved and irregular, and the linear tetrahedral element; highly recommended in the literature, is selected for meshing in this project. Despite the fact that regular domains (consisting of linear tetrahedra) are to be generated for analysis, the meshing of the domain by utilizing linear tetrahedra could also be applied to irregular meshes without modification of the simulation code.

One further justification for the selection of linear tetrahedra, is not only their simplicity (simplest 3D element) and their low computation cost compared to other 3D element types, but the means by which the graphics hardware can be utilized. Referring to sections 2.3.5 and 3.1, the 4-component variables and 4x4 matrix variable types in HLSL can be exploited optimally, since each linear tetrahedron contains 4 nodes, each node has a single thermal DoF, and thus linear tetrahedral element matrices are 4x4 in size. Thus element matrix calculations can exploit HLSL built-in types and their associated optimized functions.

Thus the geometry of the selected linear tetrahedral element is beneficial for multiple reasons.

2.7 Equations & Relationships

2.7.1 Bioheat Transfer

The most commonly applied bioheat model (heat transfer in living soft tissue) is a continuum model; the Pennes bioheat model developed by Harry Pennes in 1948, to mathematically express tissue heat transfer due to blood perfusion and metabolic heat generation. It serves as a basis for many other models which build upon the Pennes model by adding corrections and extra terms to increase accuracy or suitability for their applications (Zolfaghari & Maerefat 2011). The Pennes model may be written as:

$$\rho_{ti}C_{ti}\frac{\partial T_{ti}}{\partial t} = \nabla \cdot k_{ti}\nabla T_{ti} + W_{bl}C_{bl}(T_{art} - T_{ti}) + Q_m + Q_r(x, y, z, t) \tag{2.1}$$

where ρ_{ti}, C_{ti}, T_{ti}, t, k_{ti}, W_{bl}, C_{bl}, T_{art}, Q_m, Q_r are respectively tissue density, tissue specific heat capacity, tissue temperature, time, tissue thermal conductivity, blood perfusion rate, blood specific heat capacity, arterial blood temperature, metabolic heat generation rate and heat rate from regional heat sources (Shen, Zhang & Yang 2005a). The blood perfusion rate can be alternatively described as the mass flow rate of blood per unit volume of tissue (Gupta, Singh & Rai 2010).

Table 2.2: Pennes Bioheat Equation Nomenclature

Symbol	Description	Units
ρ_{ti}	Tissue density	$\frac{kg}{m^3}$
C_{ti}	Tissue specific heat capacity	$\frac{J}{kg \cdot {}^\circ K}$
T_{ti}	Tissue temperature	$^\circ K$
t	Time	s
k_{ti}	Tissue thermal conductivity	$\frac{W}{m \cdot {}^\circ K}$
W_{bl}	Blood perfusion rate	$\frac{kg}{m^3 \cdot s}$
C_{bl}	Blood specific heat capacity	$\frac{J}{kg \cdot {}^\circ K}$
T_{art}	Arterial blood temperature	$^\circ K$
Q_m	Metabolic heat generation rate	$\frac{W}{m^3}$
Q_r	Regional source heat generation rate	$\frac{W}{m^3}$

Thus each term in the Pennes bioheat equation is in units of $\frac{kg}{m \cdot s^3}$ (SI Base Units); rather: Watts per cubic metre ($\frac{W}{m^3}$).

2.7.2 Thermal Damage

Thermal denaturation of soft tissue has already been mentioned in section 2.5.2, as the breakdown of collagen's intramolecular crosslinks, causing collagen to lose its loadbearing ability, having changed structure towards a gel-like state. Xu and Lu (2009) discuss the means of quantifying this irreversible tissue damage by using the Arrhenius burn integration, shown in Equation (2.2).

$$\Omega = \int_0^t A e^{-E_a/RT} dt \tag{2.2}$$

where Ω is the dimensionless measure of thermal damage, A is a material parameter that equates to a frequency factor, E_a is the activation energy, R is the universal gas constant $\left(R = 8.314 \frac{J}{mol°K}\right)$, and T is the tissue temperature (in Kelvin; where 0 Kelvin is equivalent to 273.15 °C) which can also be a function of time t (s) (Prakash & Diederich 2012; Xu & Lu 2009). If T is employed as a function of time t, Equation (2.2) integrates to a form involving an incomplete gamma function.

The activation energy E_a and frequency factor A must be determined experimentally. Fortunately, the literature documents the experimentally determined quantities for several tissues, shown in Table 2.3.

Table 2.3: Activation Energy and Frequency Factors for Tissues

Tissue	**Activation Energy E_a $\times 10^5 \frac{J}{mol}$**	**Frequency Factor A s^{-1}**	**Source**
Liver	3.513	3.18×10^{55}	(Prakash & Diederich 2012)
Prostate	5.064	2.984×10^{80}	(Prakash & Diederich 2012)
Back skin (pig)	5.255	2.126×10^{81}	(Xu & Lu 2009)
Belly skin (pig)	3.935	1.151×10^{61}	(Xu & Lu 2009)
Ear skin (pig)	5.867	5.240×10^{91}	(Xu & Lu 2009)
Face skin (pig)	4.710	4.575×10^{72}	(Xu & Lu 2009)
Flank skin (pig)	4.012	1.501×10^{61}	(Xu & Lu 2009)

If the Arrhenius burn integration is directly applied at any temperature, tissue would sustain thermal damage at regular body temperature; which obviously does not occur in real life. Thermal damage is therefore assumed to take place at tissue temperatures greater than 43 °C in human tissue (Prakash 2010; Xu & Lu 2009).

To quantify the calculated thermal damage, it is compared to a threshold quantity; $\Omega = 1.0$ which defines the commencement of irreversible damage to the tissue (Xu, Lu & Seffen 2008). When $\Omega =$ 4.6, coagulative necrosis has occurred (Prakash & Diederich 2012). Time t to cellular necrosis may be defined as the duration of applied heating (maintaining constant temperature) to reach a thermal damage of $\Omega = 4.6$. This is applied in section 4.1.2.

Xu, Lu and Seffen (2008) explain that the process of thermal denaturation is considerably slow at a tissue temperature of 45 °C, however at 55 °C, thermal denaturation of collagen in skin occurs almost instantly. The degree of thermal denaturation (the fraction of denatured collagen) is quantified in Xu, Lu and Seffen (2008) and Xu and Lu (2009) as:

$$Deg(t) = 1 - e^{-\Omega(t)}. \tag{2.3}$$

The rapid process of collagen denaturation at temperatures of 55 °C and greater, can be visualized in Figure 2-18.

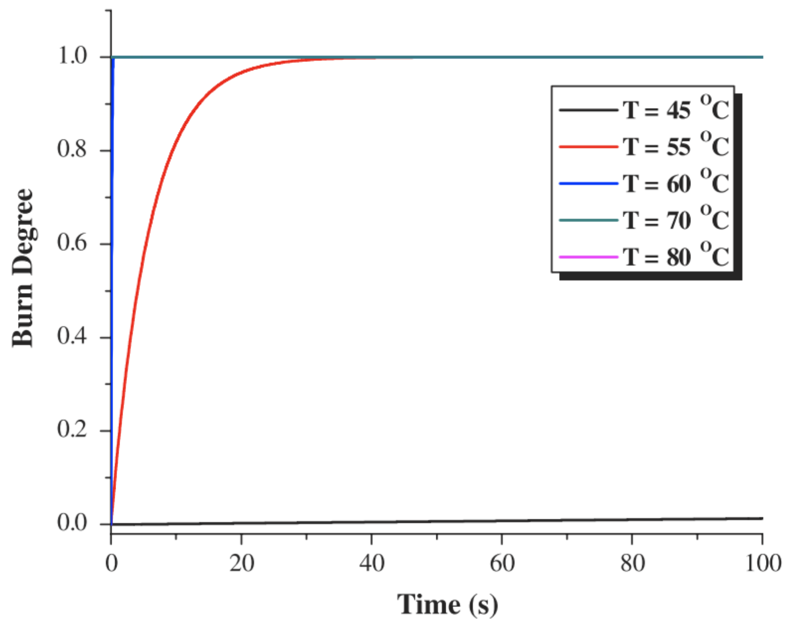


Figure 2-18: Burn Degree vs Time for Different Heating Temperatures (Xu, Lu & Seffen 2008)

See section 2.5.3 for expected applied temperatures during the typical RFA procedure.

2.8 Equation Discretization

2.8.1 Spatial Discretization

The Galerkin Method is a Weighted Residual Method commonly employed for discretizing partial differential equations (PDEs) over a finite domain. When compared to other Weighted Residual Methods such as Collocation, Sub-domain and Least Squares methods, the Galerkin method is the most accurate method (see Figure 2-19), as concluded by Roland W. Lewis, Perumal Nithiarasu and Seetharamu (2004) (by comparing the Weighted Residual Methods to each other and the exact solution for a simple fin case).

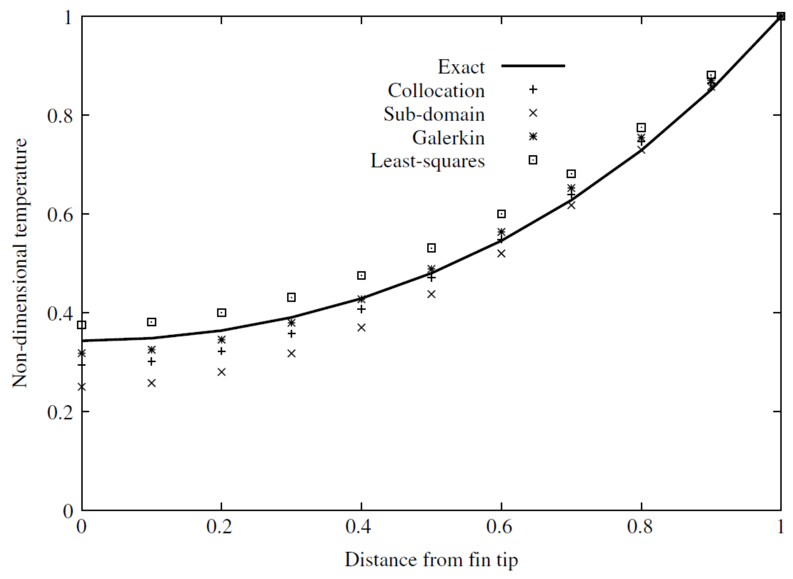


Figure 2-19: Weighted Residual Methods Comparison (Roland W. Lewis, Perumal Nithiarasu & Seetharamu 2004)

Weighted residual methods multiply the PDE with a weighting function and take the average residual over the domain (finite number of precise locations; i.e. nodes). The residual is minimized when integrating over the domain by specifying that the integral be equal to 0.

The weighting function varies for each method. The Galerkin Weighted Residual Method utilizes the shape (basis) functions (defined by element type selection in the Finite Element Method) as the weighting functions. Thus:

$$\int_{\Omega} N_i(x) R dx = 0 \tag{2.4}$$

where Ω is the domain, N_i the weighting function (shape function), and R the residual.

The Galerkin Method is used in literature for discretization of PDEs; i.e. the spatial component of the Pennes bioheat equation, over the chosen domain (Gupta, Singh & Rai 2010; Watanabe et al. 2009).

The Galerkin Weighted Residual Method is the technique selected for discretizing the spatial component of the Pennes bioheat equation in this research project

2.8.2 Temporal Discretization

The temporal component of the Pennes bioheat equation can be numerically approximated by using the Finite Difference Method. By using a Taylor series, the temperature at the *n* + 1th level can be expressed as:

$$T^{n+1} = T^n + \Delta t \frac{\partial T^n}{\partial t} + \frac{\Delta t^2}{2} \frac{\partial^2 T^n}{\partial t^2} + \cdots \tag{2.5}$$

By truncating the series from the second order terms onward:

$$\frac{\partial T^n}{\partial t} \approx \frac{T^{n+1} - T^n}{\Delta t} + O(\Delta t). \tag{2.6}$$

This is applied to the temporal component of the Pennes bioheat equation (refer to the methodology section) in order to sequentially step forward some small finite amount of time Δt, to evaluate the temperature distribution over the domain.

The temporal discretization scheme can be explicit (forward difference method; computes only with past temperatures), implicit (backward difference method; computes with past and current temperatures) or semi-implicit (Crank-Nicolson method; computes with past and current temperatures) (Roland W. Lewis, Perumal Nithiarasu & Seetharamu 2004).

According to Courtecuisse et al. (2010), an explicit scheme is particularly appropriate for real-time applications (since implicit methods produce a matrix system requiring expensive inversion).

The explicit scheme (forward difference) of the Finite Difference Method is the technique selected for discretizing the temporal component of the Pennes bioheat equation in this research project.

2.9 Equation / Simulation Parameters

2.9.1 Tissue Initial Temperature

The liver tissue initial temperature (or reference temperature) is tabulated below from a selection of literature sources.

Table 2.4: Literature Comparison of Liver Tissue Initial Temperature

Source	**Quantity**	**Units**	**Comments**
(Watanabe et al. 2009)	-	-	Not specified
(Shen, Zhang & Yang 2005a)	37	℃	-
(Hasgall PA et al. 2013)	-	-	Not specified
(Gupta, Singh & Rai 2010)	37	℃	-
(Peng, O'Neill & Payne 2011)	37	℃	-
(Xu, J et al. 2010)	37	℃	-

By examining the literature, it is obvious that the initial temperature of the liver tissue is consistently equivalent to normal body temperature; 37 °C. Furthermore, the tissue boundary can be set to 37 °C and maintained as such (Dirichlet boundary condition) during simulation (Peng, O'Neill & Payne 2011; Prakash & Diederich 2012; Watanabe et al. 2009; Xu, J et al. 2010).

Therefore, the initial temperature, and boundary conditions (where applied) for this research project will be 37 °C.

2.9.2 Temperature-Independent Tissue Density

The temperature-independent liver tissue density is tabulated below from a selection of literature sources.

Table 2.5: Literature Comparison of Liver Tissue Temperature-Independent Density

Source	Quantity	Units	Comments
(Watanabe et al. 2009)	1060	$\frac{kg}{m^3}$	Hog liver
(Shen, Zhang & Yang 2005a)	1000	$\frac{kg}{m^3}$	-
(Hasgall PA et al. 2013)	1079	$\frac{kg}{m^3}$	Human liver
(Gupta, Singh & Rai 2010)	1000	$\frac{kg}{m^3}$	-
(Peng, O'Neill & Payne 2011)	1060	$\frac{kg}{m^3}$	-
(Xu, J et al. 2010)	1030	$\frac{kg}{m^3}$	Porcine liver

By examining the literature, it appears that liver tissue density is typically approximated as slightly denser than water (1000 $\frac{kg}{m^3}$). The density of human liver is slightly greater than that of porcine varieties, judging by Hasgall PA et al. (2013).

Since the tissue density of the human liver is available, and does not greatly deviate from data provided by other literature sources, the temperature-independent liver tissue density ρ_{ti} for this research project will be 1079 $\frac{kg}{m^3}$.

2.9.3 Temperature-Independent Tissue Specific Heat Capacity

The temperature-independent liver tissue specific heat capacity is tabulated below from a selection of literature sources.

Table 2.6: Literature Comparison of Liver Tissue Temperature-Independent Specific Heat Capacity

Source	**Quantity**	**Units**	**Comments**
(Watanabe et al. 2009)	3600	$\frac{J}{kg°K}$	Hog liver
(Shen, Zhang & Yang 2005a)	4000	$\frac{J}{kg°K}$	-
(Hasgall PA et al. 2013)	3540	$\frac{J}{kg°C}$	Human liver
(Gupta, Singh & Rai 2010)	4180	$\frac{J}{kg°K}$	-
(Peng, O'Neill & Payne 2011)	3600	$\frac{J}{kg°K}$	-
(Xu, J et al. 2010)	3600	$\frac{J}{kg°K}$	Porcine liver

Similarly to the liver tissue density, the liver tissue specific heat capacity does not greatly vary from the values provided for porcine varieties. The two quantities ($\geq 4000 \frac{J}{kg°K}$) are inconsistent with other quantities derived from the literature.

It seems appropriate then, since the quantity for the human liver does not greatly deviate from the general trend in literature, that the temperature-independent liver tissue specific heat capacity C_{ti} for this research project will be 3540 $\frac{J}{kg°K}$.

Refer to section 2.9.4 to see that initial tissue temperature-independent specific heat capacity can be calculated as 3405 $\frac{J}{kg°K}$ at 37 °C using linear interpolation from the data in Table 2.7 and Equation (2.7). This value is necessary for the comparison of a temperature-independent and temperature-dependent analysis. For all other analyses, 3540 $\frac{J}{kg°K}$ will be used.

2.9.4 Temperature-Dependent Tissue Specific Heat Capacity

When referring to Xu and Lu (2009) and Watanabe et al. (2009) in Figure 2-9 and Figure 2-10 respectively, the quantitative data for temperature-dependent specific heat capacity can be observed for pig skin tissue and hog liver tissue. Since the application for this research project is directed at RFA in the liver, the most appropriate data to consider is that of Watanabe et al. (2009).

Table 2.7 displays the specific heat capacity variant with temperature for the data presented in Figure 2-10. The tabulated data shown is taken from points where the gradient changes significantly.

Table 2.7: Temperature-Dependent Specific Heat Capacity

Temperature T (°C)	**Tissue Specific Heat Capacity C_{ti} ($\frac{J}{kg°K}$)**
10	3325
49	3440
62	3610
67	3770
73	3700
75	3630
79	3650
90	3650

Since designing a polynomial function to model this data (using regression analysis) would result in a high-order polynomial to retain the shape of the denaturation peak (and thus many coefficients) the most practical way to approach modelling the non-linear relationship with respect to programmatic implementation is to utilize linear interpolation between points in the data set.

Thus, to obtain the temperature-dependent tissue specific heat capacity C_{ti} for a given temperature T, the temperature is first clamped to the range. If it is less than 10 °C, it is set to 10 °C. If it is greater than 90 °C, it is set to 90 °C. By comparing the temperature to the tabulated data, it can then be determined upon which linear interpolant the temperature resides, and the data points (T_1, C_{ti1}), (T_2, C_{ti2}) bounding it. Linear interpolation can then be performed as shown in Equation (2.7).

$$C_{ti} = \left(\frac{C_{ti2} - C_{ti1}}{T_2 - T_1}\right) \cdot (T - T_1) + C_{ti1} \tag{2.7}$$

This method will be applied in this research project for calculating the temperature-dependent liver tissue specific heat capacity.

2.9.5 Temperature-Independent Tissue Thermal Conductivity

The temperature-independent liver tissue thermal conductivity is tabulated below from a selection of literature sources.

Table 2.8: Literature Comparison of Liver Tissue Temperature-Independent Thermal Conductivity

Source	**Quantity**	**Units**	**Comments**
(Watanabe et al. 2009)	0.502	$\frac{W}{m°K}$	Hog liver
(Shen, Zhang & Yang 2005a)	0.5	$\frac{W}{m°K}$	-
(Hasgall PA et al. 2013)	0.52	$\frac{W}{m℃}$	Human liver
(Gupta, Singh & Rai 2010)	0.5	$\frac{W}{m°K}$	-
(Peng, O'Neill & Payne 2011)	0.49	$\frac{WJ}{m°K}$	Assumed typo. Units of $\frac{W}{m°K}$ probably intended.
(Xu, J et al. 2010)	0.497	$\frac{W}{m°K}$	Porcine liver

By examining the literature, the typical quantity for tissue thermal conductivity lies around 0.5 $\frac{W}{m°K}$, and the value for the human liver differs by about 4% from that amount.

This difference is still small, and the value can still be considered appropriate, thus the temperature-independent liver tissue thermal conductivity k_{ti} for this research project will be 0.52 $\frac{W}{m°K}$.

Refer to section 2.9.6 to see that initial tissue temperature-independent thermal conductivity can be calculated as 0.4604 $\frac{W}{m°K}$ at 37 °C using Equation (2.8). This value is necessary for the comparison of a temperature-independent and temperature-dependent analysis. For all other analyses, 0.52 $\frac{W}{m°K}$ will be used.

2.9.6 Temperature-Dependent Tissue Thermal Conductivity

Watanabe et al. (2009) model the temperature dependence of hog liver tissue thermal conductivity (see Figure 2-11) using a linear function.

$$k_{ti} = aT + b \tag{2.8}$$

where $a = 9.2 \times 10^{-3}\ \frac{W}{m°K^2}$, and $b = 1.2 \times 10^{-1}\ \frac{W}{m°K}$.

Note that the available range of data for temperature-dependent tissue specific heat capacity ranges from 10 °C to 90 °C, thus if the supplied temperature lies outside of these bounds, it is clamped to the range. If the temperature is to be clamped to this range for temperature-dependent tissue specific heat capacity, then it should also be clamped to the same range for temperature-dependent tissue thermal conductivity, as discrepancies may be introduced if the specific heat capacity is clamped and the thermal conductivity remains unclamped. C_{ti} would become somewhat temperature-independent (each end of the range) for temperatures outside of the data range, while unclamped temperature would cause k_{ti} to remain temperature-dependent, and exploit the linear function to produce data outside the range of the quantitative data in Watanabe et al. (2009).

2.9.7 Temperature-Independent Metabolic Heat Generation Rate

The temperature-independent liver metabolic heat generation rate is tabulated below from a selection of literature sources.

Table 2.9: Literature Comparison of Liver Tissue Temperature-Independent Metabolic Heat Generation Rate

Source	**Quantity**	**Units**	**Comments**
(Watanabe et al. 2009)	-	-	Not specified
(Shen, Zhang & Yang 2005a)	33800	$\frac{W}{m^3}$	-
(Hasgall PA et al. 2013)	10.41	$\frac{W}{kg}$	Human liver. See below calculations for validating this value in units of $\frac{W}{m^3}$.
(Gupta, Singh & Rai 2010)	-	-	Not specified
(Peng, O'Neill & Payne 2011)	-	-	Not specified
(Xu, J et al. 2010)	-	-	Not specified

Metabolic heat generation rate in the liver is given by Hasgall PA et al. (2013) as 10.41 W/kg whereas the useful unit in the Pennes bioheat equation is W/m^3. In order to convert this value to the desired units, it is multiplied by the liver tissue density.

$$10.41 \times 1079 = 11232 \ \frac{W}{m^3} \qquad (2.9)$$

Deshan et al. (2007) consider the tissue metabolic heat generation rate to be insignificant with respect to heat generated by artificial sources in ablative operational procedures, and disregards it in calculation. While that may slightly simplify calculations, Gupta, Singh and Rai (2010) tends towards the opposite direction, modelling metabolic heat generation as temperature-dependent.

$$Q_m = Q_{m0}[1 + 0.1(T - T'_0)] \qquad (2.10)$$

where Q_{m0} is the basal metabolic heat generation rate (1091 $\frac{W}{m^3}$), and T'_0 is the reference temperature (37 °C).

Since the intention is to involve metabolic heat generation in the simulation without adding unnecessary complexity, and since the quantities provided by Shen, Zhang and Yang (2005a) typically deviate from the quantities provided in other literature, the most reasonable choice seems to be select the temperature-independent metabolic heat generation rate Q_m for this research project as the calculated value of 11232 $\frac{W}{m^3}$.

2.9.8 Temperature-Independent Blood Specific Heat Capacity

The temperature-independent blood specific heat capacity is tabulated below from a selection of literature sources.

Table 2.10: Literature Comparison of Temperature-Independent Blood Specific Heat Capacity

Source	**Quantity**	**Units**	**Comments**
(Watanabe et al. 2009)	4180	$\frac{J}{kg°K}$	Hog
(Shen, Zhang & Yang 2005a)	-	-	Not specified
(Hasgall PA et al. 2013)	3617	$\frac{J}{kg℃}$	Human blood
(Gupta, Singh & Rai 2010)	3344	$\frac{J}{kg°K}$	-
(Peng, O'Neill & Payne 2011)	4180	$\frac{J}{kg°K}$	-
(Xu, J et al. 2010)	3500	$\frac{J}{kg°K}$	Porcine

Similarly to tissue specific heat capacity, it seems that two sources are inconsistent with the remaining literature, however are consistent with each other (same quantity). While it seems that blood specific heat capacity can vary greatly for porcine liver varieties in the literature, the quantity defined by Hasgall PA et al. (2013) for blood specific heat capacity is within this range.

The temperature-independent blood specific heat capacity C_{bl} for this research project will be 3617 $\frac{J}{kg°K}$.

2.9.9 Temperature-Independent Blood Perfusion Rate

The temperature-independent blood perfusion rate is tabulated below from a selection of literature sources.

Table 2.11: Literature Comparison of Temperature-Independent Blood Perfusion Rate

Source	Quantity	Units	Comments
(Watanabe et al. 2009)	-	-	Not specified
(Shen, Zhang & Yang 2005a)	0.5	$\frac{kg}{m^3s}$	Inconsistent with quantities from other literature.
(Hasgall PA et al. 2013)	902	$\frac{ml}{min \cdot kg}$	Human blood. See below calculations for validating this value in units of $\frac{kg}{m^3s}$.
(Gupta, Singh & Rai 2010)	8	$\frac{kg}{m^3s}$	-
(Peng, O'Neill & Payne 2011)	-	-	Not specified
(Xu, J et al. 2010)	1.67×10^{-5}	$\frac{m^3}{kg \cdot s}$	Porcine Consistent with validation of Hasgall et al.'s quantity.

Blood perfusion rate in the liver is given by the database as 902 ml/(min•kg), whereas the useful unit in the Pennes bioheat equation is kg/(m^{3}s), since all terms of the expanded equation should result in units of W/m^3. Other resources contain different bioheat transfer equations, different/modified forms of the Pennes bioheat equation (Gupta, Singh & Rai 2010), give blood perfusion rate in different units such as m^3/m^{3}s (Zolfaghari & Maerefat 2011), or provide altogether different values implying the use of different tissues for their models (Shen, Zhang & Yang 2005a). Thus the following calculation is performed on the value obtained from Hasgall PA et al. (2013), to obtain blood perfusion rate in kg/(m^{3}s).

$$902\ \frac{ml}{min \cdot kg} = 902 \times \frac{1}{10^6} \times \frac{1}{60}\ \frac{m^3}{kg \cdot s} \tag{2.11}$$

The unit can then be converted by multiplying the quantity by the tissue density squared.

$$\frac{902}{6.0 \times 10^7} \times 1079^2 = 17.502\ \frac{kg}{m^3s} \tag{2.12}$$

In order to determine whether this value makes any sense, some validation must be made. With a mass of between 1.0 kg and 1.5 kg in an adult human and liver tissue density of 1079 kg/m^3, the volume of the liver may be calculated to range from 9.268×10^{-4} m^3 and 1.390×10^{-3} m^3. Multiplying the blood perfusion rate by volumes in this range, it can be determined that the liver is perfused with a blood mass flow rate of between 0.016 and 0.024 kg per second.

This can be validated by other data that states that the liver receives approximately 25% of cardiac output (Eipel 2010). Defining cardiac output (mass flow rate output from the heart) is simple. It is typical knowledge that the average adult heart outputs approximately five litres of blood per minute.

$$5\frac{L}{min} = 5 \times \frac{1}{10^3} \times \frac{1}{60}\ \frac{m^3}{s} \tag{2.13}$$

Multiplying this volume flow rate by blood density (1050 kg/m^3; (Hasgall PA et al. 2013), the mass flow rate of blood output from the heart may be found.

$$\frac{5}{6.0 \times 10^4} \times 1050 = 0.0875\ \frac{kg}{s} \tag{2.14}$$

Finally, 25% of this output (the mass flow rate of blood to the liver per second) is approximately 0.022 kg/s; well within the calculated range of 0.016 and 0.024 kg/s as calculated using the liver blood perfusion rate of 17.502 kg/(m^3s). Thus the blood perfusion rate value is validated.

Notice that while the blood perfusion rate quantity given by Shen, Zhang and Yang (2005a) is inconsistent with those of other literature sources, the remaining quantities when converted to the appropriate units, are of the same order. The calculated blood perfusion rate of 17.502 $\frac{kg}{m^3 s}$ derived from the quantity given by Hasgall PA et al. (2013) is very close to that obtained by performing similar calculations (not shown) from the quantity given by Xu, J et al. (2010).

Therefore, the temperature-independent blood perfusion rate W_{bl} for this research project will be 17.502 $\frac{kg}{m^3 s}$.

2.9.10 Arterial Temperature

The liver tissue blood arterial temperature is tabulated below from a selection of literature sources.

Table 2.12: Literature Comparison of Liver Tissue Blood Arterial Temperature

Source	**Quantity**	**Units**	**Comments**
(Watanabe et al. 2009)	-	-	Not specified
(Shen, Zhang & Yang 2005a)	37	℃	-
(Hasgall PA et al. 2013)	-	-	Not specified
(Gupta, Singh & Rai 2010)	37	℃	-
(Peng, O'Neill & Payne 2011)	37	℃	-
(Xu, J et al. 2010)	37	℃	-

By examining the literature, it is obvious that the arterial blood temperature in the liver is consistently equivalent to normal body temperature; 37 °C.

Therefore, the arterial blood temperature T_{art} for this research project will be 37 °C.

2.9.11 Coefficient of Thermal Expansion

While the simulation software developed in this research project is intended for producing real-time equivalent bioheat transfer and thermal damage outcomes, the coefficient of thermal expansion for liver tissue is useful in third-party software when involved in calculating thermal strains, and thus, thermal stresses, from the simulated temperature field.

The coefficient of thermal expansion (represented by α) is defined in literature as $\alpha = \frac{1\times10^{-4}}{°\mathrm{C}}$ for skin tissue (Xu, Lu & Seffen 2008; Xu, Seffen & Lu 2008a).

It is defined as the same magnitude for liver tissue in Keangin, Wessapan and Rattanadecho (2011).

Therefore the specified quantity for the coefficient of thermal expansion of liver tissue will be used for this research project.

Shen, Zhang and Yang (2005b) and Shen, Zhang and Yang (2005a) propose a model by which to calculate thermal stresses and thermal strains as well as mechanical stresses and strains due to deformation; combining the two facets into a single equation based on porous media behaviour.

$$\begin{bmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{zz}\\ \tau_{xy}\\ \tau_{xz}\\ \tau_{yz}\end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & 0 & 0 & 0\\ c_{21} & c_{22} & c_{23} & 0 & 0 & 0\\ c_{31} & c_{32} & c_{33} & 0 & 0 & 0\\ 0 & 0 & 0 & c_{44} & 0 & 0\\ 0 & 0 & 0 & 0 & c_{55} & 0\\ 0 & 0 & 0 & 0 & 0 & c_{66}\end{bmatrix}\begin{bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{zz}\\ \gamma_{xy}\\ \gamma_{xz}\\ \gamma_{yz}\end{bmatrix} - \begin{bmatrix}\beta\\ \beta\\ \beta\\ 0\\ 0\\ 0\end{bmatrix}[\theta] \tag{2.15}$$

Where σ are normal stresses, ε are normal strains, τ are shear stresses, γ are shear strains, θ is the temperature difference (ΔT), and the coefficients c are:

$$c_{11} = c_{22} = c_{33} = \frac{1-\nu}{(1-2\nu)(1+\nu)} \tag{2.16}$$

$$c_{12} = c_{21} = c_{13} = c_{31} = c_{23} = c_{32} = \frac{\nu}{(1-2\nu)(1+\nu)} \tag{2.17}$$

$$c_{44} = c_{55} = c_{66} = \frac{\nu}{2(1+\nu)} \tag{2.18}$$

and:

$$\beta = \frac{E\alpha}{1-2\nu} \tag{2.19}$$

where E is the Young's modulus, ν is the Poisson's Ratio, and α is the coefficient of thermal expansion.

Note the right-hand-side terms of Equation (2.15). The first of the two obtains the displacement/deformation stress component, whilst the much simpler right-most term yields the thermal stress, and can be expanded by referencing Equation (2.19) as:

$$\begin{bmatrix}\frac{E\alpha}{1-2\nu}\\ \frac{E\alpha}{1-2\nu}\\ \frac{E\alpha}{1-2\nu}\\ 0\\ 0\\ 0\end{bmatrix}[\Delta T]. \tag{2.20}$$

Here, we see that thermal stresses are accounted for along the three primary axes (in 3D Cartesian space) since the shear components equate to zero, and furthermore that the use of the Young's modulus implies that linear elasticity is assumed. The relationship with the Poisson's Ratio can be seen further in (Boresi, Chong & Lee 2010) (pp. 285-7).

The fact that the thermal stresses are calculated by using the Young's modulus of elasticity demonstrates that the model is unsuitable for this application (unless further modifications are made). Not only are mechanical deformation stresses not sought in the scope of this project (only thermal stresses & strains), but non-linearly elastic stress-strain is modelled based on section 2.5.2 and section 3.3.3 literature and derived functions, thus making elastic modulus based calculations redundant (or only useful to serve as comparison; see section 4.4.3). The relationship implied between thermal stress and thermal strain by Equation (2.15) is a linear one, while it is obvious when examining section 2.5.2, that thermal stress-strain is not only non-linearly elastic, but also temperature dependent.

It is recommended that this model be examined and modified for further research purposes however, to build upon the work accomplished within the scope of this research project.

2.9.12 Poisson's Ratio

Similarly to the coefficient of thermal expansion, the Poisson's Ratio is to be employed in calculating thermal strains, and thus, thermal stresses, from the simulated temperature field using third-party finite element software. The liver tissue Poisson's Ratio is tabulated below from a selection of literature resources.

Table 2.13: Literature Comparison of Liver Tissue Poisson's Ratio

Source	**Quantity**	**Units**	**Comments**
(Carter, TJ et al. 2005)	0.45	-	-
(Manescu et al. 2012)	0.45	-	Liver
(Shen, Zhang & Yang 2005a)	0.31	-	Similarly to many parameters from this source, the quantity deviates from the majority of literature.
(Xu, Lu & Seffen 2008)	0.48	-	Skin
(Xu, Seffen & Lu 2008a)	0.48	-	Skin

The Poisson's Ratio of liver tissue tends in literature to be $\nu = 0.45$, and thus this quantity will be used for this research project.

2.10 Real-Time Simulation

Delingette and Ayache (2004) present three basic rules regarding constraints to facilitate real-time simulation of deformations:

- Rule 1: Visual feedback should be within the range of 20-60 Hz, while haptic feedback should be in the range of 300-1000 Hz; also agreed upon by Courtecuisse et al. (2010). 20 Hz display frequency is sufficient for displaying slowly moving objects, and a 300 Hz frequency for haptic rendering of very soft objects.
- Rule 2: Minimum latency between hardware components and instruments. The delay between user input and simulation output may be composed of the time taken for communications between the haptic instrument and the computer, and that of the computer to the display and to the haptic output. To improve realism, the latency must be reduced as far as possible to minimize interactive lag.
- Rule 3: Realistic motion of soft tissue should account for relaxation of the soft tissue after perturbation. The visco-elastic mechanical behaviour of the tissue is accounted for in this way.

Delingette and Ayache (2004) imply by discussion of common surgical simulator architecture, that 25 Hz is the approximate real-time visual frequency update constraint, and 500Hz the approximate for haptic rendering frequency.

In fact, 25 Hz is the frequency of Phase-Alternating Line (PAL) encoding for analogue television broadcasting, and thus is a widely accepted standard visual framerate throughout much of Asia, Australia and Europe. The United States maintains a different broadcast standard of 30 Hz.

Frank et al. (2001) claim that a frequency of 30 Hz is the minimum visual feedback frequency for real-time simulation, and that 500 Hz is the standard for haptic interaction. Kup-Sze, Hanqiu and Pheng-Ann (2003) claim that 30 Hz and 1 kHz respectively are the standard for visual and haptic rendering frequencies in real-time applications. Nefti and Abulgasem (2009) claim that 20 Hz is sufficient visual update frequency, and 1 kHz is required for haptic updates, sufficient for smooth user interaction.

For a real-time bio-thermomechanical simulation then, if haptic rendering is not a component of the simulation, then the visual display frequency will be the deciding factor of real-time for the simulation. Since 25 Hz is an accepted standard for real-time visual display by Delingette and Ayache (2004) (and within the general range supported by other literature), the simulation should be updated within 40 milliseconds each iteration. If the simulation time step matches the update period, the simulation can proceed at real-time. Should the simulation be able to calculate each iteration in less than the required real-time update period, the difference between the required time and the actual calculation time implies that the computer is waiting (thread sleeping), so as to keep a consistent update schedule.

3 Methodology

3.1 Domain Discretization

Referring to section 2.6.2, the selected element type for discretizing the domain for Finite Element Analysis is the 4-noded linear tetrahedron. Section 2.6.1 defines the domain for this research project as a homogeneous block of isotropic tissue; ignoring vasculature. This approach enables simple mesh generation, while maintaining the extensibility of the simulation code to apply to irregular geometries meshed with linear tetrahedra.

Numbering of the nodes of the tetrahedra is of the utmost importance. Wrong numbering of tetrahedral element nodes can cause the volume calculated to be negative.

To number nodes:

- Pick a corner to be a node 4.
- From node 4, look towards the direction of the occluded face where nodes 1, 2 and 3 will be.
- From this view, number nodes 1, 2 and 3 counter-clockwise after defining node 1.

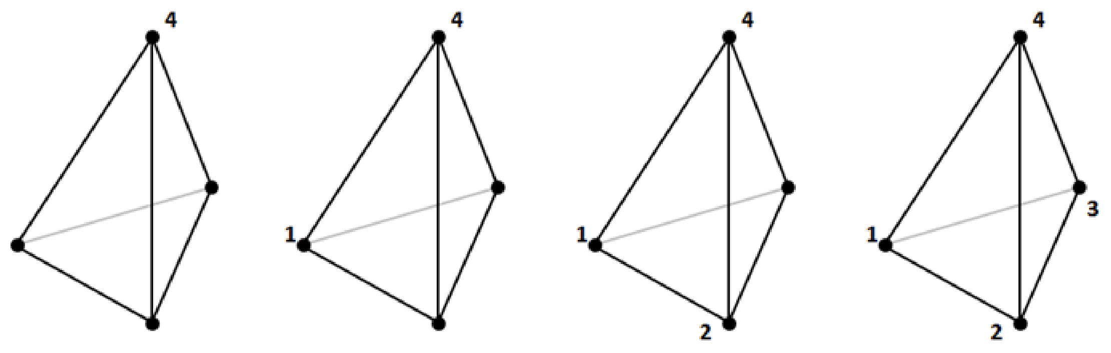


Figure 3-1: Linear Tetrahedron Node Numbering

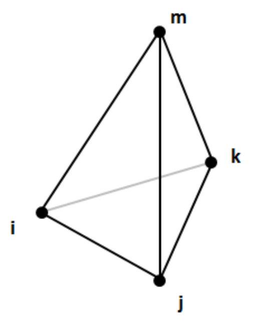


Figure 3-2: Linear Tetrahedron Node Numbering (Algebraic Notation)

Meshes for the research project are generated by an algorithm which first defines a rectangular prismatic domain divided into equally sized cubes, and then subdivides each cube into six tetrahedra of equal volume (see Figure 3-3).

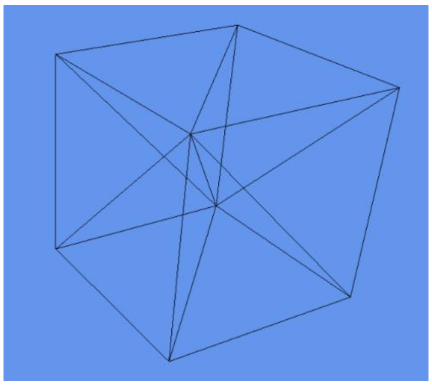


Figure 3-3: Cube Subdivision into Tetrahedra

Linear trial functions are selected for modelling field value variations across the tetrahedral elements (hence linear tetrahedra). Since the aim is to produce a simulation that runs at efficient speed (in order to achieve real-time results), the elements must be the simplest possible (using linear trial functions; quadratic and further may yield greater accuracies but at extra computational/time expenses).

Temperature T is represented in a tetrahedral element by:

$$T = N_i T_i + N_j T_j + N_k T_k + N_m T_m \tag{3.1}$$

or:

$$T = \begin{bmatrix} N_i & N_j & N_k & N_m \end{bmatrix} \begin{Bmatrix} T_i \\ T_j \\ T_k \\ T_m \end{Bmatrix} \tag{3.2}$$

where the shape functions are defined by linear trial functions as:

$$N_n = \frac{1}{6V}(a_n + b_n x + c_n y + d_n z) \tag{3.3}$$

such that:

$$[N] = [N_i \quad N_j \quad N_k \quad N_m] \tag{3.4}$$

where:

$$\begin{aligned} N_i &= \frac{1}{6V}(a_i + b_i x + c_i y + d_i z), \\ N_j &= \frac{1}{6V}(a_j + b_j x + c_j y + d_j z), \\ N_k &= \frac{1}{6V}(a_k + b_k x + c_k y + d_k z), \\ N_m &= \frac{1}{6V}(a_m + b_m x + c_m y + d_m z). \end{aligned} \tag{3.5}$$

Thus: $V, a_i, a_j, a_k, a_m, b_i, b_j, b_k, b_m, c_i, c_j, c_k, c_m, d_i, d_j, d_k$ & d_m must be determined. These values are in terms of nodal coordinates.

$$\begin{aligned} a_i &= \begin{vmatrix} x_j & y_j & z_j \\ x_k & y_k & z_k \\ x_m & y_m & z_m \end{vmatrix}, \qquad a_j = -\begin{vmatrix} x_i & y_i & z_i \\ x_k & y_k & z_k \\ x_m & y_m & z_m \end{vmatrix}, \\ a_k &= \begin{vmatrix} x_i & y_i & z_i \\ x_j & y_j & z_j \\ x_m & y_m & z_m \end{vmatrix}, \qquad a_m = -\begin{vmatrix} x_i & y_i & z_i \\ x_j & y_j & z_j \\ x_k & y_k & z_k \end{vmatrix} \end{aligned} \tag{3.6}$$

$$b_i = -\begin{vmatrix} 1 & y_j & z_j \\ 1 & y_k & z_k \\ 1 & y_m & z_m \end{vmatrix}, \qquad b_j = \begin{vmatrix} 1 & y_i & z_i \\ 1 & y_k & z_k \\ 1 & y_m & z_m \end{vmatrix},$$
$$b_k = -\begin{vmatrix} 1 & y_i & z_i \\ 1 & y_j & z_j \\ 1 & y_m & z_m \end{vmatrix}, \qquad b_m = \begin{vmatrix} 1 & y_i & z_i \\ 1 & y_j & z_j \\ 1 & y_k & z_k \end{vmatrix} \tag{3.7}$$

$$c_i = \begin{vmatrix} 1 & x_j & z_j \\ 1 & x_k & z_k \\ 1 & x_m & z_m \end{vmatrix}, \qquad c_j = -\begin{vmatrix} 1 & x_i & z_i \\ 1 & x_k & z_k \\ 1 & x_m & z_m \end{vmatrix},$$
$$c_k = \begin{vmatrix} 1 & x_i & z_i \\ 1 & x_j & z_j \\ 1 & x_m & z_m \end{vmatrix}, \qquad c_m = -\begin{vmatrix} 1 & x_i & z_i \\ 1 & x_j & z_j \\ 1 & x_k & z_k \end{vmatrix} \tag{3.8}$$

$$d_i = -\begin{vmatrix} 1 & x_j & y_j \\ 1 & x_k & y_k \\ 1 & x_m & y_m \end{vmatrix}, \qquad d_j = \begin{vmatrix} 1 & x_i & y_i \\ 1 & x_k & y_k \\ 1 & x_m & y_m \end{vmatrix},$$
$$d_k = -\begin{vmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_m & y_m \end{vmatrix}, \qquad d_m = \begin{vmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{vmatrix} \tag{3.9}$$

The volume of a tetrahedron is expressed as:

$$6V = \begin{vmatrix} 1 & x_i & y_i & z_i \\ 1 & x_j & y_j & z_j \\ 1 & x_k & y_k & z_k \\ 1 & x_m & y_m & z_m \end{vmatrix}. \tag{3.10}$$

Since the spatial derivative of T is:

$$\frac{\partial T}{\partial x} = \frac{\partial N_i}{\partial x}T_i + \frac{\partial N_j}{\partial x}T_j + \frac{\partial N_k}{\partial x}T_k + \frac{\partial N_m}{\partial x}T_m \tag{3.11}$$

…Etc. for $\frac{\partial T}{\partial y}$ and $\frac{\partial T}{\partial z}$

thus, the gradients of the shape functions are as shown in the derivative matrix:

$$[B] = \frac{1}{6V}\begin{bmatrix} b_i & b_j & b_k & b_m \\ c_i & c_j & c_k & c_m \\ d_i & d_j & d_k & d_m \end{bmatrix} \tag{3.12}$$

where:

$$\frac{\partial N_i}{\partial x} = \frac{b_i}{6V}, \qquad \frac{\partial N_i}{\partial y} = \frac{c_i}{6V}, \qquad \frac{\partial N_i}{\partial z} = \frac{d_i}{6V}. \tag{3.13}$$

Furthermore, the following integration formulae apply to linear tetrahedral elements (Eisenberg & Malvern 1973; Roland W. Lewis, Perumal Nithiarasu & Seetharamu 2004):

Over the tetrahedral element's three-dimensional domain:

$$\int_{\Omega} N_i^a N_j^b N_k^c N_m^d d\Omega = \int_{V} N_i^a N_j^b N_k^c N_m^d dV = \frac{a!\,b!\,c!\,d!\,6V}{(a+b+c+d+3)!} \tag{3.14}$$

and:

$$V = \int dx_1 dx_2 dx_3 = \frac{1}{6}\begin{vmatrix} 1 & x_{1i} & x_{2i} & x_{3i} \\ 1 & x_{1j} & x_{2j} & x_{3j} \\ 1 & x_{1k} & x_{2k} & x_{3k} \\ 1 & x_{1m} & x_{2m} & x_{3m} \end{vmatrix}. \tag{3.15}$$

On the boundaries:

$$\int\limits_{\Gamma} N_{B_1}^a N_{B_2}^b N_{B_3}^c \, d\Gamma = \int\limits_{A} N_{B_1}^a N_{B_2}^b N_{B_3}^c \, dA = \frac{a!\,b!\,c!\,2A_{B_1B_2B_3}}{(a+b+c+2)!} \tag{3.16}$$

where $A_{B_1B_2B_3}$ is the area of a triangular face on the boundary, (in 3D space), comprised of three nodes on the boundary; B_1, B_2, B_3. Note that B_1, B_2, B_3 can each equal any of nodes i, j, k, m, however, $B_1 \neq B_2 \neq B_3$, thus three of the tetrahedral element's nodes comprise a triangular face which represents a boundary surface whereupon a boundary condition is applied. Note that the unused node of the tetrahedron will have a shape function of zero, since it has no contribution or association with the boundary condition face.

$A_{B_1B_2B_3}$ can be calculated by finding half of the area of the parallelogram that is created when taking the magnitude of the cross product of two specific vectors; one vector from B_1 to B_2, and the other from B_1 to B_3.

$$\frac{1}{2}\left|\overrightarrow{B_1B_2} \times \overrightarrow{B_1B_3}\right| \tag{3.17}$$

3.2 Equation Discretization

3.2.1 Spatial Discretization of the Bioheat Equation: Formulations

Referring to Equation (2.1) in section 2.7.1, by expanding the spatial term for the Pennes Bioheat equation in 3D, the strong formulation of the equation can be expanded and written as:

$$k_{ti_x}\frac{\partial^2 T_{ti}}{\partial x^2} + k_{ti_y}\frac{\partial^2 T_{ti}}{\partial y^2} + k_{ti_z}\frac{\partial^2 T_{ti}}{\partial z^2} + W_{bl}C_{bl}T_{art} - W_{bl}C_{bl}T_{ti} + Q_m + Q_r - \rho_{ti}C_{ti}\frac{\partial T_{ti}}{\partial t} = 0 \tag{3.18}$$

where k_x, k_y, and k_z are thermal conductivities in the x, y and z directions respectively. Thus k_{ti_x}, k_{ti_y}, and k_{ti_z} are those thermal conductivity properties of the biological tissue. (These properties can also be temperature-dependent, as with other tissue properties.)

The weak formulation of the Pennes Bioheat equation is found by multiplying the strong formulation with a weighting function w, and integrating over the domain Ω.

$$\int_{\Omega} w\left[k_{ti_x}\frac{\partial^2 T_{ti}}{\partial x^2} + k_{ti_y}\frac{\partial^2 T_{ti}}{\partial y^2} + k_{ti_z}\frac{\partial^2 T_{ti}}{\partial z^2} + W_{bl}C_{bl}T_{art} - W_{bl}C_{bl}T_{ti} + Q_m + Q_r - \rho_{ti}C_{ti}\frac{\partial T_{ti}}{\partial t}\right] d\Omega = 0 \tag{3.19}$$

$$\int_{\Omega} w\left[\frac{\partial}{\partial x}\left(k_{ti_x}\frac{\partial T_{ti}}{\partial x}\right) + \frac{\partial}{\partial y}\left(k_{ti_y}\frac{\partial T_{ti}}{\partial y}\right) + \frac{\partial}{\partial z}\left(k_{ti_z}\frac{\partial T_{ti}}{\partial z}\right) + W_{bl}C_{bl}T_{art} - W_{bl}C_{bl}T_{ti} + Q_m + Q_r - \rho_{ti}C_{ti}\frac{\partial T_{ti}}{\partial t}\right] d\Omega = 0 \tag{3.20}$$

For components such as: $\int_{\Omega}\frac{\partial}{\partial x}\left(k_{ti_x}\frac{\partial T_{ti}}{\partial x}\right) w d\Omega$, integration by parts is used:

$$\int u(x)v'(x)dx = uv - \int u'(x)v(x)dx \tag{3.21}$$

such that:

$$\int_{\Omega}\frac{\partial}{\partial x}\left(k_{ti_x}\frac{\partial T_{ti}}{\partial x}\right) w d\Omega = \int_{\Gamma} w k_{ti_x}\frac{\partial T_{ti}}{\partial x} n_x d\Gamma - \int_{\Omega} k_{ti_x}\frac{\partial w}{\partial x}\frac{\partial T_{ti}}{\partial x} d\Omega \tag{3.22}$$

where n_x is a direction cosine (component of boundary normal), and Γ represents the boundary of the domain Ω.

A similar form of this formula, called Green’s Function or Green’s Lemma, may be used instead, however achieves the same results.

Thus for the spatial terms of Equation (3.20):

$$\begin{aligned}\int_{\Omega} w\left[\frac{\partial}{\partial x}\left(k_{ti_x}\frac{\partial T_{ti}}{\partial x}\right)+\frac{\partial}{\partial y}\left(k_{ti_y}\frac{\partial T_{ti}}{\partial y}\right)+\frac{\partial}{\partial z}\left(k_{ti_z}\frac{\partial T_{ti}}{\partial z}\right)\right]d\Omega \\ = -\int_{\Omega}\left[k_{ti_x}\frac{\partial w}{\partial x}\frac{\partial T_{ti}}{\partial x}+k_{ti_y}\frac{\partial w}{\partial y}\frac{\partial T_{ti}}{\partial y}+k_{ti_z}\frac{\partial w}{\partial z}\frac{\partial T_{ti}}{\partial z}\right]d\Omega \\ + \int_{\Gamma} w\left[k_{ti_x}\frac{\partial T_{ti}}{\partial x}n_x+k_{ti_y}\frac{\partial T_{ti}}{\partial y}n_y+k_{ti_z}\frac{\partial T_{ti}}{\partial z}n_z\right]d\Gamma\end{aligned} \tag{3.23}$$

where n_x, n_y, and n_z are direction cosines (axial components) of the outward direction normal to the boundary surface.

Boundary (Γ) conditions are commonly divided into the following four possible types for heat transfer problems:

Table 3.1: Boundary Condition Types for Heat Transfer Problems

Type	**Boundary Component**	**Category**
Temperature	Γ_T	Dirichlet
Heat Flow	Γ_{qf}	Neumann
Convection	Γ_{qc}	Neumann
Radiation	Γ_{qr}	Neumann

Temperature is specified for a Dirichlet Boundary Condition (on Γ_T), and heat flux is defined for a Neumann Boundary Condition (on Γ_q). Note that:

$$\Gamma = \Gamma_T \cup \Gamma_q \tag{3.24}$$

and:

$$\Gamma_q = \Gamma_{qf} + \Gamma_{qc} + \Gamma_{qr}. \tag{3.25}$$

Consider the following thermal boundary condition types:

$$T = T_b(x,y,z,t) \text{ on } \Gamma_T \tag{3.26}$$

$$q_x n_x + q_y n_y + q_z n_z = -q_b \text{ on } \Gamma_{qf} \tag{3.27}$$

$$q_x n_x + q_y n_y + q_z n_z = h(T - T_a) \text{ on } \Gamma_{qc} \tag{3.28}$$

$$q_x n_x + q_y n_y + q_z n_z = \sigma\varepsilon T^4 - \alpha q_r \text{ on } \Gamma_{qr} \tag{3.29}$$

where T_b is a prescribed boundary temperature, q_b is a prescribed boundary heat flow (flux), h is the convection coefficient, T_a is the atmospheric temperature, σ is the Stefan-Boltzmann constant, ε is the surface emission coefficient, α is the surface absorption coefficient, and q_r is the incident radiant heat flow per unit surface area.

By applying Fourier’s Law, where, for example in 1D:

$$q_x = -k_x \frac{\partial T}{\partial x} \tag{3.30}$$

then applying in 3D, the heat flux boundary conditions can be written as:

$$k_x \frac{\partial T}{\partial x} n_x + k_y \frac{\partial T}{\partial y} n_y + k_z \frac{\partial T}{\partial z} n_z = q_b \text{ on } \Gamma_{qf} \tag{3.31}$$

$$k_x \frac{\partial T}{\partial x} n_x + k_y \frac{\partial T}{\partial y} n_y + k_z \frac{\partial T}{\partial z} n_z = -h(T - T_a) \text{ on } \Gamma_{qc} \tag{3.32}$$

$$k_x \frac{\partial T}{\partial x} n_x + k_y \frac{\partial T}{\partial y} n_y + k_z \frac{\partial T}{\partial z} n_z = -(\sigma \varepsilon T^4 - \alpha q_r) \text{ on } \Gamma_{qr} \tag{3.33}$$

thus the boundary conditions can be collated as follows:

$$\begin{aligned} \int_\Gamma w \left[k_{ti_x} \frac{\partial T_{ti}}{\partial x} n_x + k_{ti_y} \frac{\partial T_{ti}}{\partial y} n_y + k_{ti_z} \frac{\partial T_{ti}}{\partial z} n_z \right] d\Gamma \\ = \int_{\Gamma_T} w \left[k_{ti_x} \frac{\partial T_{ti}}{\partial x} n_x + k_{ti_y} \frac{\partial T_{ti}}{\partial y} n_y + k_{ti_z} \frac{\partial T_{ti}}{\partial z} n_z \right] d\Gamma \\ + \int_{\Gamma_{qf}} w q_b d\Gamma - \int_{\Gamma_{qc}} w[h(T_{ti} - T_a)] d\Gamma \\ - \int_{\Gamma_{qr}} w\left[\sigma \varepsilon T_{ti}^4 - \alpha q_r\right] d\Gamma \end{aligned} \tag{3.34}$$

and by applying Fourier’s Law as shown in Equation (3.30):

$$\begin{aligned} \int_\Gamma w \left[k_{ti_x} \frac{\partial T_{ti}}{\partial x} n_x + k_{ti_y} \frac{\partial T_{ti}}{\partial y} n_y + k_{ti_z} \frac{\partial T_{ti}}{\partial z} n_z \right] d\Gamma \\ = -\int_{\Gamma_T} w\{q\}^T\{n\} d\Gamma + \int_{\Gamma_{qf}} w q_b d\Gamma - \int_{\Gamma_{qc}} w[h(T_{ti} - T_a)] d\Gamma \\ - \int_{\Gamma_{qr}} w\left[\sigma \varepsilon T_{ti}^4 - \alpha q_r\right] d\Gamma \end{aligned} \tag{3.35}$$

where, in 3D:

$$\{q\}^T = [q_x \quad q_y \quad q_z] \tag{3.36}$$

and:

$$\{n\} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}. \tag{3.37}$$

Subbing back into Equation (3.23):

$$
\begin{aligned}
\int_{\Omega} w\left[k_{ti_x}\frac{\partial^2 T_{ti}}{\partial x^2}+k_{ti_y}\frac{\partial^2 T_{ti}}{\partial y^2}+k_{ti_z}\frac{\partial^2 T_{ti}}{\partial z^2}\right]d\Omega \\
= -\int_{\Omega}\left[k_{ti_x}\frac{\partial w}{\partial x}\frac{\partial T_{ti}}{\partial x}+k_{ti_y}\frac{\partial w}{\partial y}\frac{\partial T_{ti}}{\partial y}+k_{ti_z}\frac{\partial w}{\partial z}\frac{\partial T_{ti}}{\partial z}\right]d\Omega \\
-\int_{\Gamma_T} w\{q\}^T\{n\}d\Gamma+\int_{\Gamma_{qf}} wq_b d\Gamma-\int_{\Gamma_{qc}} w[h(T_{ti}-T_a)]d\Gamma \\
-\int_{\Gamma_{qr}} w\left[\sigma\varepsilon T_{ti}^4-\alpha q_r\right]d\Gamma
\end{aligned}
\tag{3.38}
$$

and finally subbing back into Equation (3.20), yields the weak formulation of the governing bioheat transfer equation:

$$
\begin{aligned}
\int_{\Omega} w\left[W_{bl}C_{bl}T_{art}-W_{bl}C_{bl}T_{ti}+Q_m+Q_r-\ \rho_{ti}C_{ti}\frac{\partial T_{ti}}{\partial t}\right]d\Omega \\
-\int_{\Omega}\left[k_{ti_x}\frac{\partial w}{\partial x}\frac{\partial T_{ti}}{\partial x}+k_{ti_y}\frac{\partial w}{\partial y}\frac{\partial T_{ti}}{\partial y}+k_{ti_z}\frac{\partial w}{\partial z}\frac{\partial T_{ti}}{\partial z}\right]d\Omega \\
-\int_{\Gamma_T} w\{q\}^T\{n\}d\Gamma+\int_{\Gamma_{qf}} wq_b d\Gamma-\int_{\Gamma_{qc}} w[h(T_{ti}-T_a)]d\Gamma \\
-\int_{\Gamma_{qr}} w\left[\sigma\varepsilon T_{ti}^4-\alpha q_r\right]d\Gamma = 0
\end{aligned}
\tag{3.39}
$$

with boundary conditions defined on Γ_T, Γ_{qf}, Γ_{qc} and Γ_{qr}.

and initial conditions:

$$T = T_0 \text{ at } t = 0.0.$$

3.2.2 Spatial Discretization of the Bioheat Equation: Galerkin Method - FEM

Demonstrated in Roland W. Lewis, Perumal Nithiarasu and Seetharamu (2004) and Nikishkov (2010), the (Weighted Residual) Galerkin method of finite element discretization is accurate and effective when applied to partial differential equations. The Galerkin Method's uniqueness lies in the fact that the element's shape functions are substituted for the weighting functions of the weighted residual method (section 2.8.1).

For a finite element, the field value (i.e. temperature) is approximated by some trial function, which yields shape functions in accordance with the element geometry, such that the field value can be interpolated by the shape functions and element nodal field values.

$$T = [N]\{T\} \tag{3.40}$$

A similar representation can be given for a weighting function w.

$$w = [N][w] \tag{3.41}$$

$$w^T = [w]^T[N]^T \tag{3.42}$$

w is scalar, and since the transpose of a scalar is its self:

$$w = w^T = [w]^T[N]^T. \tag{3.43}$$

$[w]^T$ is arbitrary, so therefore:

$$w = [N]^T. \tag{3.44}$$

Also note then, that:

$$\frac{\partial w}{\partial x} = \left[\frac{\partial N}{\partial x}\right]^T, \ \frac{\partial w}{\partial y} = \left[\frac{\partial N}{\partial y}\right]^T, \frac{\partial w}{\partial z} = \left[\frac{\partial N}{\partial z}\right]^T \tag{3.45}$$

and:

$$\frac{\partial T}{\partial x} = \left[\frac{\partial N}{\partial x}\right]\{T\}, \ \frac{\partial T}{\partial y} = \left[\frac{\partial N}{\partial y}\right]\{T\}, \frac{\partial T}{\partial z} = \left[\frac{\partial N}{\partial z}\right]\{T\}. \tag{3.46}$$

This introduces the Galerkin Method, where the weighting functions are replaced by the nodal shape functions. Thus Equation (3.39) becomes:

$$\begin{aligned}\int_{\Omega} \Bigg[& W_{bl}C_{bl}T_{art}[N]^T - W_{bl}C_{bl}[N]^T[N]\{T_{ti}\} + Q_m[N]^T + Q_r[N]^T \\ & - \rho_{ti}C_{ti}[N]^T[N]\left\{\frac{\partial T_{ti}}{\partial t}\right\}\Bigg] d\Omega \\ & - \int_{\Omega}\Bigg[k_{ti_x}\left[\frac{\partial N}{\partial x}\right]^T\left[\frac{\partial N}{\partial x}\right]\{T_{ti}\} + k_{ti_y}\left[\frac{\partial N}{\partial y}\right]^T\left[\frac{\partial N}{\partial y}\right]\{T_{ti}\} \\ & + k_{ti_z}\left[\frac{\partial N}{\partial z}\right]^T\left[\frac{\partial N}{\partial z}\right]\{T_{ti}\}\Bigg] d\Omega - \int_{\Gamma_T}\{q\}^T\{n\}[N]^T d\Gamma \\ & + \int_{\Gamma_{qf}} q_b[N]^T d\Gamma - \int_{\Gamma_{qc}}[h[N]^T[N]\{T_{ti}\} - hT_a[N]^T]d\Gamma \\ & - \int_{\Gamma_{qr}}\left[\sigma\varepsilon T_{ti}^4[N]^T - \alpha q_r[N]^T\right]d\Gamma = 0.\end{aligned} \tag{3.47}$$

By differentiating the temperature-interpolation equation shown in Equation (3.40):

$$\begin{Bmatrix} \frac{\partial T}{\partial x} \\ \frac{\partial T}{\partial y} \\ \frac{\partial T}{\partial z} \end{Bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \cdots \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \cdots \\ \frac{\partial N_1}{\partial z} & \frac{\partial N_2}{\partial z} & \cdots \end{bmatrix} \{T\} = [B]\{T\} \tag{3.48}$$

where $[B]$ is the derivative/gradient matrix; in this case, a matrix of temperature gradient interpolation; in 3D for this case.

Furthermore the thermal conductivities ("thermal stiffness") can be represented in matrix form:

$$[D_{ti}] = \begin{bmatrix} k_{ti_x} & 0 & 0 \\ 0 & k_{ti_y} & 0 \\ 0 & 0 & k_{ti_z} \end{bmatrix}. \tag{3.49}$$

By substituting the derivative and thermal conductivity matrices into Equation (3.47):

$$\begin{aligned} \int_\Omega \Big[W_{bl}C_{bl}T_{art}[N]^T - W_{bl}C_{bl}[N]^T[N]\{T_{ti}\} + Q_m[N]^T + Q_r[N]^T \\ - \rho_{ti}C_{ti}[N]^T[N]\left\{\frac{\partial T_{ti}}{\partial t}\right\}\Big] d\Omega - \int_\Omega [B]^T[D_{ti}][B]\{T_{ti}\}d\Omega \\ - \int_{\Gamma_T} \{q\}^T\{n\}[N]^T d\Gamma + \int_{\Gamma_{qf}} q_b[N]^T d\Gamma \\ - \int_{\Gamma_{qc}} [h[N]^T[N]\{T_{ti}\} - hT_a[N]^T]d\Gamma \\ - \int_{\Gamma_{qr}} \left[\sigma\varepsilon T_{ti}^4[N]^T - \alpha q_r[N]^T\right]d\Gamma = 0. \end{aligned} \tag{3.50}$$

Expanding:

$$\begin{aligned} \int_\Omega W_{bl}C_{bl}T_{art}[N]^T d\Omega - \int_\Omega W_{bl}C_{bl}[N]^T[N]\{T_{ti}\}d\Omega + \int_\Omega Q_m[N]^T d\Omega \\ + \int_\Omega Q_r[N]^T d\Omega - \int_\Omega \rho_{ti}C_{ti}[N]^T[N]\left\{\frac{\partial T_{ti}}{\partial t}\right\}d\Omega \\ - \int_\Omega [B]^T[D_{ti}][B]\{T_{ti}\}d\Omega - \int_{\Gamma_T} \{q\}^T\{n\}[N]^T d\Gamma \\ + \int_{\Gamma_{qf}} q_b[N]^T d\Gamma - \int_{\Gamma_{qc}} h[N]^T[N]\{T_{ti}\}d\Gamma + \int_{\Gamma_{qc}} hT_a[N]^T d\Gamma \\ - \int_{\Gamma_{qr}} \sigma\varepsilon T_{ti}^4[N]^T d\Gamma + \int_{\Gamma_{qr}} \alpha q_r[N]^T d\Gamma = 0. \end{aligned} \tag{3.51}$$

Note that k_{ti} will be subbed for $[D_{ti}]$, since it will be assumed that the thermal conductivity in the tissue will be the same for all axes; $k_{ti_x} = k_{ti_y} = k_{ti_z}$.

If the equation is then rearranged:

$$
\begin{aligned}
-\int_{\Omega} \rho_{ti} C_{ti} [N]^T [N] \left\{\frac{\partial T_{ti}}{\partial t}\right\} d\Omega - \int_{\Omega} k_{ti} [B]^T [B] \{T_{ti}\} d\Omega \\
- \int_{\Omega} W_{bl} C_{bl} [N]^T [N] \{T_{ti}\} d\Omega - \int_{\Gamma_{qc}} h [N]^T [N] \{T_{ti}\} d\Gamma \\
- \int_{\Gamma_{qr}} \sigma \varepsilon T_{ti}^4 [N]^T d\Gamma + \int_{\Omega} W_{bl} C_{bl} T_{art} [N]^T d\Omega + \int_{\Omega} Q_m [N]^T d\Omega \\
+ \int_{\Omega} Q_r [N]^T d\Omega - \int_{\Gamma_T} \{q\}^T \{n\} [N]^T d\Gamma + \int_{\Gamma_{qf}} q_b [N]^T d\Gamma \\
+ \int_{\Gamma_{qc}} h T_a [N]^T d\Gamma + \int_{\Gamma_{qr}} \alpha q_r [N]^T d\Gamma = 0.
\end{aligned}
\tag{3.52}
$$

If we let:

Table 3.2: Discretized Bioheat Transfer Equation Components

Capacitance Matrix	$[C] = \int_{\Omega} \rho_{ti} C_{ti} [N]^T [N] d\Omega$
Thermal Stiffness Matrix (conduction)	$[K_C] = \int_{\Omega} k_{ti} [B]^T [B] d\Omega$
Thermal Stiffness Matrix (Perfusion)	$[K_p] = \int_{\Omega} W_{bl} C_{bl} [N]^T [N] d\Omega$
Thermal Stiffness Matrix (convection)	$[K_h] = \int_{\Gamma_q} h [N]^T [N] d\Gamma_q$
Thermal Stiffness Matrix (radiation)	$[K_r]\{T\} = \int_{\Gamma_{qr}} \sigma \varepsilon T_{ti}^4 [N]^T d\Gamma$
Thermal Load/Forcing-Vector (perfusion)	$\{R_p\} = \int_{\Omega} W_{bl} C_{bl} T_{art} [N]^T d\Omega$
Thermal Load/Forcing-Vector (metabolic heat generation)	$\{R_{Qm}\} = \int_{\Omega} Q_m [N]^T d\Omega$
Thermal Load/Forcing-Vector (regional heat sources) (*Do not confuse with radiative*)	$\{R_{Qr}\} = \int_{\Omega} Q_r [N]^T d\Omega$
Thermal Load/Forcing-Vector (Temperature boundary condition)	$\{R_T\} = -\int_{\Gamma_T} \{q\}^T \{n\} [N]^T d\Gamma$
Thermal Load/Forcing-Vector (Heat Flux boundary condition)	$\{R_{qf}\} = \int_{\Gamma_{qf}} q_b [N]^T d\Gamma$
Thermal Load/Forcing-Vector (Convection boundary condition)	$\{R_{qc}\} = \int_{\Gamma_{qc}} h T_a [N]^T d\Gamma$
Thermal Load/Forcing-Vector (Radiation boundary condition)	$\{R_{qr}\} = \int_{\Gamma_{qr}} \alpha q_r [N]^T d\Gamma$

Then Equation (3.52) becomes:

$$
\begin{aligned}
-[C]\left\{\frac{\partial T_{ti}}{\partial t}\right\} - \left([K_C] + [K_p] + [K_h] + [K_R]\right)\{T_{ti}\} + \{R_p\} + \{R_{Qm}\} + \{R_{Qr}\} \\
+ \{R_T\} + \{R_{qf}\} + \{R_{qc}\} + \{R_{qr}\} = 0.
\end{aligned}
\tag{3.53}
$$

Rearranging:

$$[C]\left\{\frac{\partial T_{ti}}{\partial t}\right\} + \left([K_C] + [K_p] + [K_h] + [K_R]\right)\{T_{ti}\} \\ = \{R_p\} + \{R_{Qm}\} + \{R_{Qr}\} + \{R_T\} + \{R_{qf}\} + \{R_{qc}\} + \{R_{qr}\}. \tag{3.54}$$

Note that radiative boundary conditions and radiative thermal stiffness will not be considered for this project, due to the nature of the problem in question (biological tissue).

Note that convective boundary conditions and convective thermal stiffness will not be considered for this project, due to the nature of the problem in question (biological organ, not subjected to convective boundary conditions due to its position *in-vivo*).

Thus $[K_h]$, $[K_R]$, $\{R_{qc}\}$ and $\{R_{qr}\}$ will be removed from further consideration. Equation (3.54) becomes:

$$[C]\left\{\frac{\partial T_{ti}}{\partial t}\right\} + \left([K_C] + [K_p]\right)\{T_{ti}\} = \{R_p\} + \{R_{Qm}\} + \{R_{Qr}\} + \{R_T\} + \{R_{qf}\}. \tag{3.55}$$

Let the forcing vector:

$$\{R\} = \{R_p\} + \{R_{Qm}\} + \{R_{Qr}\} + \{R_T\} + \{R_{qf}\} \tag{3.56}$$

and the stiffness matrix:

$$[K] = [K_C] + [K_p] \tag{3.57}$$

therefore Equation (3.55) becomes:

$$[C]\left\{\frac{\partial T_{ti}}{\partial t}\right\} + [K]\{T_{ti}\} = \{R\}. \tag{3.58}$$

3.2.3 Temporal Discretization of the Bioheat Equation: Finite Difference Method

The time-dependent term still needs to be discretized. This is achieved using the finite difference method (FDM). Using a Taylor series:

$$T^{n+1} = T^n + \Delta t \frac{\partial T^n}{\partial t} + \frac{\Delta t^2}{2} \frac{\partial^2 T^n}{\partial t^2} + \cdots \tag{3.59}$$

By dividing all terms by Δt and truncating the second and higher-order terms, the series becomes:

$$\frac{\partial T^n}{\partial t} \approx \frac{T^{n+1} - T^n}{\Delta t} + O(\Delta t). \tag{3.60}$$

If a parameter θ is introduced such that:

$$T^{n+\theta} = \theta T^{n+1} + (1-\theta) T^n \tag{3.61}$$

then Equation (3.58) becomes:

$$[C]\left\{\frac{T_{ti}^{n+1} - T_{ti}^{n}}{\Delta t}\right\} + [K]\{\theta T_{ti}^{n+1} + (1-\theta)T_{ti}^{n}\} = \theta\{R\}^{n+1} + (1-\theta)\{R\}^n \tag{3.62}$$

which can be arranged to:

$$\begin{aligned}([C] + \theta\Delta t[K])\{T_{ti}\}^{n+1} &\\ = ([C] - (1-\theta)\Delta t[K])\{T_{ti}\}^n &\\ + \Delta t(\theta\{R\}^{n+1} + (1-\theta)\{R\}^n)&\end{aligned} \tag{3.63}$$

where the type of scheme can be modified based on the implicitness parameter θ.

Table 3.3: Finite Difference Discretization Schemes

θ	Scheme	(Method)	Stability
0.0	Fully explicit	Forward Difference	Conditionally Stable
1.0	Fully implicit	Backward Difference	Unconditionally Stable
0.5	Semi-implicit	Crank-Nicolson	Marginally Stable

The fully explicit scheme is selected for this application due to its simplicity. Thus $\theta = 0.0$.

Therefore, for a fully explicit scheme, Equation (3.63) becomes:

$$[C]\{T_{ti}\}^{n+1} = [C]\{T_{ti}\}^n - \Delta t[K]\{T_{ti}\}^n + \Delta t\{R\}^n \tag{3.64}$$

or:

$$\begin{aligned}[C]\{T_{ti}\}^{n+1} = [C]\{T_{ti}\}^n - \Delta t\big([K_C] + [K_p]\big)\{T_{ti}\}^n &\\ + \Delta t\Big(\{R_p\}^n + \{R_{Qm}\}^n + \{R_{Qr}\}^n + \{R_T\}^n + \{R_{qf}\}^n\Big)&\end{aligned} \tag{3.65}$$

and thus the entirety of the right hand side of the equation, since it is known, will sum to become a vector $\{f\}$, and the final equation which must then be computationally addressed is:

$$[C]\{T_{ti}\}^{n+1} = \{f\}^n. \tag{3.66}$$

3.2.4 Matrix Formulation

If the relevant components from Table 3.2, and the shape functions for the selected element (linear tetrahedron) in Equation (3.4) are considered, then the result is:

Table 3.4: Discretized Components for a Linear Tetrahedral Element

Capacitance Matrix	$[C] = \int_{\Omega} \rho_{ti} C_{ti} \begin{bmatrix} N_i^2 & N_i N_j & N_i N_k & N_i N_m \\ N_j N_i & N_j^2 & N_j N_k & N_j N_m \\ N_k N_i & N_k N_j & N_k^2 & N_k N_m \\ N_m N_i & N_m N_j & N_m N_k & N_m^2 \end{bmatrix} d\Omega$
Thermal Stiffness Matrix (conduction)	$[K_C] = \int_{\Omega} k_{ti} [B]^T [B] d\Omega$
Thermal Stiffness Matrix (Perfusion)	$[K_p] = \int_{\Omega} W_{bl} C_{bl} \begin{bmatrix} N_i^2 & N_i N_j & N_i N_k & N_i N_m \\ N_j N_i & N_j^2 & N_j N_k & N_j N_m \\ N_k N_i & N_k N_j & N_k^2 & N_k N_m \\ N_m N_i & N_m N_j & N_m N_k & N_m^2 \end{bmatrix} d\Omega$
Thermal Load/Forcing-Vector (perfusion)	$\{R_p\} = \int_{\Omega} W_{bl} C_{bl} T_{art} \begin{bmatrix} N_i \\ N_j \\ N_k \\ N_m \end{bmatrix} d\Omega$
Thermal Load/Forcing-Vector (metabolic heat generation)	$\{R_{Qm}\} = \int_{\Omega} Q_m \begin{bmatrix} N_i \\ N_j \\ N_k \\ N_m \end{bmatrix} d\Omega$
Thermal Load/Forcing-Vector (regional heat sources)	$\{R_{Qr}\} = \int_{\Omega} Q_r \begin{bmatrix} N_i \\ N_j \\ N_k \\ N_m \end{bmatrix} d\Omega$
Thermal Load/Forcing-Vector (Temperature boundary condition)	$\{R_T\} = -\int_{\Gamma_T} q_n \begin{bmatrix} N_i \\ N_j \\ N_k \\ N_m \end{bmatrix} d\Gamma_T$
Thermal Load/Forcing-Vector (Heat Flux boundary condition)	$\{R_{qf}\} = \int_{\Gamma_{qf}} q_b \begin{bmatrix} N_i \\ N_j \\ N_k \\ N_m \end{bmatrix} d\Gamma$

Note that:

$$q_n = \{q\}^T\{n\} = [q_x \quad q_y \quad q_z] \begin{Bmatrix} n_x \\ n_y \\ n_z \end{Bmatrix}. \tag{3.67}$$

Employing the integration formulae for linear tetrahedral elements in Equations (3.14) and (3.16):

Table 3.5: Matrix/Vector - Form of Components for a Linear Tetrahedral Element

Capacitance Matrix	$[C] = \frac{\rho_{ti} C_{ti} V}{20} \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$
Thermal Stiffness Matrix (conduction)	
$[K_C] = \frac{k_{ti}}{36V} \begin{bmatrix} b_i^2 + c_i^2 + d_i^2 & b_i b_j + c_i c_j + d_i d_j & b_i b_k + c_i c_k + d_i d_k & b_i b_m + c_i c_m + d_i d_m \\ b_j b_i + c_j c_i + d_j d_i & b_j^2 + c_j^2 + d_j^2 & b_j b_k + c_j c_k + d_j d_k & b_j b_m + c_j c_m + d_j d_m \\ b_k b_i + c_k c_i + d_k d_i & b_k b_j + c_k c_j + d_k d_j & b_k^2 + c_k^2 + d_k^2 & b_k b_m + c_k c_m + d_k d_m \\ b_m b_i + c_m c_i + d_m d_i & b_m b_j + c_m c_j + d_m d_j & b_m b_k + c_m c_k + d_m d_k & b_m^2 + c_m^2 + d_m^2 \end{bmatrix}$	
Thermal Stiffness Matrix (Perfusion)	$[K_p] = \frac{W_{bl} C_{bl} V}{20} \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$
Thermal Load/Forcing-Vector (perfusion)	$\{R_p\} = \frac{W_{bl} C_{bl} T_{art} V}{4} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
Thermal Load/Forcing-Vector (metabolic heat generation)	$\{R_{Qm}\} = \frac{Q_m V}{4} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
Thermal Load/Forcing-Vector (regional heat sources)	$\{R_{Qr}\} = \frac{Q_r V}{4} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
Thermal Load/Forcing-Vector (Temperature boundary condition)	$\{R_T\} = \{R_{T_{jkm}}\} \cup \{R_{T_{ikm}}\} \cup \{R_{T_{ijm}}\} \cup \{R_{T_{ijk}}\}$ $\{R_{T_{jkm}}\} = -\frac{q_n A_{jkm}}{3} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \dots$
Thermal Load/Forcing-Vector (Heat Flux boundary condition)	$\{R_{qf}\} = \{R_{qf_{jkm}}\} \cup \{R_{qf_{ikm}}\} \cup \{R_{qf_{ijm}}\}$ $\cup \{R_{qf_{ijk}}\}$ $\{R_{qf_{jkm}}\} = \frac{q_b A_{jkm}}{3} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \dots$

Where the volume V of the element can be found by Equation (3.10), the area A of a triangular surface in three dimensional space can be found by Equation (3.17), and the parameters $\rho_{ti}, C_{ti}, k_{ti}, W_{bl}, C_{bl}, T_{art}, Q_m$ and Q_r are defined per node/element, or where appropriate are constant for all nodes/elements across the domain. Some of these parameters themselves, may be temperature-dependent, thus a solution would be obtained by solving for temperatures, then temperature-dependent parameters, then temperatures again, repeating the cycle until adequate convergence conditions are satisfied.

The boundary condition parameters of $\{R_T\}$ and $\{R_{qf}\}$; that is: q_n (by its components: $q_x, q_y, q_z, n_x, n_y, n_z$; see Equation (3.67)) and q_b, can be defined, the simplest boundary being a purely insulated one, such that $\{R_T\}$ is ignored, and:

$$\{R_{qf}\} = \{0\}. \tag{3.68}$$

Note that in Table 3.5, four options exist for $\{R_T\}$ and for $\{R_{qf}\}$. Consider Figure 3-2; the tetrahedron has four faces. Each face surface is a potential boundary segment for the domain comprised of tetrahedra. If a tetrahedral element contains multiple faces that serve as boundaries, then $\{R_T\}$ and $\{R_{qf}\}$ will be summations of the boundary condition contributions of the relevant faces (unions). Refer to Equation (3.16) and the relevant discussion.

Furthermore, in the case that some coordinate P located at some $(x_P, y_P,\ z_P, t_P)$ within the element is considered a point heat source of Q_{r_P} Watts (W), (as opposed to a uniform source over the entire element; $\frac{W}{m^3}$), then the regional heat source vector for the tetrahedral element containing it, becomes:

$$\{R_{Qr_P}\} = Q_{r_P} \begin{bmatrix} N_i \\ N_j \\ N_k \\ N_m \end{bmatrix}_{(x_P, y_P,\ z_P, t_P)} \tag{3.69}$$

thus distributing the appropriate contributions from the point heat source to the tetrahedral element nodes.

A similar approach could theoretically be taken to applying point temperature boundary conditions (or other forcing vectors). See Equation (3.26) to note that a specified temperature could be a point temperature. (Uniform temperatures can also be applied to the desired face.)

Furthermore, boundary temperatures may also be set by defining which nodes have known set temperatures, and substituting these values into the global matrix assembly such that they may contribute to the total forcing vector, and ensuring that new temperatures are not calculated for the specified nodes. Alternatively, the boundary condition vector $\{R_T\}$ is responsible for distributing surface temperatures to the nodes. The simplest method of implementing Dirichlet Boundary Conditions however, is to set and fix the nodal temperatures, regarding numerical solution of global matrices.

3.2.5 Element Matrices

Thus for each element:

Table 3.6: Detailed Matrix/Vector - Form of Components for a Linear Tetrahedral Element

Capacitance Matrix	$[C] = \frac{\rho_{ti} C_{ti} V}{20} \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$
Thermal Stiffness Matrix (conduction)	
$[K_C] = \frac{k_{ti}}{36V} \begin{bmatrix} b_i^2 + c_i^2 + d_i^2 & b_i b_j + c_i c_j + d_i d_j & b_i b_k + c_i c_k + d_i d_k & b_i b_m + c_i c_m + d_i d_m \\ b_j b_i + c_j c_i + d_j d_i & b_j^2 + c_j^2 + d_j^2 & b_j b_k + c_j c_k + d_j d_k & b_j b_m + c_j c_m + d_j d_m \\ b_k b_i + c_k c_i + d_k d_i & b_k b_j + c_k c_j + d_k d_j & b_k^2 + c_k^2 + d_k^2 & b_k b_m + c_k c_m + d_k d_m \\ b_m b_i + c_m c_i + d_m d_i & b_m b_j + c_m c_j + d_m d_j & b_m b_k + c_m c_k + d_m d_k & b_m^2 + c_m^2 + d_m^2 \end{bmatrix}$	
Thermal Stiffness Matrix (Perfusion)	$[K_p] = \frac{W_{bl} C_{bl} V}{20} \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$
Thermal Load/Forcing-Vector (perfusion)	$\{R_p\} = \frac{W_{bl} C_{bl} T_{art} V}{4} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
Thermal Load/Forcing-Vector (metabolic heat generation)	$\{R_{Qm}\} = \frac{Q_m V}{4} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
Thermal Load/Forcing-Vector (regional heat sources) - uniform source U $(\frac{W}{m^3})$ ○ Q_{r_U} - point source P (W) ○ Q_{r_P} ○ (x_P, y_P, z_P, t_P)	$\{R_{Qr}\} = \{R_{Qr_U}\} \cup \{R_{Qr_P}\}$ $\{R_{Qr_U}\} = \frac{Q_{r_U} V}{4} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ $\{R_{Qr_P}\} = Q_{r_P} \begin{bmatrix} N_i \\ N_j \\ N_k \\ N_m \end{bmatrix}_{(x_P, y_P, z_P)}$
Thermal Load/Forcing-Vector (Temperature boundary condition)	$\{R_T\} = \{R_{T_{jkm}}\} \cup \{R_{T_{ikm}}\} \cup \{R_{T_{ijm}}\} \cup \{R_{T_{ijk}}\}$ $\{R_{T_{jkm}}\} = -\frac{q_n A_{jkm}}{3} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix},$ $\{R_{T_{ikm}}\} = -\frac{q_n A_{ikm}}{3} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix},$ $\{R_{T_{ijm}}\} = -\frac{q_n A_{ijm}}{3} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix},$ $\{R_{T_{ijk}}\} = -\frac{q_n A_{ijk}}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$

Thermal Load/Forcing-Vector (Heat Flux boundary condition)	$\{R_{qf}\} = \{R_{qf_{jkm}}\} \cup \{R_{qf_{ikm}}\} \cup \{R_{qf_{ijm}}\} \cup \{R_{qf_{ijk}}\}$ $\{R_{qf_{jkm}}\} = \frac{q_b A_{jkm}}{3}\begin{bmatrix}0\\1\\1\\1\end{bmatrix},$ $\{R_{qf_{ikm}}\} = \frac{q_b A_{ikm}}{3}\begin{bmatrix}1\\0\\1\\1\end{bmatrix},$ $\{R_{qf_{ijm}}\} = \frac{q_b A_{ijm}}{3}\begin{bmatrix}1\\1\\0\\1\end{bmatrix},$ $\{R_{qf_{ijk}}\} = \frac{q_b A_{ijk}}{3}\begin{bmatrix}1\\1\\1\\0\end{bmatrix}$

Where the parameters $\rho_{ti}, C_{ti}, k_{ti}, W_{bl}, C_{bl}, T_{art}, Q_m, Q_{r_U}$ and Q_{r_P} are defined for the element or generally for the domain, as well as q_n and q_b. The parameters may be temperature-dependent and require iterative solutions to converge. The constants of the $[K_C]$ matrix are defined in Equations (3.6) through to (3.9), the element's volume V is found from Equation (3.10), and the area of a triangle (i.e. for a boundary condition) in three-dimensional space is found using Equation (3.17).

The discretized (spatial: FEM, temporal: FDM - fully explicit) governing equation for bioheat transfer is Equation (3.65).

With this information, the element matrices may be calculated and assembled.

3.2.6 Element Matrices – Temperature Dependence

If temperature dependence for thermal parameters is considered, then the thermal properties at each node of the element will vary with their respective nodal temperatures. Thus, if we consider thermal properties that vary per node: $\rho_{ti}, C_{ti}, k_{ti}, W_{bl}$ and C_{bl} vary per node due to temperature dependence. Q_{r_U} and Q_{r_P} (as well as V, and boundary condition parameters q_n, q_b and $A_{B_1B_2B_3}$) vary per element. Q_m and T_{art} are constant across the mesh, as the metabolic heat generation rate applies to the whole organ, as does the arterial blood temperature.

Table 3.7: Detailed Matrix/Vector - Form of Components for a Linear Tetrahedral Element (T-Dependent Thermal Parameters)

Capacitance Matrix	$[C] = \frac{V}{20}\begin{bmatrix} \rho_{ti_i}C_{ti_i} & 0 & 0 & 0 \\ 0 & \rho_{ti_j}C_{ti_j} & 0 & 0 \\ 0 & 0 & \rho_{ti_k}C_{ti_k} & 0 \\ 0 & 0 & 0 & \rho_{ti_m}C_{ti_m} \end{bmatrix}\begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$
Thermal Stiffness Matrix (conduction)	
$[K_C] = \frac{1}{36V}\begin{bmatrix} k_{ti_i} & 0 & 0 & 0 \\ 0 & k_{ti_j} & 0 & 0 \\ 0 & 0 & k_{ti_k} & 0 \\ 0 & 0 & 0 & k_{ti_m} \end{bmatrix} \cdot \begin{bmatrix} b_i^2 + c_i^2 + d_i^2 & b_ib_j + c_ic_j + d_id_j & b_ib_k + c_ic_k + d_id_k & b_ib_m + c_ic_m + d_id_m \\ b_jb_i + c_jc_i + d_jd_i & b_j^2 + c_j^2 + d_j^2 & b_jb_k + c_jc_k + d_jd_k & b_jb_m + c_jc_m + d_jd_m \\ b_kb_i + c_kc_i + d_kd_i & b_kb_j + c_kc_j + d_kd_j & b_k^2 + c_k^2 + d_k^2 & b_kb_m + c_kc_m + d_kd_m \\ b_mb_i + c_mc_i + d_md_i & b_mb_j + c_mc_j + d_md_j & b_mb_k + c_mc_k + d_md_k & b_m^2 + c_m^2 + d_m^2 \end{bmatrix}$	
Thermal Stiffness Matrix (Perfusion)	$[K_p] = \frac{V}{20}\begin{bmatrix} W_{bl_i}C_{bl_i} & 0 & 0 & 0 \\ 0 & W_{bl_j}C_{bl_j} & 0 & 0 \\ 0 & 0 & W_{bl_k}C_{bl_k} & 0 \\ 0 & 0 & 0 & W_{bl_m}C_{bl_m} \end{bmatrix}\begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$
Thermal Load/Forcing-Vector (perfusion)	$\{R_p\} = \frac{T_{art}V}{4}\begin{bmatrix} W_{bl_i}C_{bl_i} & 0 & 0 & 0 \\ 0 & W_{bl_j}C_{bl_j} & 0 & 0 \\ 0 & 0 & W_{bl_k}C_{bl_k} & 0 \\ 0 & 0 & 0 & W_{bl_m}C_{bl_m} \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
Thermal Load/Forcing-Vector (metabolic heat generation)	$\{R_{Qm}\} = \frac{Q_mV}{4}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
Thermal Load/Forcing-Vector (regional heat sources) - uniform source U $(\frac{W}{m^3})$ ○ Q_{r_U} - point source P (W) ○ Q_{r_P} ○ (x_P, y_P, z_P, t_P)	$\{R_{Qr}\} = \{R_{Qr_U}\} \cup \{R_{Qr_P}\}$ $\{R_{Qr_U}\} = \frac{Q_{r_U}V}{4}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ $\{R_{Qr_P}\} = Q_{r_P}\begin{bmatrix} N_i \\ N_j \\ N_k \\ N_m \end{bmatrix}_{(x_P, y_P, z_P)}$
Thermal Load/Forcing-Vector (Temperature boundary condition)	$\{R_T\} = \{R_{T_{jkm}}\} \cup \{R_{T_{ikm}}\} \cup \{R_{T_{ijm}}\} \cup \{R_{T_{ijk}}\}$

	$\left\{R_{T_{jkm}}\right\} = -\frac{q_n A_{jkm}}{3}\begin{bmatrix}0\\1\\1\\1\end{bmatrix}$, $\left\{R_{T_{ikm}}\right\} = -\frac{q_n A_{ikm}}{3}\begin{bmatrix}1\\0\\1\\1\end{bmatrix}$, $\left\{R_{T_{ijm}}\right\} = -\frac{q_n A_{ijm}}{3}\begin{bmatrix}1\\1\\0\\1\end{bmatrix}$, $\left\{R_{T_{ijk}}\right\} = -\frac{q_n A_{ijk}}{3}\begin{bmatrix}1\\1\\1\\0\end{bmatrix}$
Thermal Load/Forcing-Vector (Heat Flux boundary condition)	$\{R_{qf}\} = \left\{R_{qf_{jkm}}\right\} \cup \left\{R_{qf_{ikm}}\right\} \cup \left\{R_{qf_{ijm}}\right\} \cup \left\{R_{qf_{ijk}}\right\}$ $\left\{R_{qf_{jkm}}\right\} = \frac{q_b A_{jkm}}{3}\begin{bmatrix}0\\1\\1\\1\end{bmatrix}$, $\left\{R_{qf_{ikm}}\right\} = \frac{q_b A_{ikm}}{3}\begin{bmatrix}1\\0\\1\\1\end{bmatrix}$, $\left\{R_{qf_{ijm}}\right\} = \frac{q_b A_{ijm}}{3}\begin{bmatrix}1\\1\\0\\1\end{bmatrix}$, $\left\{R_{qf_{ijk}}\right\} = \frac{q_b A_{ijk}}{3}\begin{bmatrix}1\\1\\1\\0\end{bmatrix}$

Note that the temperature-dependent thermal parameters in this research project are only the tissue specific heat capacity C_{ti} (see section 2.9.4), and the tissue thermal conductivity k_{ti} (see section 2.9.6).

3.2.7 Global Matrix Assembly

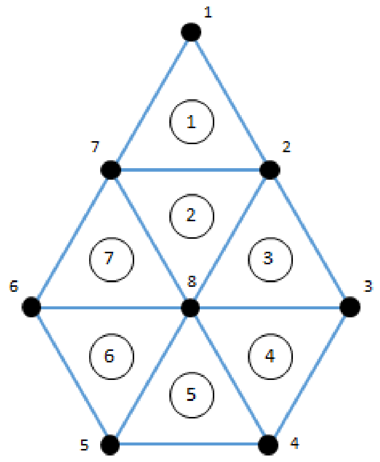


Figure 3-4: Two-Dimensional Example Domain for Global Matrix Assembly

Assume a simple two-dimensional case, shown in Figure 3-4, where the domain is discretized into a total of 7 constant-strain (linear) triangular elements connecting 8 nodes. By fully evaluating the elemental stiffness matrix $[K]$ as shown in Equation (3.65) for any of these elements, the typical elemental stiffness matrix will be of the form of:

$$[K_e] = \begin{bmatrix} \alpha_{ii} & \alpha_{ij} & \alpha_{ik} \\ \alpha_{ji} & \alpha_{jj} & \alpha_{jk} \\ \alpha_{ki} & \alpha_{kj} & \alpha_{kk} \end{bmatrix} \tag{3.70}$$

such that:

$$[K_{e1}] = \begin{bmatrix} a_{11} & a_{17} & a_{12} \\ a_{71} & a_{77} & a_{72} \\ a_{21} & a_{27} & a_{22} \end{bmatrix}, [K_{e2}] = \begin{bmatrix} b_{77} & b_{78} & b_{72} \\ b_{87} & b_{88} & b_{82} \\ b_{27} & b_{28} & b_{22} \end{bmatrix} \ldots \text{ etc.} \tag{3.71}$$

The global stiffness matrix $[K]$ dimensions will be $n \times n$ where n is equal to the global sum of DoF of each node. Since there are 8 nodes, and each has only 1 DoF (temperature being scalar), the global matrix will be of dimensions: $(8 \times 1) \times (8 \times 1)$. All elements of the global matrix are initialized to 0.

$$[K] = \begin{array}{c} \left[\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \\ \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \end{array} \tag{3.72}$$

The stiffness matrix elements of $[K_{e1}]$ are then added into the global stiffness matrix accordingly.

$$[K]=\begin{array}{c|c}\begin{bmatrix}a_{11}&a_{12}&0&0&0&0&a_{17}&0\\a_{21}&a_{22}&0&0&0&0&a_{27}&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\a_{71}&a_{72}&0&0&0&0&a_{77}&0\\0&0&0&0&0&0&0&0\end{bmatrix}&\begin{array}{c}1\\2\\3\\4\\5\\6\\7\\8\end{array}\\ \begin{array}{cccccccc}1&2&3&4&5&6&7&8\end{array}&\end{array}\tag{3.73}$$

Then the stiffness matrix elements of $[K_{e2}]$ are added.

$$[K]=\begin{array}{c|c}\begin{bmatrix}a_{11}&a_{12}&0&0&0&0&a_{17}&0\\a_{21}&a_{22}+b_{22}&0&0&0&0&a_{27}+b_{27}&b_{28}\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\a_{71}&a_{72}+b_{72}&0&0&0&0&a_{77}+b_{77}&b_{78}\\0&b_{82}&0&0&0&0&b_{87}&b_{88}\end{bmatrix}&\begin{array}{c}1\\2\\3\\4\\5\\6\\7\\8\end{array}\\ \begin{array}{cccccccc}1&2&3&4&5&6&7&8\end{array}&\end{array}\tag{3.74}$$

The procedure continues until all of the elements from all element stiffness matrices have been added into the global stiffness matrix.

$$[K]=\begin{bmatrix}a_{11}&a_{12}&0&0&0&0&a_{17}&0\\a_{21}&a_{22}+b_{22}+c_{22}&c_{23}&0&0&0&a_{27}+b_{27}&b_{28}+c_{28}\\0&c_{32}&c_{33}+d_{33}&d_{34}&0&0&0&c_{38}+d_{38}\\0&0&d_{43}&d_{44}+e_{44}&e_{45}&0&0&d_{48}+e_{48}\\0&0&0&e_{54}&e_{55}+f_{55}&f_{56}&0&e_{58}+f_{58}\\0&0&0&0&f_{65}&f_{66}+g_{66}&g_{67}&f_{68}+g_{68}\\a_{71}&a_{72}+b_{72}&0&0&0&g_{76}&a_{77}+b_{77}+g_{77}&b_{78}+g_{78}\\0&b_{82}+c_{82}&c_{83}+d_{83}&d_{84}+e_{84}&e_{85}+f_{85}&f_{86}+g_{86}&b_{87}+g_{87}&\theta_{88}\end{bmatrix}\tag{3.75}$$

Where: $\theta_{88}=b_{88}+c_{88}+d_{88}+e_{88}+f_{88}+g_{88}$

A similar procedure is utilized for assembling the global capacitance matrix $[C]$.

For the global forcing vector $\{R\}$, the element forcing vectors are evaluated in the form of:

$$\{R_e\}=\begin{Bmatrix}\beta_i\\\beta_j\\\beta_k\end{Bmatrix}.\tag{3.76}$$

The global forcing vector is developed for this case.

$$\{R\}=\begin{array}{cc}\begin{Bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{Bmatrix}&\begin{array}{c}1\\2\\3\\4\\5\\6\\7\\8\end{array}\end{array}\tag{3.77}$$

The forcing vector elements of $[R_{e1}]$ are then added into the global stiffness matrix accordingly.

$$\{R\} = \begin{array}{cc} \left\{\begin{matrix} h_1 \\ h_2 \\ 0 \\ 0 \\ 0 \\ 0 \\ h_7 \\ 0 \end{matrix}\right\} & \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} \end{array} \tag{3.78}$$

Then the forcing vector elements of $[R_{e2}]$ are added.

$$\{R\} = \begin{array}{cc} \left\{\begin{matrix} h_1 \\ h_2 + i_2 \\ 0 \\ 0 \\ 0 \\ 0 \\ h_7 + i_7 \\ i_8 \end{matrix}\right\} & \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} \end{array} \tag{3.79}$$

The procedure continues until all of the elements from all element forcing vectors have been added into the global forcing vector.

$$\{R\} = \left\{\begin{matrix} h_1 \\ h_2 + i_2 + j_2 \\ j_3 + k_3 \\ k_4 + l_4 \\ l_5 + m_5 \\ m_6 + n_6 \\ h_7 + i_7 + n_7 \\ i_8 + j_8 + k_8 + l_8 + m_8 + n_8 \end{matrix}\right\} \tag{3.80}$$

This procedure is utilized with tetrahedral elements in three-dimensions, simply by instead using:

$$[K_e] = \begin{bmatrix} \alpha_{ii} & \alpha_{ij} & \alpha_{ik} & \alpha_{im} \\ \alpha_{ji} & \alpha_{jj} & \alpha_{jk} & \alpha_{jm} \\ \alpha_{ki} & \alpha_{kj} & \alpha_{kk} & \alpha_{km} \\ \alpha_{mi} & \alpha_{mj} & \alpha_{mk} & \alpha_{mm} \end{bmatrix} \tag{3.81}$$

and:

$$\{R_e\} = \left\{\begin{matrix} \beta_i \\ \beta_j \\ \beta_k \\ \beta_m \end{matrix}\right\}. \tag{3.82}$$

3.3 Equation Parameters

3.3.1 Temperature-Independent Parameters

The temperature-independent parameters used in the simulation for bioheat transfer and thermal damage are as follows:

Table 3.8: Temperature-Independent Parameters

Symbol	Description	Value	Units	Section Determined	Comments
ρ_{ti}	Tissue density	1079	$\frac{kg}{m^3}$	2.9.2	
C_{ti}	Tissue specific heat capacity	3540	$\frac{J}{kg \cdot °K}$	2.9.3	Use 3405.0 for case comparing temperature field from temperature-dependent and temperature-independent C_{ti}.
k_{ti}	Tissue thermal conductivity	0.52	$\frac{W}{m \cdot °K}$	2.9.5	Use 0.4604 for case comparing temperature field from temperature-dependent and temperature-independent k_{ti}.
W_{bl}	Blood perfusion rate	17.502	$\frac{kg}{m^3 \cdot s}$	2.9.9	
C_{bl}	Blood specific heat capacity	3617	$\frac{J}{kg \cdot °K}$	2.9.8	
T_{art}	Arterial blood temperature	37	$°C$	2.9.10	
Q_m	Metabolic heat generation rate	11232	$\frac{W}{m^3}$	2.9.7	
Q_r	Regional source heat generation rate	-	$\frac{W}{m^3}$	-	Regional heat source is instead replaced by fixed nodal Dirichlet source temperature T_r (below).
T_r	Source temperature	60	$°C$	2.5.3	
E_a	Activation energy	3.513×10^5	$\frac{J}{mol}$	2.7.2	(Liver)
A	Frequency factor	3.18×10^{55}	$\frac{1}{s}$	2.7.2	(Liver)
ν	Poisson's ratio	0.45	-	2.9.12	
α	Coefficient of thermal expansion	1×10^{-4}	$°C^{-1}$	2.9.11	

Furthermore, for temperature-independent linearly elastic stress-strain relations, the Young's Modulus of liver tissue E is derived from Table 3.10 in section 3.3.3 as $E = \frac{0.061}{0.4} = 152.5$ kPa from data at 37 °C. Nonlinearly elastic stress-strain simply uses Equation (3.85) and coefficients for the 37 °C curve from Table 3.13.

3.3.2 Temperature-Dependent Parameters

Table 3.9: Temperature-Dependent Parameters

Symbol	Description	Value	Units	Section Determined
C_{ti}	Temperature-dependent tissue specific heat capacity	Obtain from linear interpolation of Table 2.7 data with Equation (2.7).	$\frac{J}{kg \cdot {}^{\circ}K}$	2.9.4
k_{ti}	Temperature-dependent tissue thermal conductivity	Obtained from Equation (2.8).	$\frac{W}{m \cdot {}^{\circ}K}$	2.9.6

Furthermore, for temperature-dependent nonlinearly elastic stress-strain relations, Equation (3.85) is employed. When a temperature and strain are known, the stress is derived from the stress-strain curve (Equation (3.85)) by coefficients for the upper and lower bounding temperature stress-strain curves (coefficients in Table 3.13), and subsequently interpolated between in a similar manner to (2.7).

3.3.3 Temperature-Dependent Stress-Strain Relationship

Referring to Figure 2-14, the temperature-dependence of the stress-strain relationship can be observed (strain ε is equivalent to the extension ratio λ minus 1). While the quantitative data is derived from pig ear skin, it is the most comprehensive source of temperature-dependent stress-strain relationships surveyed in literature for this research project.

Since stress is a function of both strain and temperature, a multivariable polynomial could be constructed using polynomial multiple regression. The resultant polynomial describes a surface that best fits the following tabulated data in Table 3.10; derived from Xu and Lu (2009) quantitative data shown in Figure 2-14.

Table 3.10: Temperature-Dependent Stress-Strain Relationship Data

T = 37 °C		T = 45 °C		T = 50 °C		T = 60 °C		T = 70 °C		T = 80 °C	
ε	σ (MPa)	ε	σ (MPa)	ε	σ (MPa)	ε	σ (MPa)	ε	σ (MPa)	ε	σ (MPa)
0	0.000	**0**	0.000	**0**	0.000	**0**	0.000	**0**	0.000	**0**	0.000
0.2	0.026	**0.2**	0.0175	**0.2**	0.0175	**0.2**	0.013	**0.2**	0.013	**0.2**	0.013
0.4	0.061	**0.4**	0.044	**0.4**	0.044	**0.4**	0.031	**0.4**	0.031	**0.4**	0.031
0.5	0.118	**0.5**	0.079	**0.5**	0.079	**0.5**	0.048	**0.5**	0.048	**0.5**	0.048
0.6	0.281	**0.6**	0.171	**0.6**	0.140	**0.6**	0.075	**0.6**	0.075	**0.6**	0.075
0.7	0.579	**0.7**	0.386	**0.7**	0.289	**0.7**	0.122	**0.7**	0.114	**0.7**	0.101
0.8	1.000	**0.8**	0.693	**0.8**	0.522	**0.8**	0.206	**0.8**	0.167	**0.8**	0.140

By using polynomial multiple regression, the derived fourth-order polynomial function describing the temperature-dependent stress-strain relationship for the data in Table 3.10 is:

$$\begin{aligned}\sigma = c_1T^4 + c_2T^3\varepsilon + c_3T^2\varepsilon^2 + c_4T\varepsilon^3 + c_5\varepsilon^4 + c_6T^3 + c_7T^2\varepsilon + c_8T\varepsilon^2 \\ + c_9\varepsilon^3 + c_{10}T^2 + c_{11}T\varepsilon + c_{12}\varepsilon^2 + c_{13}T + c_{14}\varepsilon + c_{15}\end{aligned} \tag{3.83}$$

where the coefficients are:

Table 3.11: Coefficients for Equation (3.83)

c_1	-2.910E-07	c_9	7.006E+00
c_2	-1.155E-07	c_{10}	-5.636E-03
c_3	1.930E-03	c_{11}	8.122E-02
c_4	-1.198E-01	c_{12}	1.045E+00
c_5	1.267E+00	c_{13}	2.030E-01
c_6	6.720eE-05	c_{14}	-1.644E+00
c_7	-8.076E-04	c_{15}	-2.647E+00
c_8	-1.434E-01		

Unfortunately, the polynomial does not fit the data well at low strains, and higher orders (fifth and sixth orders) do not improve this fit significantly at low strains.

For example, at a temperature of 50 °C and a strain of 0.6, Equation (3.83) produces a stress of 0.138 MPa which closely matches the tabulated value in Table 3.10, with an error of ~1%. However, at a low strain of 0.2 at the same temperature, the same equation produces a stress of 0.027 MPa; an error of greater than 50% when compared to the expected value of 0.0175 MPa. This is obviously not suitable for this application since only high strains will yield results remotely accurate to the data.

An alternative then, is to model best-fit polynomials for the stress-strain relationship at each temperature using polynomial regression analysis of the tabulated data, and employ interpolation between the obtained stresses to compute the stress at a given temperature and strain.

A fourth-order polynomial is modelled for each stress-strain curve, of the form:

$$\sigma = c_1\varepsilon^4 + c_2\varepsilon^3 + c_3\varepsilon^2 + c_4\varepsilon + c_5 \tag{3.84}$$

where the coefficients are:

Table 3.12: Coefficients for Equation (3.84)

T (°C)	c_1	c_2	c_3
37	7.696E-01	3.833E+00	-2.640E+00
45	2.681E+00	-4.134E-01	-5.599E-01
50	3.207E+00	-2.361E+00	6.056E-01
60	1.285E+00	-1.198E+00	4.416E-01
70	-2.793E-02	4.934E-01	-2.309E-01
80	-3.100E-01	7.022E-01	-2.566E-01
T (°C)	c_4	c_5	**Coefficient of Determination** (r^2)
37	5.189E-01	-5.890E-04	0.9995
45	2.109E-01	-4.888E-04	0.9991
50	3.898E-02	-1.265E-04	0.9996
60	1.255E-02	5.052E-05	1.0000
70	9.209E-02	-1.201E-05	1.0000
80	8.884E-02	5.929E-05	0.9993

By clamping the input temperature T to the range of 37°C to 80°C, then determining the interpolant upon which it lies, the stress can then be interpolated, as is detailed in section 2.9.4.

Unfortunately, even then, the fourth-order polynomials exhibit poor shape at strains below 0.4, as can be seen in Figure 3-5.

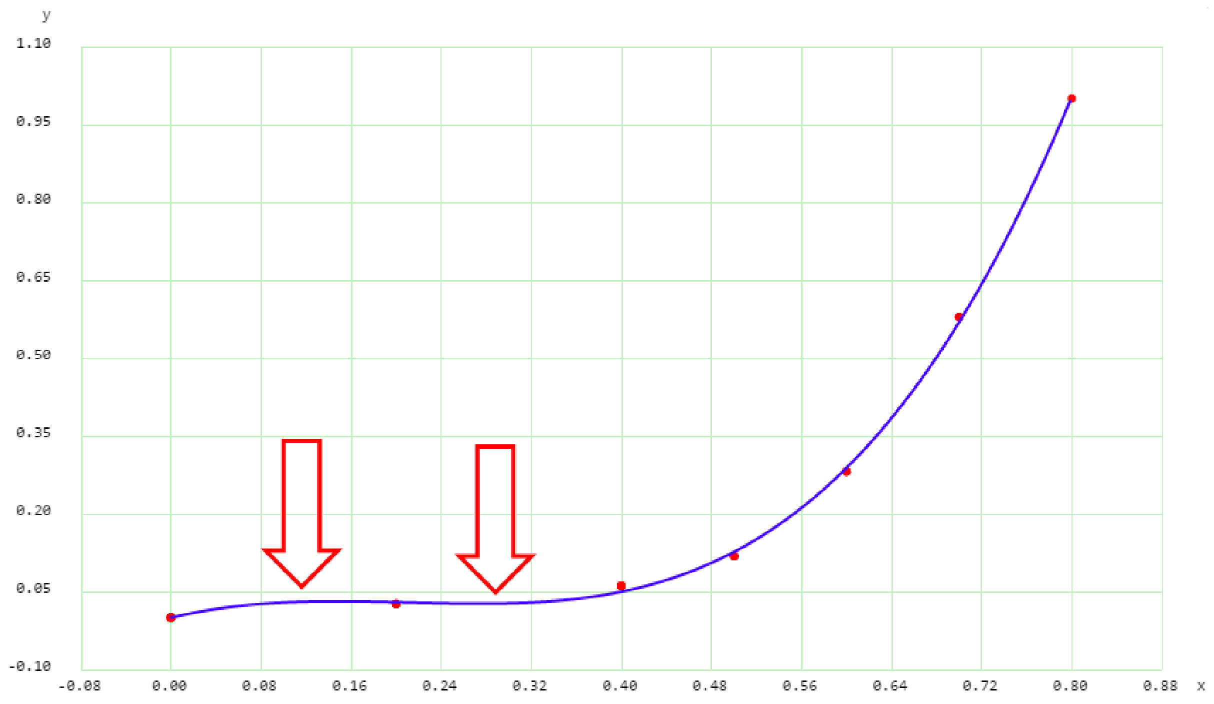


Figure 3-5: Fourth-Order Polynomial of Non-Linear Stress-Strain at 37 °C

Thus the solution here is to model the best fit polynomials for low strains up to 0.4, then thereafter use the modelled fourth-order polynomials for approximating stresses at higher strains. It is assumed that thermally induced strains would be much smaller than those which would be mechanically induced by surgeons, applying the operational instrument forces described in section 2.5.3.

By using the data in Table 3.10 for smaller strains of 0.0, 0.2, and 0.4, a second-order polynomial can be modelled for each stress-strain curve.

$$\sigma = c_1\varepsilon^2 + c_2\varepsilon + c_3 \tag{3.85}$$

Table 3.13: Coefficients for Equation (3.85)

T (°C)	c_1	c_2	c_3	**Coefficient of Determination (r^2)**
37	1.125E-01	1.075E-01	-1.648E-17	1.0000
45	1.125E-01	6.500E-02	-9.541E-18	1.0000
50	1.125E-01	6.500E-02	-9.541E-18	1.0000
60	6.250E-02	5.250E-02	-1.659E-17	1.0000
70	6.250E-02	5.250E-02	-1.659E-17	1.0000
80	6.250E-02	5.250E-02	-1.659E-17	1.0000

Therefore, the temperature-dependent stress can be determined from the temperature and strain by utilizing a combination of polynomial functions produced by polynomial regression analysis, and linear interpolation (similarly, see Equation (2.7)).

3.4 Simulation Parameters

3.4.1 Boundary Condition Type

The boundary conditions employed on the mesh for transient bioheat transfer simulation are those discussed at the end of section 3.2.4 and in Equation (3.68). Insulated (Neumann) thermal boundary conditions (at the mesh edge/surface) are the simplest to implement programmatically.

To simulate the operational instrument heat source, a single node (typically the most central to the mesh) is applied with a fixed temperature (Dirichlet) boundary condition, as defined in section 2.5.3.

When thermal strains are calculated from the temperature field (via third-party software; discussed in section 3.5.2), fixed support boundary conditions are applied to the mesh edge/surface.

Since the organic liver geometry is not represented by the simulated meshes in this research project, but rather a small block of homogeneous isotropic tissue is represented, it is adequate to apply insulated, fixed support boundary conditions.

As an added note, thermal damage in the simulation software is capped at the point of necrosis of $\Omega = 4.6$ (see section 2.7.2).

3.4.2 Time Step & Mesh Element Size

Roland W. Lewis, Perumal Nithiarasu and Seetharamu (2004) state that the stable time step for transient thermal analysis is defined as:

$$\Delta t \leq \frac{l^2}{b\alpha} \tag{3.86}$$

where l is the element size (or a minimum/characteristic length), b is an adjustable factor and α is the thermal diffusivity, defined as:

$$\alpha = \frac{k_{ti}}{\rho_{ti} c_{ti}}. \tag{3.87}$$

ANSYS, as cited in Stout and Billings (2002) recommends a b value of 4, however Stout and Billings (2002) also lists trialling values of b ranging from 1 to 1000, depending on the coarseness/fineness of the mesh in use. Higher values of b are recommended for finer meshes, as they reduce the time step.

In order to calculate an adequate time step, the thermal diffusivity can be determined as:

$$\alpha = \frac{k_{ti}}{\rho_{ti} c_{ti}} = \frac{0.4604}{1079 \times 3405} = 1.253 \times 10^{-7} \ \frac{m^2}{s}. \tag{3.88}$$

By taking the factor recommended by ANSYS ($b = 4$) and the minimum side length of a tetrahedral element in the mesh taken as 1 mm, then:

$$\Delta t \leq \frac{l^2}{b\alpha} \leq \frac{0.001^2}{4 \times 1.253 \times 10^{-7}} \leq 1.995 \ s. \tag{3.89}$$

Obviously, this limit for a time step size is huge, so with a great deal of experimentation and adjustment of time step size in the simulation, it was found that stability was present for meshes of 1 mm element size for $\Delta t = 0.04$, which is the time step equal to 1/25th of a second, i.e. real-time (see section 2.10). A mesh element size of 1mm or finer was required in order to produce a very smooth temperature distribution for lower-level heat sources (54 °C), since the effects of blood perfusion in bioheat transfer cause a rapid cooling of the tissue within a very small distance from the heat source. This also explains the need for RFA procedures to employ instruments such as the needle shown in Figure 2-1, with expanding needles for increasing the heated coagulation zone in the tissue. Thus a b value of 200 worked for a mesh element size of 1mm for a real-time suitable time step.

Some of the results analysed in this research project show a very fine mesh element size (0.1 – 0.5 mm) for the sake of a very smooth temperature distribution. Such sizes are not adequate for real-time, as the time step applied for these was 0.002 seconds. They are primarily for the purpose of increasing the level of detail available for analysis of the tissue bio-thermomechanics. To achieve real-time for such element sizes, the mesh would need to be simple enough to permit 500+ iterations per second; i.e. very few nodes, elements and global DoF.

Note that while the time step is calculated based on transient thermal behaviour, it is implemented as the general time step for the overall simulation. Temperatures are updated based on this time step, and the resultant temperature field is used to subsequently determine thermal damage and thermal stress-strain. Only the temperature distribution history is used for progressive time step-based calculations (using the explicit method, as detailed in the methodology in section 3.2.3).

3.4.3 Mesh Resolution

Mesh resolutions employed for real-time simulations for development during the research project were typically 11×11×11 nodes, i.e. 1,331 nodes (also global DoF; 1 per node – temperature), 10×10×10 cubes subdivided into 6 tetrahedra each, for a total of 6,000 linear tetrahedral elements. Figure 4-22 demonstrates that it is possible, the way this simulation program was implemented, to achieve approximately 22,000 nodes/global DoF (28×28×28 nodes) for real-time bioheat transfer and thermal damage simulation.

Mesh resolutions employed for more detailed analyses were 21×21×21 nodes, i.e. 9,261 nodes (also global DoF), 20×20×20 cubes subdivided into 6 tetrahedra each, for a total of 48,000 linear tetrahedral elements.

Due to the limitation of third-party software used for thermal strain calculations to 1,300 nodes maximum (see section 3.5.2), a mesh resolution of 11×11×9 nodes was used to firstly simulate a temperature distribution which was used in the third-party software.

3.4.4 Margin of Error

For the Gauss-Seidel solution of the global matrices, each successive iteration should cause the calculated field value (i.e. temperature) to change by smaller and smaller increments, as it ideally converges towards a particular solution. Programmatically, the number of iterations should be limited to some maximum number, in the case that convergence is not ever reached, and thus avoid an infinite loop. However, some margin of error may be specified, such that the iterative loop may be terminated prior to reaching the maximum number of iterations, if some satisfactory level of convergence has already been reached. Terminating the iterative loop at satisfactory convergence results in a saving of computation time/cost.

Typically for simulations performed for this research project, the margin of error specified was 0.001.

Smaller margins of error allow greater accuracy of computed results, however can require a much larger number of solver iterations in order to achieve such levels of accuracy. If the margin of error is too large, convergence may not occur due to accrued error during successive simulation iterations. If the margin of error is too small, the maximum number of solver iterations may be reached (and the solver loop terminated) before reaching suitable convergence. For the sake of real-time computation, accuracy and the number of solver iterations (Gauss-Seidel) must be carefully weighed and some optimal margin of error established. See section 3.6.7 for justification for the choice of margin of error (of 0.001) for simulations undertaken in this research project.

3.5 Simulation Development Tools & Resources

3.5.1 Hardware

The computer used for simulation development and execution is a Metabox Prime P177SM laptop model, with the following specifications:

- Processor: Intel® Core™ i7-4700MQ CPU @ 2.40GHz (4 physical CPUs, hyper-threaded to 8)
- Memory: 16384 MB DDR3 RAM
- Integrated Graphics: Intel® HD Graphics 4600
- Dedicated Graphics: NVIDIA GeForce GTX 770M
    - Dedicated video memory: 3072 MB GDDR5
    - Memory data rate: 4008 MHz
    - Graphics clock: 705 MHz
    - CUDA cores: 960
    - Memory Interface Width: 192 bit
- Mini IEEE-1394a FireWire® port (necessary if using Phantom Omni® haptic devices; unused for this research project)
- Windows 7 Professional operating system

This model was originally selected with the intent of compatibility with current commercial haptic devices. Implementation of haptic devices in this research project is no longer under consideration however. The focus of real-time simulation has shifted to visual real-time simulation output/feedback; a simulated rate of 25 Hz; as selected and justified in section 2.10.

3.5.2 Programming Languages and APIs

C++ is high-level programming language, in contrast to low-level programming languages such as Assembly Language. C++ is also a compiled language, compiling into machine code executables prior to execution; typically achieving greater speed than runtime languages (Java, C#). C++ is a multi-industry standard for procedural or object-oriented professional and entertainment software and application development. It has outstanding documentation and support, and is a frequent language for Application Programming Interfaces (APIs) involved with interfacing with hardware such as GPUs. It is due to its compatibility with haptic instrument APIs such as the Phantom Omni® QuickHaptics™ toolkit, its Direct3D11 API by Microsoft® (for GPU interaction) and its compiled nature, that it has been selected as the primary development language for the simulation software in this research project.

Microsoft's WIN32 API for C++ enables development of window-based applications for Windows operating systems. This is a necessary component of developing Direct3D applications, as it is responsible for providing a window upon which Direct3D can output rendered visual data. Any necessary graphical user interface development for this project is achieved through this API.

Microsoft's DirectX® Direct3D11 API for C++ enables interfacing with graphics hardware, enabling GPGPU programming (via DirectX11® DirectCompute) and graphical display programming. It is well documented, however is only supported on Windows operating systems. Due to the popularity of Windows, Direct3D was selected (rather than OpenGL) for GPU interfacing, GPGPU programming and rendering for this research project. A minimum version; version 11, of DirectX® Direct3D is specifically required, since it supports Shader Model 5; a standard which implements the use of compute shaders; i.e. DirectCompute.

Microsoft's High Level Shader Language (HLSL) pairs with use of the Direct3D API. HLSL code is compiled in shader programs which are executed at different stages of the rendering/programmable pipeline on the GPU. It has a syntax similar to C++, however its feature set is more limited and specialized.

LISA is a finite element analysis package for Windows, which is used in this project for the exclusive purpose of calculating thermal strains from the temperature distribution calculated by the simulation designed in this research project. GPU accelerated finite element bioheat transfer analysis and thermal damage can be performed in real time in this project's simulation, however the current disconnect is that temperature distributions must be exported from the simulation and imported into LISA for further analysis to define thermal strains. The free version of LISA supports meshes of up to 1300 nodes, thus limiting the mesh resolution, as described in section 3.4.3.

3.6 Simulation Development

3.6.1 Program Architecture

The design of the research project simulation is based on Object-Oriented Design (OOD) principles (as opposed to procedural), such that classes of code allow instantiation of virtual objects with virtual properties, and can be arranged in a hierarchy of ownership/control. Development of the application stood at approximately 7000 total lines of original code upon retrieval of results for the research project. Some of this information is comprised of empty line spacing, comments and code documentation to improve readability, however the majority is directed to fulfil the application's core purpose. Some of this bulk can be reduced upon careful refactoring of the source code.

This research project's software simulation architectural hierarchy is simplified and displayed in Figure 3-6.

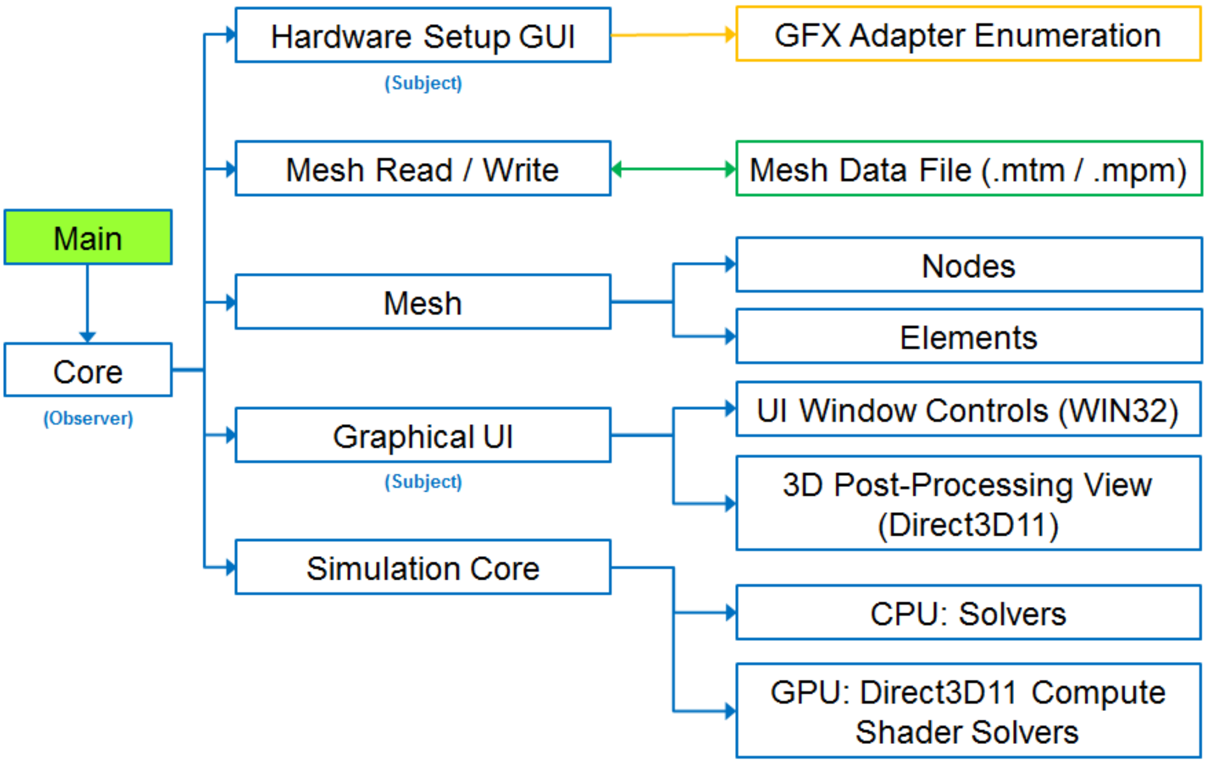


Figure 3-6: Simulation Program Architecture

The main entry point to the application is designated its own source file, also containing simple pre-processor directives such as:

```
#define GENERATE_MESH
```

which can be commented out (by inserting "//" at the beginning of the line) to disable mesh generation. This allows the application to run specifically to generate a custom mesh file, or specifically as a simulation application, loading in a mesh file from a specific file path/location.

The "Main" component creates an instance of "Core", which governs the overall initialization and execution behaviour of the simulation application. The hierarchy is structured such that all instantiated major subcomponents of a component must successfully initialize in order for the program to continue execution. For example, if the "D3DSimulate" class (instantiated by "SimulationCore") detects that the user's installed version of DirectX does not support the use of GPGPU compute shaders (DirectCompute), then "D3DSimulate" initialization fails, thus "SimulationCore" initialization fails, thus “Core” initialization fails, thus finally the application initialization fails in “Main”, and the user is notified of the specific problem (i.e. DirectX version incompatibility).

As stated, “Core” governs the general behaviour of the application, including the WIN32 thread loop, the hardware setup, mesh file generation, writing and reading/loading, contains the virtual mesh object data structure, and governs the behaviour of and interaction between the graphical user interface (GUI/UI) and the “SimulationCore” component. Ordered in this hierarchy, the GUI and “SimulationCore” cannot directly communicate between each other, so a programmatic design pattern called the “Observer Design Pattern” is implemented. Examining Figure 3-7, the two objects owned by Observer A cannot communicate directly to each other; any communication must flow through Observer A. Typically, an object cannot initiate communication with its owner, however this design pattern registers the owner as an “observer” to the owned objects, which are “subjects”. Each subject can then use a specific function to notify their observers (green arrows in Figure 3-7). Thus if a button or other control is clicked on the GUI, it notifies “Core”, which checks the “GUI” object to determine what has happened, and then proceeds to direct “SimulationCore” to execute a particular behaviour in response.

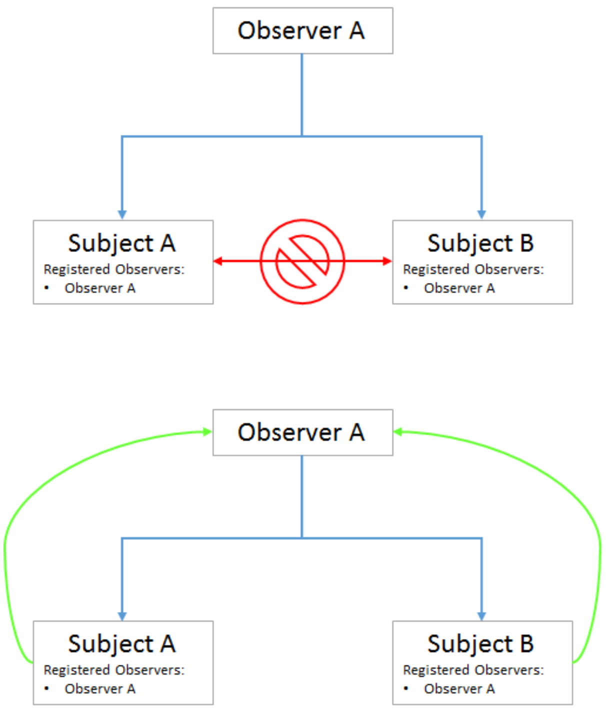


Figure 3-7: Observer Design Pattern

The Hardware Setup GUI component of the application launches first before any other initializations are completed, and provides the user the option to determine whether the CPU or a specific GPU will execute the simulation. This component enumerates the available graphics hardware on the computer via Direct3D; retrieving a description of each graphics adapter, and producing a list for the user from which a specific graphics adapter can be selected. The user confirms their selection of hardware settings by clicking a button to proceed. The Hardware Setup GUI (a subject) notifies “Core” (observer) of the selection, and “Core” retrieves the hardware settings from the Hardware Setup GUI, passing them as initialization parameters to the main GUI, as well as “SimulationCore”. These components are initialized conditionally on whether the designated mesh file has successfully loaded and been stored into a data structure. When these components have completed initialization, “Core” completes initialization, and the application is at a ready state to be interacted with by the user.

The mesh is loaded into the application from a mesh file specific to the application, and specified in the application parameters. A file path is initially passed to an input file stream object (C++ standard library) which iteratively reads each line of text in the mesh file, storing the line to a string (C++ standard library), feeding the string into an input string stream. From the line's input string stream, pieces of data in the line are fed to relevant data components in the application mesh data structure. An analysis of the mesh connectivity is subsequently performed on the loaded mesh, in order to determine the layout of the CSR vectors (shown in section 2.3.7). The application determines the locations of all potential element matrix contributions in the global matrices, and organizes the CSR vectors to represent all of these nonzero matrix elements.

The GUI consists of the main application window, which houses the main application menu, the display window and the control panel window. Furthermore, it instantiates "D3DDisplay"; the object in charge of rendering post-processing data to the display window. "D3DDisplay" implements two vital shaders for rendering: the vertex and pixel shaders. These interpolate mesh node positional data, determining their position on the display window, and colours, determined by colour gradient from mesh node field data values (i.e. temperature, thermal damage). The combination of "D3DDisplay" and the GUI supports display features which can be toggled in the application, such as wireframe rendering, edged face rendering, vertex rendering, toggling of colour gradients and camera behaviour using the mouse, such as rotation and zooming to examine the mesh. User interface controls are located in the main menu bar, as well as within tabs on the control panel; both of which are implemented with the WIN32 API.

The Simulation Core is responsible for finite element simulation using the mesh, on the hardware designated by the user; i.e. CPU/GPU. It instantiates "D3DSimulate", which controls designation of shader resources and execution of shaders on the GPU. "SimulationCore" houses functions for calculation of tetrahedral element volume, stiffness matrices, capacitance matrices and load/forcing vectors. It is responsible for element matrix calculations, global matrix assembly, and global matrix solution using the Gauss-Seidel method determined in section 2.3.2, all on the CPU.

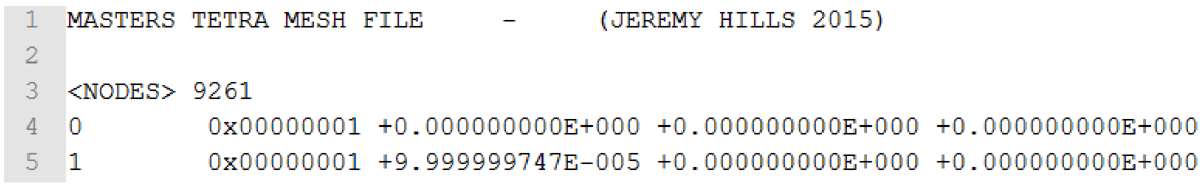
The simulation application consists of a multitude of other source files: 12 C++ source files (.cpp), 17 C++ header files (.h), 1 resource file (.rc) and 3 shader files (.hlsl); 33 files in total. Biological tissue and simulation parameters, timers for profiling, utility functions, observer design pattern source code, and inherited parent classes (i.e. "D3DBase" code is common to "D3DDisplay" and "D3DSimulate"). Extensive modification of the source code may see some reduction in total lines and source files as the overall application design is refined, although that is irrelevant to the scope of this research project.

3.6.2 Mesh Data Files

For the simulation application, a custom mesh file format was developed; a Masters Tetra Mesh, with file extension ".mtm". Each file contains the total number of nodes and tetrahedra comprising the mesh.

For each node, five pieces of data are recorded (Figure 3-8):

- Node ID: This is unique to each node, and assists the application to index nodes for access in reading/writing operations of nodal data in the application.
- Node flags: This is a hexadecimal number which is read into the application, converting it into an integer of 4 bytes. Each bit ("bit flag") facilitates representation of particular data regarding the node; i.e. whether or not it exists on the mesh boundary. This is simply a way of storing the equivalent of 32 boolean variables in the space of a few characters, which may be of greater use if the application is extended.
- x, y and z coordinate data. Each coordinate is stored in scientific notation, and signed whether positive or negative. Signing positive values maintains the alignment of the numerical data in the file when viewed.



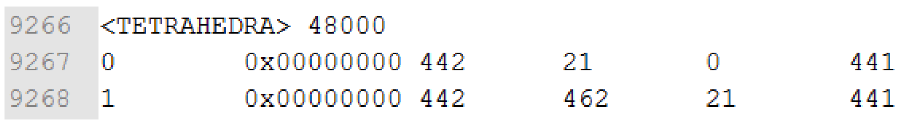
```
MASTERS TETRA MESH FILE     -     (JEREMY HILLS 2015)

<NODES> 9261
0         0x00000001 +0.000000000E+000 +0.000000000E+000 +0.000000000E+000
1         0x00000001 +9.999999747E-005 +0.000000000E+000 +0.000000000E+000
```

Figure 3-8: Masters Tetra Mesh File (Nodes)

For each tetrahedral element, six pieces of data are recorded (Figure 3-9):

- Tetrahedron ID: This is unique to each tetrahedron, and applies to indexing similarly to node IDs.
- Tetrahedron flags: Similar to node flags, though no current use applied for the simulation as it stands.
- *i*, *j*, *k* and *m* node IDs comprising the tetrahedral element. These IDs must be ordered according to the rules specified in section 3.1, otherwise not only can calculated volumes (and thus element matrices/vectors) risk being negative, but face normals for the post-processing display can be inverted, such that only back-facing polygons are rendered, and polygons closest to the user become seemingly invisible.



```
<TETRAHEDRA> 48000
0         0x00000000 442       21        0         441
1         0x00000000 442       462       21        441
```

Figure 3-9: Masters Tetra Mesh File (Tetrahedra)

The application determines when to switch modes from reading nodes to reading tetrahedra based on "headers". These include angled brackets that indicate the beginning of a new section in the file, and the heading between the angled brackets indicates to the application which section, based on a string to string comparison.

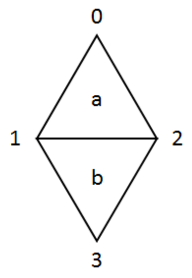
3.6.3 Shader Resources

In section 2.3.7, the technique of matrix compressed sparse row storage (CSR/CRS) is discussed. In order to index some data at a specific row/column location in the matrix, the row and column must be known in order to locate the desired data. For a large sparse matrix, this is an effective space and time-saving technique to employ. However, at the end of section 2.3.6, the difficulties inherent in parallelized operations (those of multiple simultaneous write operations to the same memory address) are discussed. Consider the fact that, as shown in section 3.2.7, at some locations in a global matrix or global forcing vector, multiple element contributions are applied. Should these contributions be applied simultaneously during shader execution, a majority of the written data is lost during multiple attempts to perform multiple writes at a multitude of specific memory addresses. Note that such a problem would occur, even if CSR was not employed.

The solution applied in this research project was to further analyse the mesh connectivity on loading/initialization to produce a data structure representative of a CSR matrix, however providing unique data addresses for the contributions from each mesh element. A "parallel gather" method (see Courtecuisse et al. (2010)) was combined with CSR to avoid potential write conflicts to resources stored in VRAM. This comprises an extra step in the GPU computation of the simulation on global matrix assembly, as all element contributions must be committed, and then a phase of summing all of these contributions and performing a single writing operation to each of the CSR elements. Consider the global matrix shown in the following example:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & 0 \\ a_{10} & a_{11}+b_{11} & a_{12}+b_{12} & b_{13} \\ a_{20} & a_{21}+b_{21} & a_{22}+b_{22} & b_{23} \\ 0 & b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix}$$

For the simple mesh of:



If the mesh elements from which contributions to the global matrix are applied, are indicated by the letters *a* and *b*, then implementing CSR along with a data structure for representing individual element contributions:

$$contributed\ element\ value = [a_{00}\quad a_{01}\quad a_{02}\quad a_{10}\quad a_{11}\quad b_{11}\quad a_{12}\quad b_{12}\quad b_{13}\quad a_{20}\quad a_{21}\quad b_{21}\quad a_{22}\quad b_{22}\quad b_{23}\quad b_{31}\quad b_{32}\quad b_{33}]^T$$

$$mesh\ element\ index = [a\quad a\quad a\quad a\quad a\quad b\quad a\quad b\quad b\quad a\quad a\quad b\quad a\quad b\quad b\quad b\quad b\quad b]^T$$

$$matrix\ element\ value = [m_{00}\quad m_{01}\quad m_{02}\quad m_{10}\quad m_{11}\quad m_{12}\quad m_{13}\quad m_{20}\quad m_{21}\quad m_{22}\quad m_{23}\quad m_{31}\quad m_{32}\quad m_{33}]^T$$

$$matrix\ element\ pointer = [0\quad 1\quad 2\quad 3\quad 4\quad 6\quad 8\quad 9\quad 10\quad 12\quad 14\quad 15\quad 16\quad 17]^T$$

$$column\ index = [0\quad 1\quad 2\quad 0\quad 1\quad 2\quad 3\quad 0\quad 1\quad 2\quad 3\quad 1\quad 2\quad 3]^T$$

$$row\ pointer = [0\quad 3\quad 7\quad 11]^T$$

Say we wish to store the contribution b_{12} to its appropriate address before summing with a_{12} (in order to avoid a write conflict). Knowing the ID of the element contributing it (*b*), the row and the column in which it should be located (row 1, column 2 when considering the matrix is zero-indexed), the following procedure is executed:

- ID = *b*.
- row = 1.
- column = 2.
- rp = *row pointer*[row] = 2.
- Compare column against *column index* array from *column index*[rp] until a match is found. Let the index at which the match occurs be "ci". Thus ci = 5.
- mep = *matrix element pointer*[ci] = 6.
- Compare ID against *mesh element index* array from *mesh element index*[mep] until a match is found. Let the index at which the match occurs be "mei". Thus mei = 7.
- The (zero-indexed) *contributed element value*[mei] = b_{12}.

Thus the address for element *b*'s contribution of b_{12} has been located, and the value can be written/read. This process applies for storing the element contributions, and for retrieving them to sum them for each global matrix element during assembly of the global matrix.

This technique may appear complicated and inefficient, however it outperformed attempts to execute interlocked addition functions for the same purpose (see section 2.3.6).

The data structures designed to hold the information from the above example for a given sparse global matrix in the application code are named as follows:

- GlobalCoeffMatrixPool
- GlobalCoeffMatrixData
- GlobalCoeffMatrixRow

Shown in Table 3.14 below, each element in the above three arrays contains different data types, and works similarly to the data held in the six arrays in the above example.

Table 3.14: Shader Resource - Example Comparison

Shader Resource	**Similar to:**
GlobalCoeffMatrixPool[i].value	*contributed element value*
GlobalCoeffMatrixPool[i].meshElementIndex	*mesh element index*
GlobalCoeffMatrixData[i].value	*matrix element value*
GlobalCoeffMatrixData[i].matrixElementPointer	*matrix element pointer*
GlobalCoeffMatrixData[i].columnIndex	*column index*
GlobalCoeffMatrixRow[i].rowPointer	*row pointer*

Shader structured buffer resources are created by first defining on the CPU, a data structure (16 byte aligned) to be propagated in an array. For example, a 16 byte aligned data structure which can be used by both GlobalCoeffMatrixPool and GlobalCoeffMatrixData looks like:

```
struct MData {
	float value;                        // 4 bytes
	unsigned int columnIndex;           // 4 bytes
	unsigned int meshElementIndex;      // 4 bytes
	unsigned int matrixElementPointer;  // 4 bytes
};
```

Obviously, elements in the GlobalCoeffMatrixPool array will not make use of “columnIndex” or “matrixElementPointer”, and elements in the GlobalCoeffMatrixData array will not make use of “meshElementIndex” (see Table 3.14). Nevertheless, the same data structure is able to be implemented to suit the interests of GlobalCoeffMatrixPool and GlobalCoeffMatrixData by simply ignoring unneeded variables. This seems like a much more expensive method of storage when compared to a simple CRS matrix, however it is still of the order $O(n)$ with regards to global DoF, as opposed to alternative means (see section 2.3.7).

The next step in creating a shader structured buffer resource (CPU side) is to declare a Direct3D11 Buffer pointer (pointers contain just the address to an object/block of memory, not the object data its self).

```
ID3D11Buffer*         m_pGCoeffMatrixPoolBuffer;
```

When this has been done, some initial data can be prepared in an array, to be stored to the buffer. Some or all of this data may be copied from other arrays/locations.

```
std::vector<MData> data(meshElementIndices.size());
for (unsigned int i = 0; i < data.size(); ++i) {
       data[i].value = 0.0f;
       data[i].columnIndex = 0;
       data[i].meshElementIndex = meshElementIndices[i];
       data[i].matrixElementPointer = 0;
}
```

A Direct3D11 subresource data structure is then prepared for storing the array data. This typically involves creating the empty subresource data structure, zeroing the memory that it occupies, and providing a pointer to the system memory address where the array of initial data resides.

```
D3D11_SUBRESOURCE_DATA initData = {};
ZeroMemory(&initData, sizeof(initData));
initData.pSysMem = &data[0];
```

Then a Direct3D11 buffer description structure must be created, where variables are set which detail the type and behaviour of the buffer. The *Usage* defines how the buffer may be used by the GPU and the CPU. D3D11_USAGE_DEFAULT specifies that the GPU may read from and write to the resource, however the CPU is not permitted access. CPU access permissions typically cause GPU usage of the resource to be slower. *ByteWidth* refers to the size in bytes, occupied by the buffer being created. *BindFlags* specifies how the resource will be bound to the pipeline. The resource could specify that it will be bound to the pipeline as a vertex buffer for example, however in this case D3D11_BIND_UNORDERED_ACCESS means that the resource will be read from and written to. *CPUAccessFlags* specifies what permissions (read/write) the CPU has with the buffer. *StructureByteStride* represents the size in bytes, of a single element stored in the buffer’s array. *MiscFlags* specifies any other miscellaneous properties of the buffer. In this case, it is a structured buffer.

```
D3D11_BUFFER_DESC desc = {};
ZeroMemory(&desc, sizeof(desc));
desc.Usage = D3D11_USAGE_DEFAULT;
desc.ByteWidth = sizeof(MData)* data.size();
desc.BindFlags = D3D11_BIND_UNORDERED_ACCESS;
desc.CPUAccessFlags = 0;
desc.StructureByteStride = sizeof(MData);
desc.MiscFlags = D3D11_RESOURCE_MISC_BUFFER_STRUCTURED;
```

Finally, Direct3D11 can attempt to create the buffer from the defined initial data, the buffer description and the buffer pointer.

```
m_pD3dDevice->CreateBuffer(&desc, &initData, &m_pGCoeffMatrixPoolBuffer);
```

Direct3D functions that create Direct3D objects should check to determine whether creation of the object has been successful.

Next on the CPU side, a resource view must be created to tell Direct3D and the shader exactly how the resource should be used. Multiple resource views can be created for a single resource, thus a single buffer might be written to in one shader, and then be supplied to a different shader as a read-only resource.

```
ID3D11UnorderedAccessView*                  m_pGCoeffMatrixPoolUAV;
```

In this case, an unordered access view (UAV) should be created. A UAV description structure is created, its memory zeroed, and some parameters defined. Setting the *Format* to DXGI_FORMAT_UNKNOWN signals that the buffer structure is based on a user-defined type (i.e. *MData*). The *ViewDimension* specifies how the resource will be accessed. *Buffer.FirstElement* indicates the zero-based index of the first buffer element to be accessed. *Buffer.Flags* indicates additional variations on the buffer type (not used here). *NumElements* indicates the number of elements in the resource.

```
D3D11_UNORDERED_ACCESS_VIEW_DESC uavDesc = {};
ZeroMemory(&uavDesc, sizeof(uavDesc));
uavDesc.Format = DXGI_FORMAT_UNKNOWN;
uavDesc.ViewDimension = D3D11_UAV_DIMENSION_BUFFER;
uavDesc.Buffer.FirstElement = 0;
uavDesc.Buffer.Flags = 0;
uavDesc.Buffer.NumElements = numElements;
```

Finally, the resource view can be created from the buffer, the resource view description and the resource view pointer (in this case the UAV pointer).

```
m_pD3dDevice->CreateUnorderedAccessView(m_pGCoeffMatrixPoolBuffer, &uavDesc,
                                                              &m_pGCoeffMatrixPoolUAV);
```

When the resource view has been created, it must be assigned (along with other resource views for the same or other buffers) to specific registers on the GPU.

```
std::vector<ID3D11UnorderedAccessView*> uavs = {m_pGCoeffMatrixPoolUAV};
m_pD3dImmediateContext->CSSetUnorderedAccessViews(0, uavs.size(), &uavs[0], NULL);
```

This completes the resource creation procedure on the CPU side. On the GPU side, the buffer element data structure must first be defined in HLSL in the shader.

```
struct MData {
       float value;                     // 4 bytes
       uint columnIndex;                // 4 bytes
       uint meshElementIndex;           // 4 bytes
       uint matrixElementPointer;       // 4 bytes
};
```

The next and final step in the shader, is to define the buffer.

```
RWStructuredBuffer<MData>          GlobalCoeffMatrixPool        : register(u0);
```

The buffer can then be accessed in the shader.

3.6.4 Shaders

The implementation of compute shaders in the simulation application involves several steps on the CPU side.

1. Direct3D device and context initialization.

The ID3D11Device interface is used for allocating resources and checking feature support.

The ID3D11DeviceContext is used for binding resources to the rendering pipeline, for changing rendering parameters, and for issuing rendering commands.

Both a device and a device context must be initialized in Direct3D first before further commands are issued. Both must be initialized from a known IDXGIAdapter. It is this adapter that is defined in the application by the user, thanks to the hardware setup phase.

```
D3D_FEATURE_LEVEL featureLevels[] = { D3D_FEATURE_LEVEL_11_1,
       D3D_FEATURE_LEVEL_11_0 };

D3D11CreateDevice(
       m_pAdapter,
       D3D_DRIVER_TYPE_UNKNOWN,
       NULL,
       createDeviceFlags,
       featureLevels,
       _countof(featureLevels),
       D3D11_SDK_VERSION,
       &m_pD3dDevice,
       NULL,
       &m_pD3dImmediateContext
       );
```

2. Verification of supported compute shader level.

As discussed in section 3.5.2, Direct3D11 is required in order to facilitate the use of compute shaders. If the initialized ID3D11Device feature level is older than D3D_FEATURE_LEVEL_11_0, the application’s initialization fails.

```
if (m_pD3dDevice->GetFeatureLevel() < D3D_FEATURE_LEVEL_11_0) {

       // ...

       return false;
}
```

3. Compilation of shader(s).

When the compute shader code has been written in HLSL, the .hlsl source file must be compiled through Direct3D in order to be executable on graphics hardware. The Direct3D function D3DCompileFromFile is responsible for this task. The entry point function for a shader must be specified also; similarly to the entry point function of a C++ application (i.e. the function titled “main”).

```
HRESULT result = D3DCompileFromFile(
              srcFile,
              NULL,
              D3D_COMPILE_STANDARD_FILE_INCLUDE,
              entryPoint,
              profile,
              flags,
              0,
              &shaderBlob,
              &errorBlob
              );
```

When the shader file has been compiled into a “blob” of data, the shader its self can then be created by using the Direct3D function CreateComputeShader, from the shader file data “blob” and an ID3D11ComputeShader pointer.

```
ID3D11ComputeShader*        m_pComputeShader;
```

The CreateComputeShader function is utilized as follows:

```
m_pD3dDevice->CreateComputeShader(shaderBlob->GetBufferPointer(),
                            shaderBlob->GetBufferSize(), NULL, m_pComputeShader);
```

4. Initialization of resources (textures, buffers… see section 3.6.3).

Buffer and texture resources must be allocated if a shader is to have any means of input or output, otherwise it will have limited if any useful purpose.

5. Initialization of resource views (see section 3.6.3).

Resource views are the means of making the defined buffer/texture resources accessible by the shader.

6. Set resource views (see section 3.6.3).

A list of specific resource views can quickly be set or changed to affect what resources are accessible to a shader, and how they are accessible.

7. Set the current shader.

As discussed in section 2.3.6, multiple compute shaders may be necessary for performing a large computational procedure that requires global (device-wide) thread synchronization. In order to execute a series of different compute shaders, the set shader must be changed. This is achieved using the Direct3D CSSetShader function.

```
m_pD3dImmediateContext->CSSetShader(m_pComputeShader, NULL, 0);
```

8. Dispatch thread groups for shader execution.

To execute a shader, it must be dispatched to be executed by a number of thread groups. If the thread group size (dimensions of shader thread group measured in number of threads in x, y or z; see section 2.3.6) is first known. For example, in the simulation application, thread groups are one-dimensional, and only have size in the x direction. The number of thread groups required is calculated as the number of total threads required divided by the number of threads per thread group. This number must be explicitly rounded up, since by default, integer division is implicitly rounded down in programming languages.

```
unsigned int threadGroupSizeX = static_cast<unsigned int>(
              ceil(static_cast<double>()
                / static_cast<double>(THREAD_GROUP_SIZE_X)));
```

When the dimensions of the thread group grid size are defined (number of thread groups to be executed in x, y and z dimensions), the Direct3D function Dispatch is called.

```
m_pD3dImmediateContext->Dispatch(threadGroupSizeX, 1, 1);
```

When a dispatch call has been made, a different shader can be set, and further dispatch calls executed. These queue up on the GPU for execution, asynchronously to the CPU.

In this research project, global thread synchronization is necessary at multiple stages of the simulation. The set shaders and dispatch calls are made as follows:

```
// PHASE “0”: Global Matrices - Reset
m_pD3dImmediateContext->CSSetShader(m_pCSPhase0, NULL, 0);
m_pD3dImmediateContext->Dispatch(threadGroupSizeX, 1, 1);
// PHASE 1: Global Matrices - Assembly
m_pD3dImmediateContext->CSSetShader(m_pCSPhase1, NULL, 0);
m_pD3dImmediateContext->Dispatch(threadGroupSizeX, 1, 1);
// PHASE 2: Global Matrices - Preparation
m_pD3dImmediateContext->CSSetShader(m_pCSPhase2, NULL, 0);
m_pD3dImmediateContext->Dispatch(threadGroupSizeX, 1, 1);
// PHASE 3: Global Matrices - Solution
m_pD3dImmediateContext->CSSetShader(m_pCSPhase3, NULL, 0);
m_pD3dImmediateContext->Dispatch(threadGroupSizeX, 1, 1);
// PHASE 4: Calculate new T-dependent thermal parameters
m_pD3dImmediateContext->CSSetShader(m_pCSPhase4, NULL, 0);
m_pD3dImmediateContext->Dispatch(threadGroupSizeX, 1, 1);
// PHASE 5: Store output data to FieldDataBuffer
m_pD3dImmediateContext->CSSetShader(m_pCSPhase5, NULL, 0);
m_pD3dImmediateContext->Dispatch(threadGroupSizeX, 1, 1);
```

Where m_pCSPhase# are individual compute shaders. In the simulation application, source code for all compute shaders exist within the same .hlsl source file, and separate shaders are compiled from the one file. The shaders used in the simulation application are described in Table 3.15.

Table 3.15: Simulation Compute Shaders

Phase	Shader Name	Purpose
“0”	m_pCSPhase0	Resets element values in global matrix/vector buffers, to prevent accidental carry-over of temporary data from previous iterations.
1	m_pCSPhase1	Calculates element matrices/vector and stores their global matrix/vector contributions to the “pool” data structures discussed in section 3.6.3.
2	m_pCSPhase2	Sums the element contributions for each global matrix/vector element and performs preliminary operations to prepare global coefficient matrix and forcing vector for iterative numerical solution.
3	m_pCSPhase3	Performs Gauss-Seidel operations on the global equation system (matrix/vector).
4	m_pCSPhase4	If the simulation is specified by settings as temperature-dependent, this shader recalculates the temperature-dependent parameters for each node, based on that node’s temperature.
5	m_pCSPhase5	Stores the calculated temperatures / thermal damage to a buffer, from which the results may be copied from GPU to CPU.

The HLSL shader source code is laid out to describe the shader input/output buffer resources.

```
StructuredBuffer<GTetra>    TetraBuffer                     : register(t0);
RWStructuredBuffer<GNode>   NodeBuffer                      : register(u0);
```

```
RWStructuredBuffer<GData>   FieldDataBuffer             : register(u1);
RWStructuredBuffer<MData>   GlobalCoeffMatrixPool       : register(u2);
RWStructuredBuffer<MData>   GlobalCoeffMatrixData       : register(u3);
RWStructuredBuffer<MRow>    GlobalCoeffMatrixRow        : register(u4);
RWStructuredBuffer<MData>   GlobalForcingVectorPool     : register(u5);
RWStructuredBuffer<MData>   GlobalForcingVectorData     : register(u6);
```

Where GTetra, GNode, GData, MData and MRow are data structures defined both in the shader (HLSL) source and on the CPU (C++). For example:

```
struct GNode {
       float x;                        // Node x coordinate
       float y;                        // Node y coordinate
       float z;                        // Node z coordinate
       float tDensity;                 // Tissue density
       float tSpecificHeatCapacity;    // Tissue specific heat capacity
       float tConductivity;            // Tissue thermal conductivity
       float blSpecificHeatCapacity;   // Blood specific heat capacity
       float blPerfusion;              // Blood perfusion rate
};
```

After this, secondary functions such as those which calculate element matrices, are "forward-declared"; for example:

```
float4x4 ComputeTetraThermalStiffnessMatrix(float volume, uint n0, uint n1, uint n2,
uint n3);
```

Forward declarations are the means of telling the source code compilation unit that a function definition for that type, name and set of parameters exists somewhere in the source code file. Forward declaration is necessary, because if a particular command in the source code calls a specific function that has not been defined in the source before that point, the compiler will not know if the function's definition/implementation exists.

Functions in HLSL are defined similarly to the syntax implemented in C++. For example, a simple function to truncate a floating point number:

```
float TruncFloat(float fToTruncate, float places) {
       int a = (int)(fToTruncate / places);
       return ((float)a * places);
}
```

Each compute shader has a specific "main" function as its entry point (as previously mentioned). An example of such a function is defined below for the particularly brief main function of m_pCSPhase4; the main function having been named "Phase4".

```
[numthreads(THREAD_GROUP_SIZE_X, THREAD_GROUP_SIZE_Y, THREAD_GROUP_SIZE_Z)]
void Phase4(uint3 dispatchThreadID : SV_DispatchThreadID) {
       // Node / Row Index
       uint row = dispatchThreadID.x;

       // Conditional to ignore extraneous threads
       if (row < numNodes) {

              float temperature = GlobalCoeffMatrixRow[row].current;

              // Calculate Tissue Specific Heat Capacity
              NodeBuffer[row].tSpecificHeatCapacity =
                                     CalculateTissueSpecificHeatCapacity(temperature);

              // Calculate Tissue Conductivity
```

```
            NodeBuffer[row].tConductivity =
                                CalculateTissueThermalConductivity(temperature);
      }
}
```

Note how the thread ID (discussed in section 2.3.6) is defined. The x ID of the thread could be based on its ID local to the thread group, or globally. We are interested in its global ID of the thread, and since thread groups in the application have been defined as one-dimensional to suit the problem type at hand, then it is only the x coordinate/ID of the thread that we are interested in. This is derived from the 3-component unsigned (i.e. positive value only) integer dispatchThreadID. By using dispatchThreadID.x as the index by which resources are indexed by the shader for a particular thread, then each single thread for this shader (m_pCSPhase4) will recalculate the tissue specific heat capacity and tissue thermal conductivity for a single node.

Thus, a large process can be highly parallelized for multi-thread/core execution on the GPU.

3.6.5 CPU Profiling

In order to time the performance of the simulation on the CPU, the Windows functions QueryPerformanceFrequency and QueryPerformanceCounter are implemented. The performance counter is a high resolution (<1 µs) time stamp that can be used to measure time intervals. By creating a class to manage the timer behaviour, an operation can simply be timed by including the header file, creating the timer, calling the SetToCurrent function just before the operation to be timed, and then checking the value returned by calling DeltaMicroseconds when the operation has completed. The following C++ header and source files show the structure of the class. Calling SetToCurrent stores the current timestamp to m_prev, and calling DeltaMicroseconds obtains the new current timestamp, subtracts m_prev, and divides the result by the frequency m_freq (number of performance counter ticks per second). The frequency obtained by QueryPerformanceFrequency is constant for the hardware, and need only be obtained once; at initialization.

Timer.h

```
#ifndef MASTERS_TIMER_H
#define MASTERS_TIMER_H
class Timer {
public:
	Timer();
	~Timer();
	bool					Init();
	unsigned long				DeltaMicroseconds();
	void					SetToCurrent();
	static const unsigned long		m_microPerSecond;

private:
	__int64					m_freq;
	__int64					m_prev;
};

#endif
```

Timer.cpp

```
#include "Timer.h"
#define WIN32_LEAN_AND_MEAN
#include <Windows.h>
#include <cmath>

const unsigned long Timer::m_microPerSecond = 1000000;

Timer::Timer() {}

Timer::~Timer() {}

bool Timer::Init() {
	if (!QueryPerformanceFrequency((LARGE_INTEGER*)&m_freq)) {
		return false;
	}
	return true;
}

unsigned long Timer::DeltaMicroseconds() {
	__int64 c, delta;
	QueryPerformanceCounter((LARGE_INTEGER*)&c);
	delta = m_microPerSecond * (c - m_prev);
	m_prev = c;
	return static_cast<long>(std::round(static_cast<double>(delta)
		/ static_cast<double>(m_freq)));
}

void Timer::SetToCurrent() {
	QueryPerformanceCounter((LARGE_INTEGER*)&m_prev);
}
```

3.6.6 GPU Profiling

The CPU and GPU run asynchronously, therefore it is important to note that GPU performance **can't** be profiled by using the type of timing technique employed in section 3.6.5. Were a person to use the technique like so:

```
m_pTimer->SetToCurrent();
m_pD3dImmediateContext->Dispatch(threadGroupSizeX, 1, 1);
long deltaMicroseconds = m_pTimer->DeltaMicroseconds();
```

then they would only be measuring the time taken for the CPU to send the Dispatch task to the GPU's queue for execution.

GPU profiling is achieved slightly differently, by using Direct3D. Firstly, Direct3D Query objects must be created.

```
ID3D11Query*  m_pQueryDisjoint;
ID3D11Query*  m_pQueryComputeBegin;
ID3D11Query*  m_pQueryComputeEnd;
```

By creating a Query description, Direct3D is notified as to whether the Query object will be holding frequency and disjoint information, or a timestamp. A disjoint is the rare event that the counter became discontinuous due to unplugging an AC power lead, overheating or throttling changes due to other laptop events. The Query objects are created from the ID3D11Query pointers and the Query descriptions.

```
D3D11_QUERY_DESC descDisjoint;
ZeroMemory(&descDisjoint, sizeof(descDisjoint));
descDisjoint.Query = D3D11_QUERY_TIMESTAMP_DISJOINT;
m_pD3dDevice->CreateQuery(&descDisjoint, &m_pQueryDisjoint);

D3D11_QUERY_DESC descComputeBegin;
ZeroMemory(&descComputeBegin, sizeof(descComputeBegin));
descComputeBegin.Query = D3D11_QUERY_TIMESTAMP;
m_pD3dDevice->CreateQuery(&descComputeBegin, &m_pQueryComputeBegin);

D3D11_QUERY_DESC descComputeEnd;
ZeroMemory(&descComputeEnd, sizeof(descComputeEnd));
descComputeEnd.Query = D3D11_QUERY_TIMESTAMP;
m_pD3dDevice->CreateQuery(&descComputeEnd, &m_pQueryComputeEnd);
```

When the Query objects have been initialized, their timestamp orders are set in the GPU's queue, similarly to a shader dispatch call.

```
m_pD3dImmediateContext->Begin(m_pQueryDisjoint);
m_pD3dImmediateContext->End(m_pQueryComputeBegin);
m_pD3dImmediateContext->Dispatch(threadGroupSizeX, 1, 1);
m_pD3dImmediateContext->End(m_pQueryComputeEnd);
m_pD3dImmediateContext->End(m_pQueryDisjoint);
```

The GPU proceeds through the queue, and the CPU must wait by incrementally thread sleeping until disjoint and timestamp data exists and can be retrieved.

```
while (m_pD3dImmediateContext->GetData(m_pQueryDisjoint, NULL, 0, 0) == S_FALSE) {
       Sleep(1);
}
```

If a disjoint has occurred, then any obtained timestamp data will not reflect the real GPU performance and should be disregarded. This check is performed by the "if" control statement that bounds the calls to "GetData". These function calls retrieve the timestamps to variables which can be worked with. Similarly now to CPU profiling, the difference between timestamps (end minus beginning) divided by the frequency, gives the number of seconds taken by the GPU to perform the timed dispatch (and subsequent execution) operation.

```
float computeTimeEnd = 0.0f;

D3D11_QUERY_DATA_TIMESTAMP_DISJOINT tsDisjoint;
m_pD3dImmediateContext->GetData(m_pQueryDisjoint, &tsDisjoint, sizeof(tsDisjoint), 0);

if (!tsDisjoint.Disjoint) {
       UINT64 tsComputeBegin, tsComputeEnd;
       m_pD3dImmediateContext->GetData(m_pQueryComputeBegin, &tsComputeBegin,
                                                             sizeof(UINT64), 0);
       m_pD3dImmediateContext->GetData(m_pQueryComputeEnd, &tsComputeEnd,
                                                             sizeof(UINT64), 0);
       computeTimeEnd = float(tsComputeEnd - tsComputeBegin) /
                                                      float(tsDisjoint.Frequency);
}
```

Finally, the microseconds occupied by the shader dispatch and execution are calculated by multiplying the number of seconds (a floating point number) by the amount of microseconds in a single second (one million), and the result is converted to a whole number of microseconds by casting to a long (long integer) variable type.

```
long computedDeltaMicrosecondsEnd = static_cast<long>(computeTimeEnd * 1000000.0f);
```

As an example of profiling the multiple shader phases discussed in section 3.6.4, a profiling test was run for a single sub-iteration for a temperature-dependent case (calculate temperature field and update temperature-dependent thermal parameters; multiple sub-iterations cause the temperature-dependent parameter values to converge). The profiled simulation sub-iteration used the following simulation parameter values:

- Time step: 0.002 s
- Margin of error: 0.001 s
- Mesh resolution: 20×20×20 cubes; subdivided into 6 linear tetrahedra each
- Mesh element size: 0.1 mm (cube side length prior to subdivision)

The results are tabulated and compared graphically in Table 3.16 and Figure 3-10 respectively:

Table 3.16: GPU Shader Profiling

Process	**Duration (µs)**	**Percentage of Total Duration**
Update Constant Buffer	1	<0.1%
Reset Global Matrix Buffers	1671	8.7%
Calculate Element Matrix Contributions	14260	74.4%
Assemble Global Matrices	1327	6.9%
Solve Global Matrices	1876	9.8%
Calculate New Thermal Parameters	24	0.1%
Store Output Data to Buffer	13	<0.1%
TOTAL	19172	100%

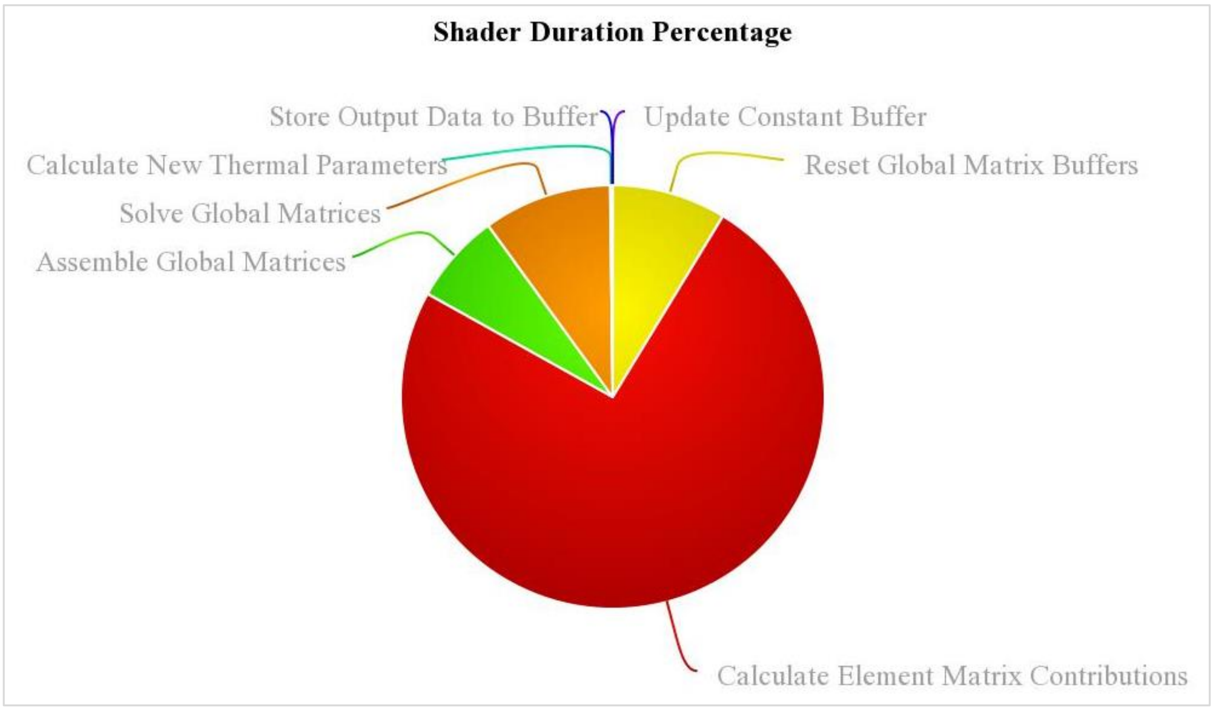


Figure 3-10: GPU Shader Profiling - Contributions

As can be seen, the vast majority of execution time is occupied with the population of the global coefficient matrix and global forcing vector. This is because the calculation of element matrix contributions must be performed, stored, and then a parallel gather (“Assemble Global Matrices”) implemented to pool together these element matrix contributions into the global matrices in order to avoid GPU memory write conflicts. If further improvements are to be made, then the global assembly process needs priority attention, hopefully to provide a more efficient means of assembly. This is however, likely to be difficult to achieve without implementing specific numerical methods which change the nature of the solution methodology; i.e. matrix preconditioning.

3.6.7 Convergence Testing

Convergence testing was implemented by utilizing two parameters: a margin of error, and a maximum number of Gauss-Seidel iterations before solver termination. Each solver iteration, the previously calculated field value (temperature) is subtracted from the newly calculated one, and the magnitude of the difference recorded. If the magnitude is greater than the margin of error, then the solver continues iterating.

For example, the CPU Gauss-Seidel solver loops for every iteration until the maximum number of solver iterations is reached. For each loop, it determines that the current iteration is converging, and it is up to the comparison of each individual node's error with the specified global margin of error, to prove that the solver is not yet converging within the designated margin of error. If the solver it satisfactorily converging however, the loop breaks and the simulation proceeds with its next task.

```
for (unsigned int iteration = 0; iteration < maxSolverIterations; ++iteration) {
	bool iterationConverging = true;
	for (unsigned int node = 0; node < m_pMesh->nodes.size(); ++node) {

		//... Do Gauss-Seidel iteration

		if (m_globalCoeffMatrixRow[node].error > errorMargin) {
			iterationConverging = false;
		}
	}
	if (iterationConverging) {
		break;
	}
}
```

Figure 3-11 shows the number of Gauss-Seidel solver iterations taken for the typical simulation executed for this research project to converge within the margin of error. The number of iterations required jumps sharply when the margin is specified at 0.00001. The typical margin of error used for simulations was 0.001 however (see section 3.4.4).

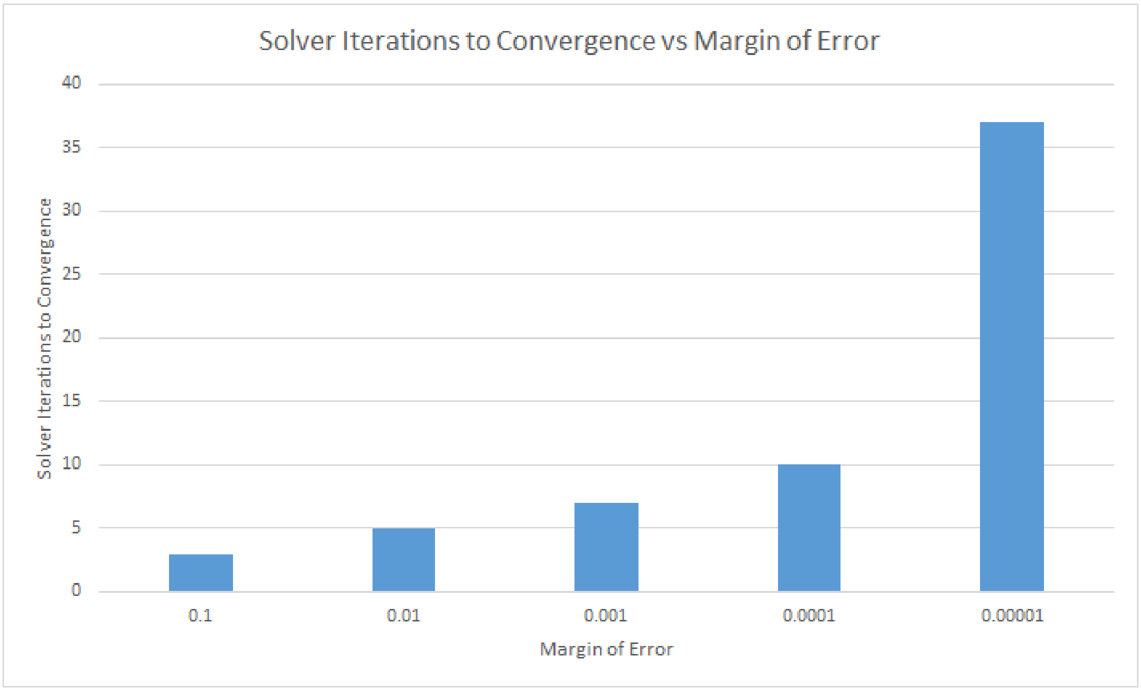


Figure 3-11: Solver Iterations to Convergence vs Margin of Error

4 Results & Discussion of Results

4.1 Temperature-Independent vs Temperature-Dependent Thermal Parameter Temperature/Damage Distributions

4.1.1 Temperature

Simulation parameters for the results derived to compare temperature distribution for temperature-independent versus temperature-dependent thermal parameters are as follows:

- Time step: 0.002 s
- Time steps executed: 1000 steps
- Margin of error: 0.001
- Mesh resolution: 20×20×20 cubes: subdivided into 6 linear tetrahedra each
- Mesh element size: 0.1 mm (cube side length prior to subdivision)
- Source point temperature: 54 °C

After 2 seconds, the temperature distributions in tissue with temperature-independent and temperature-dependent thermal parameters (tissue specific heat capacity and tissue thermal conductivity) were evaluated and compared. The temperature distribution for all mesh nodes in the mid-plane of the virtual block of tissue are shown in Figure 4-1 and Figure 4-2.

Note the rapid temperature falloff that compares well with the steady state results obtained by Karaa, Zhang and Yang (2005) in Figure 2-7 and Figure 2-8.

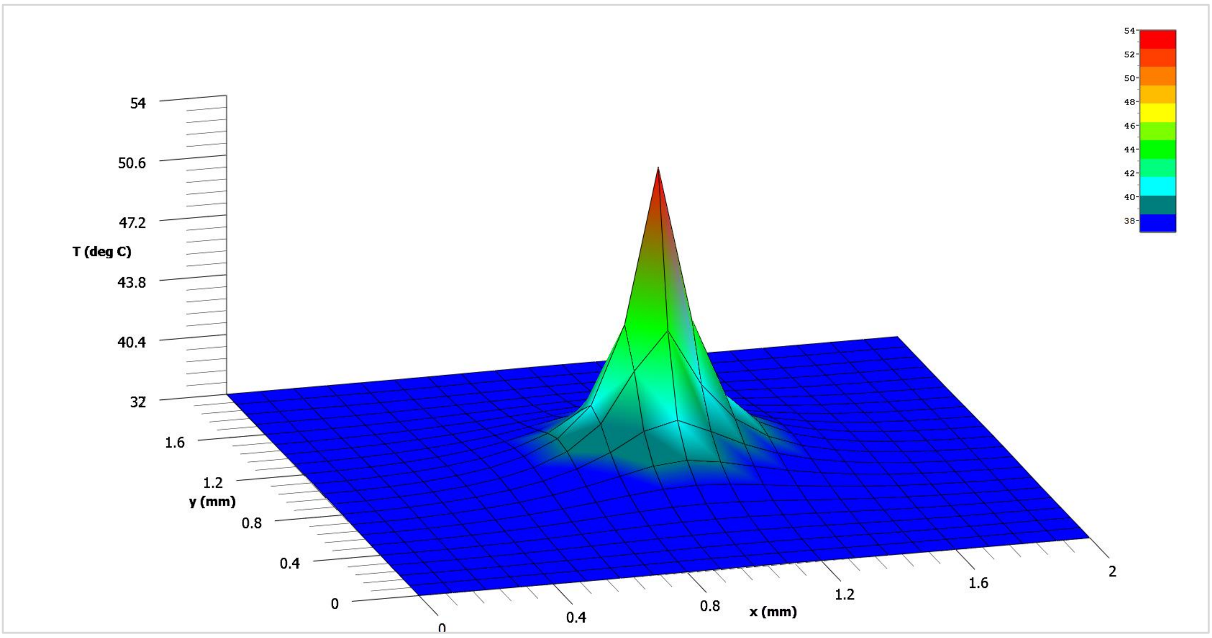


Figure 4-1: Temperature-Independent Thermal Parameter Temperature Distribution

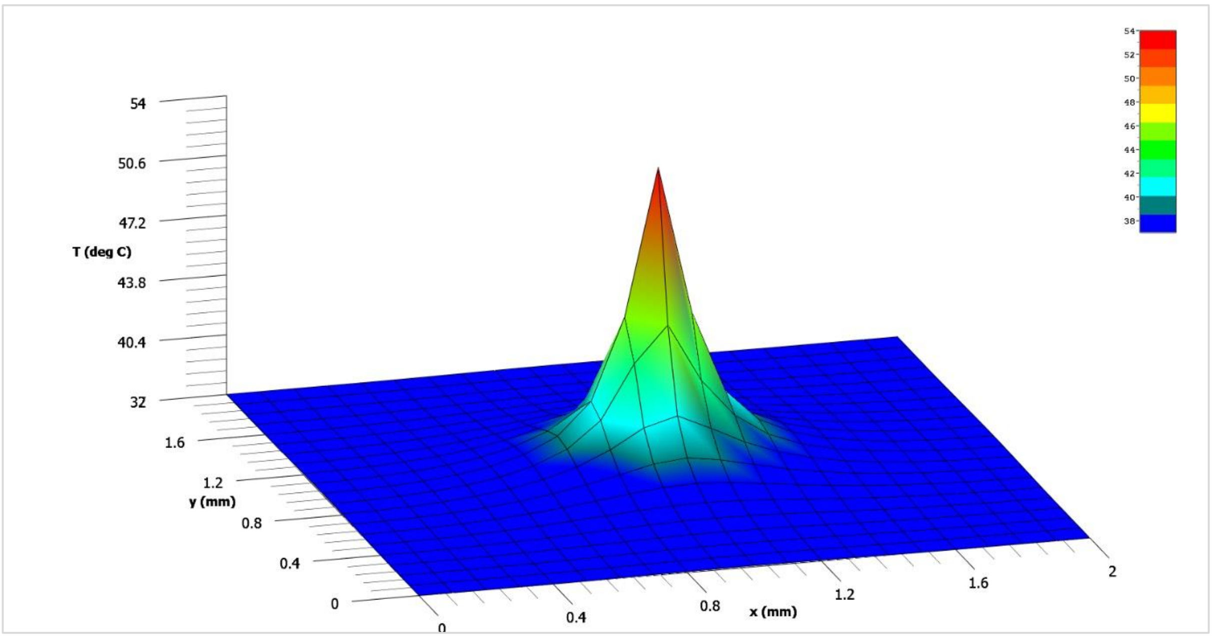


Figure 4-2: Temperature-Dependent Thermal Parameter Temperature Distribution

Close examination shows what appears to be only a minor variation in temperature distribution for the temperature-dependent and temperature-independent cases. Even in Figure 4-3, which compares the two temperature distributions for the mid-line mesh nodes of the mid-plane for both cases, it is difficult to notice a significant difference (at least visually) in the temperature distribution.

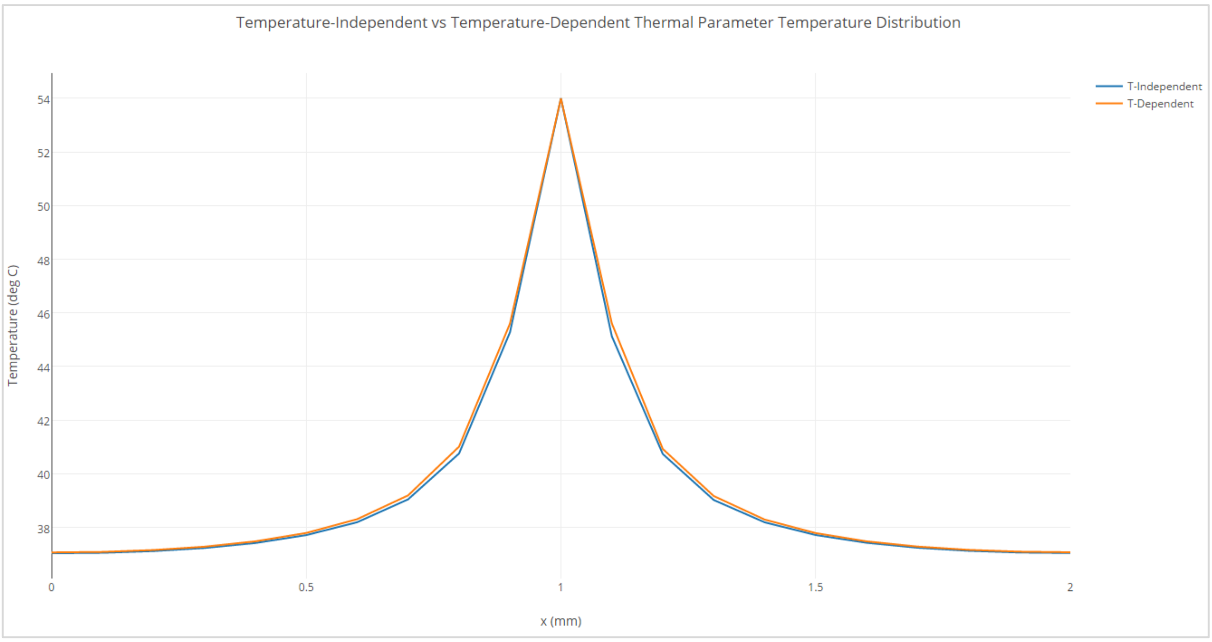


Figure 4-3: Temperature-Independent vs Temperature-Dependent Thermal Parameter Temperature Distribution

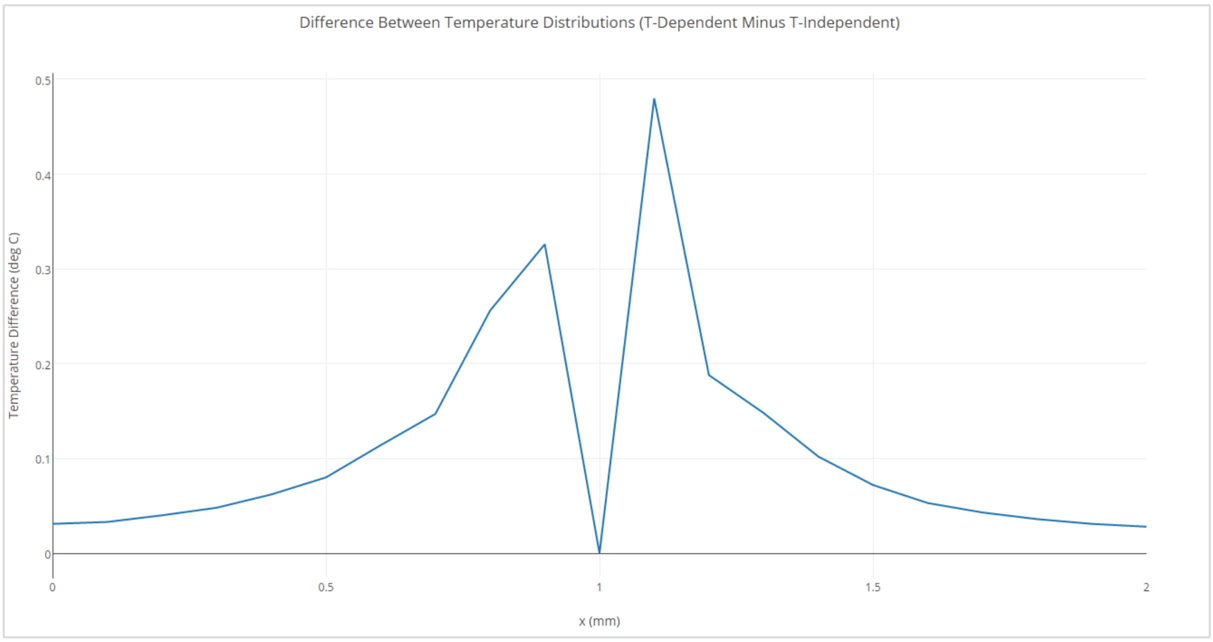


Figure 4-4: Difference between Temperature Distributions (T-Dependent minus T-Independent)

Note that the temperature immediately left of the heat source indicates a temperature difference of 0.3 °C, while a difference of 0.5 °C can be observed on the immediate right. The asymmetry of the result is most likely due to the nature of GPU computation, where Gauss-Seidel iterations are completed in chunks; that is, not all thread groups execute simultaneously. Some thread groups are completed before other thread groups are moved to the GPU multiprocessors for computation. Symmetry should be more apparent, given increased simulation duration. The significance of the outcome at only 2 seconds already sufficiently indicates that the case of temperature-dependent thermal parameters causes a slight increase in the temperature throughout the distribution. In this case: up to half a degree Celsius.

Watanabe et al. (2009) concluded that temperature far from the temperature source does not differ greatly between distributions for temperature-independent and temperature-dependent thermal parameter (tissue specific heat capacity and tissue thermal conductivity) cases. They found that temperature for the temperature-independent parameter model actually increases more than the temperature-dependent parameter model at the centre of the electrode heat source (being temporal; i.e. flux-based/Neumann, not fixed as in this project; i.e. Dirichlet).

The differences between these results and those of Watanabe et al. (2009) are most likely due to the choice of temperature-independent tissue specific heat capacity and tissue thermal conductivity. (Watanabe et al. 2009) chose a temperature-independent tissue specific heat capacity of $C_{ti} = 3600\ \frac{J}{kg{\cdot}°K}$ and a temperature-independent tissue thermal conductivity of $k_{ti} = 0.502\ \frac{W}{m{\cdot}°K}$. Both of these were selected parameters based on duck tissue, while their temperature-dependent counterparts were modelled on hog liver; represented in Figure 2-10 Figure 2-11. Comparing two results based entirely on different tissue types demeans the significance of their comparison, as the temperature field by its self could vary significantly between two different species' tissues (duck and hog) alone. The reasoning for selecting a temperature-independent tissue specific heat capacity of $C_{ti} = 3405\ \frac{J}{kg{\cdot}°K}$ and a temperature-independent tissue thermal conductivity of $k_{ti} = 0.4604\ \frac{W}{m{\cdot}°K}$ for this research project is that these values are derived/interpolated from the temperature-dependent data at a temperature of 37 °C. This causes the temperature-independent and temperature-dependent parameter scenarios to be naturally comparable, since at a tissue temperature of 37 °C, both scenarios will yield the same value for tissue specific heat capacity and tissue thermal conductivity, and thus the same temperature as expected. Furthermore, considering the relationships between tissue specific heat capacity and tissue thermal conductivity and temperature; as shown in Figure 2-10 Figure 2-11; it is obvious that heat transfer and thus temperature will increase in the presence of a heat source for a temperature-dependent thermal parameter temperature distribution scenario as tissue temperature increases. Watanabe et al. (2009) would likely find their analyses (temperature-dependent and temperature-independent parameters) incoherent even if both were performed on a single point of tissue at exactly 37 °C, due to their parameter mismatch.

For further reference of general temperature distribution behaviour, consider Figure 4-5 and Figure 4-6, where the temperature distribution along the x axis of the tissue mesh mid-plane is plotted at different time intervals. As time progresses, the temperature distribution tends/converges towards a steady state distribution, and the distribution peak its self, smoothens.

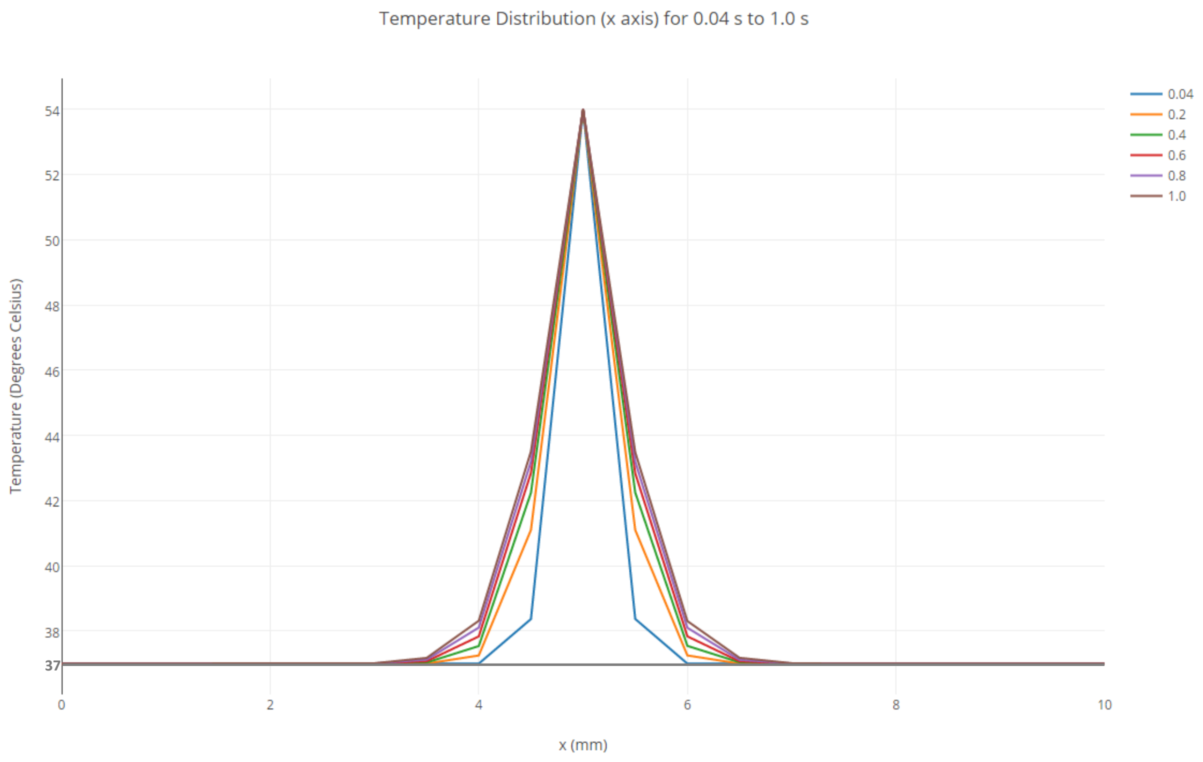


Figure 4-5: Temperature Distribution from 0.04 s to 1.0 s

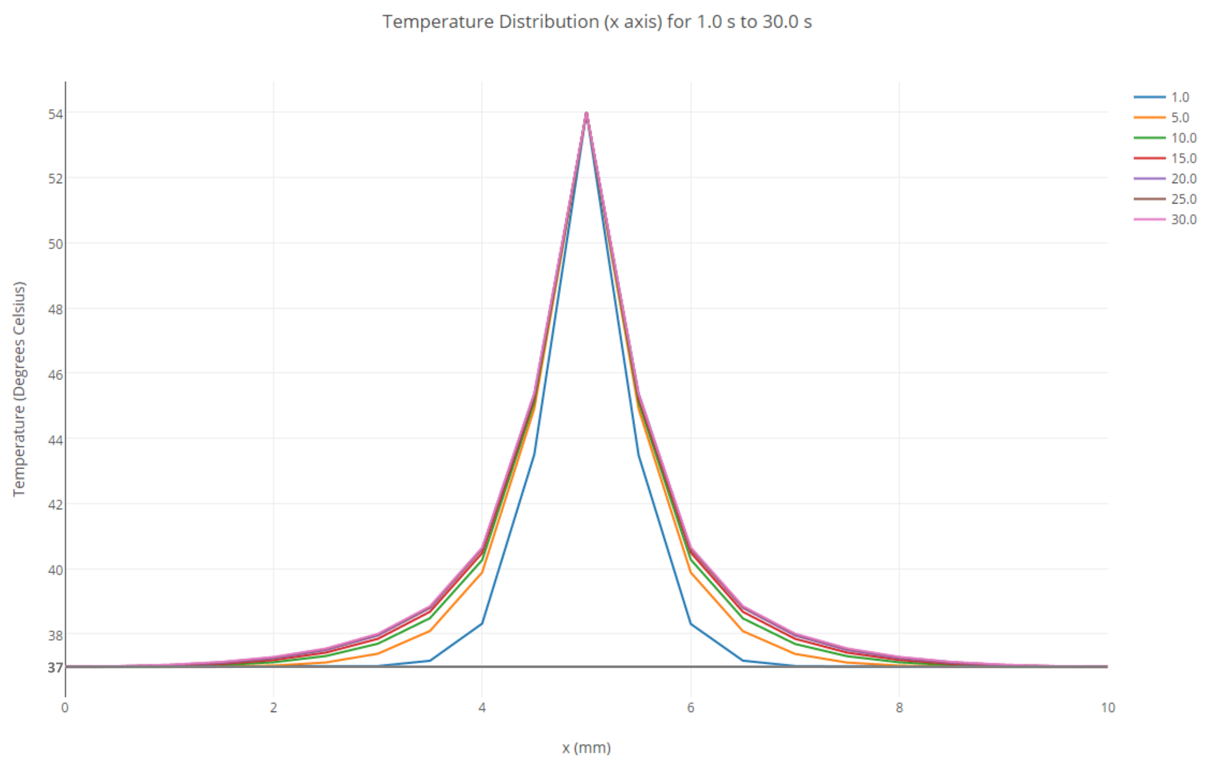


Figure 4-6: Temperature Distribution from 1.0 s to 30.0 s

Furthermore, note that the simulation, while being performed on a uniform tetrahedral mesh (with tetrahedral elements formed into a cubic mesh), is extensible to arbitrary geometries such as the shape of a human liver (Figure 4-7). Where output based on anatomically accurate geometry is desired, a mesh of sufficient resolution (Figure 4-8) can produce a temperature distribution as shown in Figure 4-9, where it can be seen that the mesh resolution is refined in close proximity to the single point heat source. Much greater detail of the temperature distribution (at t = 2 seconds) can be observed in Figure 4-10, and the corresponding thermal damage distribution upon the tissue can be observed in Figure 4-11. Regular geometrical arrangements based on tetrahedral elements are employed in this research project for the results, since they can more easily be generated at different resolutions, and in-plane results can more easily be quantified and presented in this thesis report. These images prove the ability of the simulation software to work with irregular meshes.

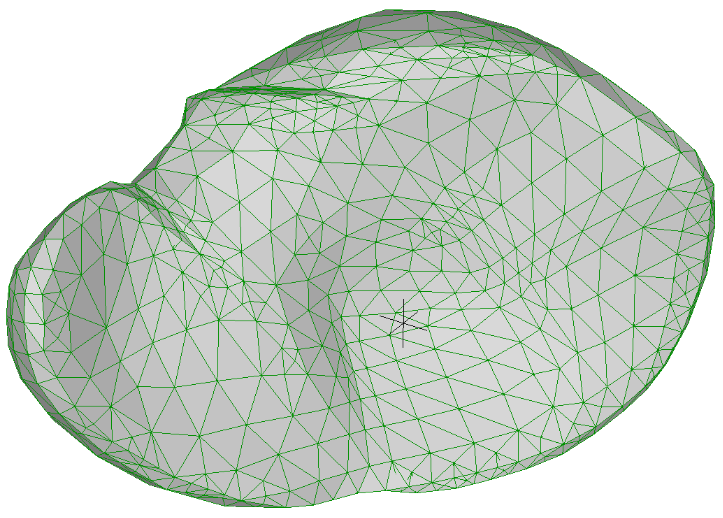


Figure 4-7: Tetrahedral Liver Mesh

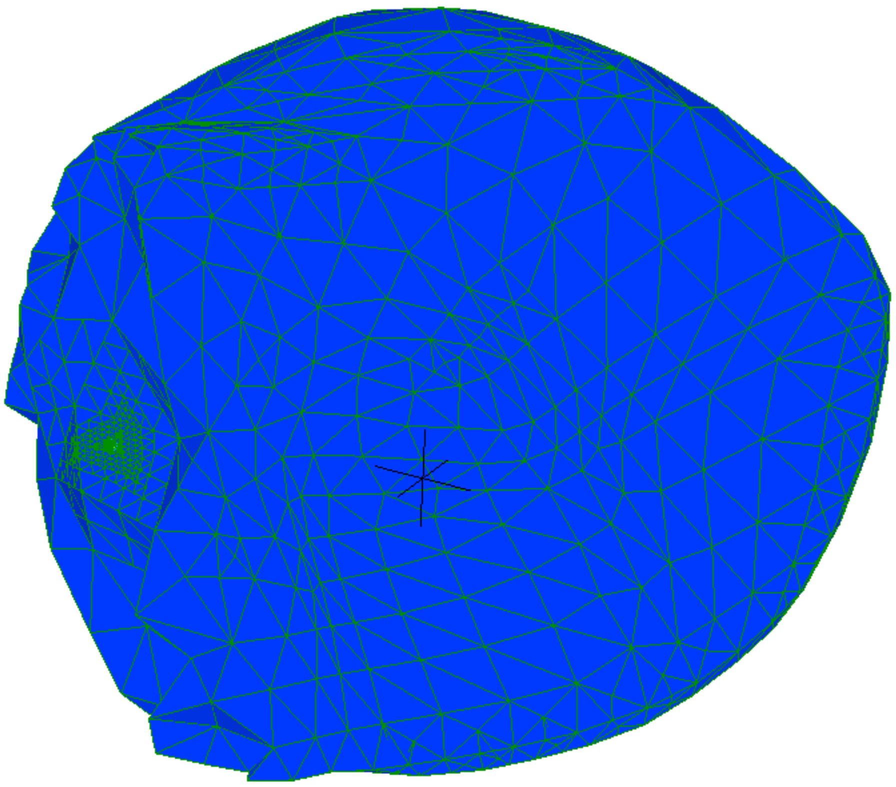


Figure 4-8: Tetrahedral Liver Mesh Section

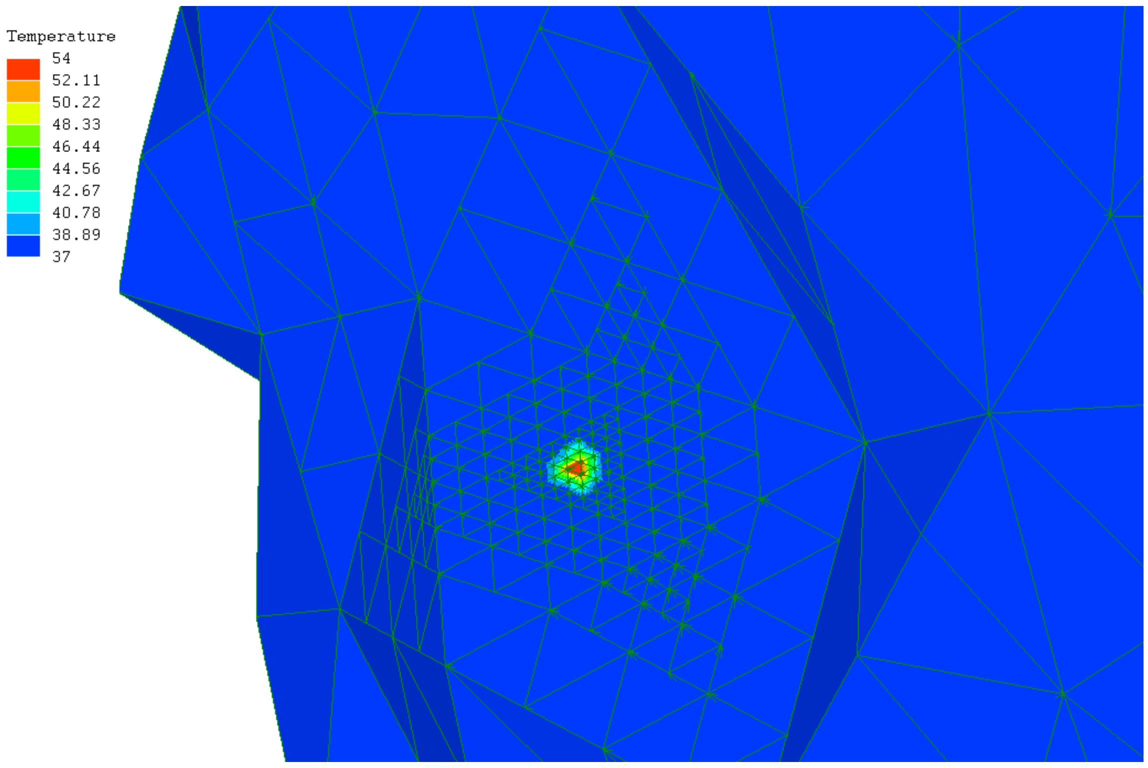


Figure 4-9: Point Source Temperature Distribution upon Tetrahedral Liver Mesh Section

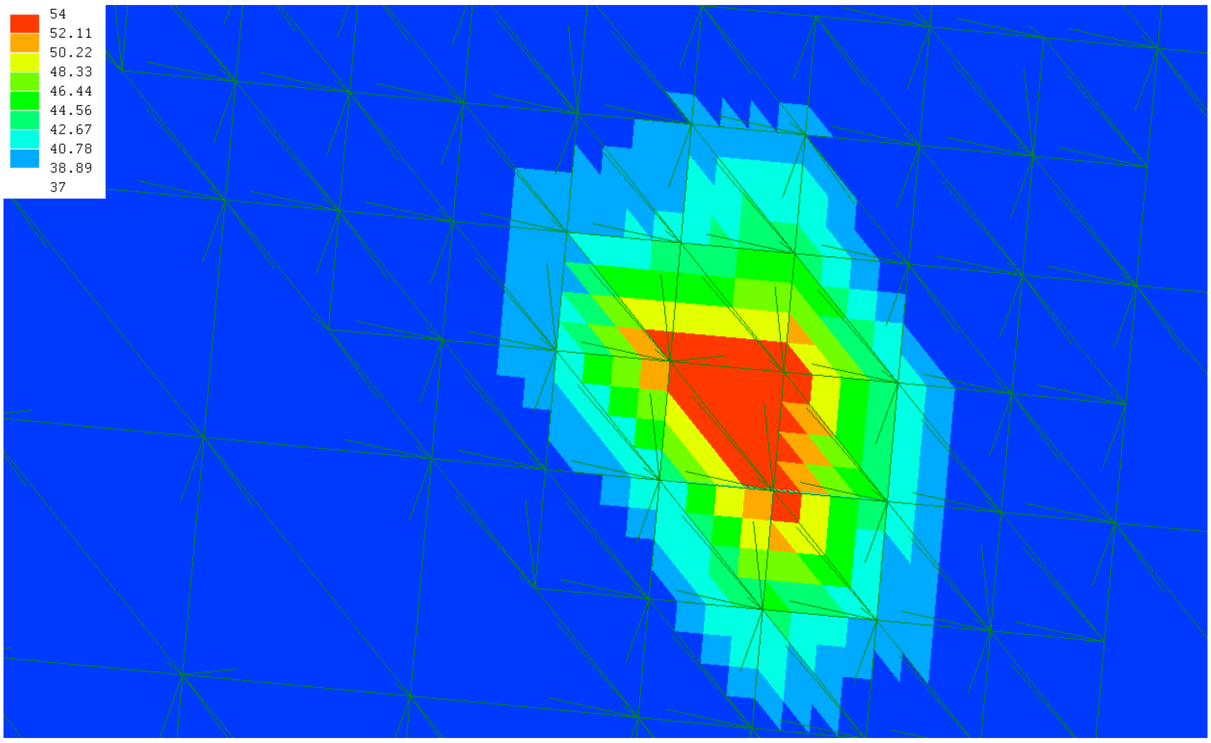


Figure 4-10: Liver Point-Source Temperature Distribution (°C) at t = 2s

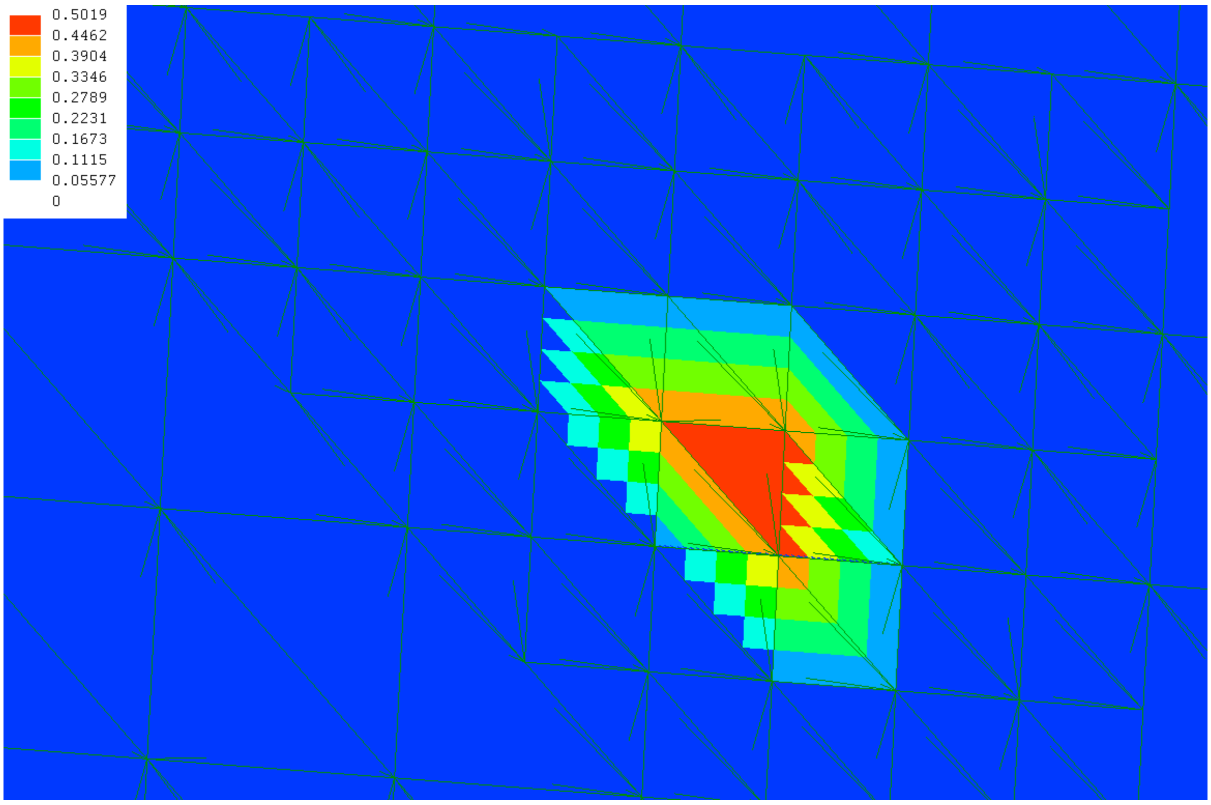


Figure 4-11: Liver Point-Source Thermal Damage Distribution (Ω) at t = 2s

4.1.2 Thermal Damage

The difference in degree of thermal damage sustained by the tissue due to the temperature distribution also seems to be barely noticeable upon visual inspection/comparison, as shown in Figure 4-12 and Figure 4-13. Examining Figure 4-3, the temperature distribution for both cases exceeds the threshold temperature for initiating tissue thermal damage ($T = 43\,°C$; see section 2.7.2) at the source temperature location, and the nodes immediately adjacent to the source temperature location. The effect of blood perfusion tends to produce a rapid temperature falloff such that only a very small amount of tissue can receive thermal damage. This exhibits the body's defence mechanism against thermal damage.

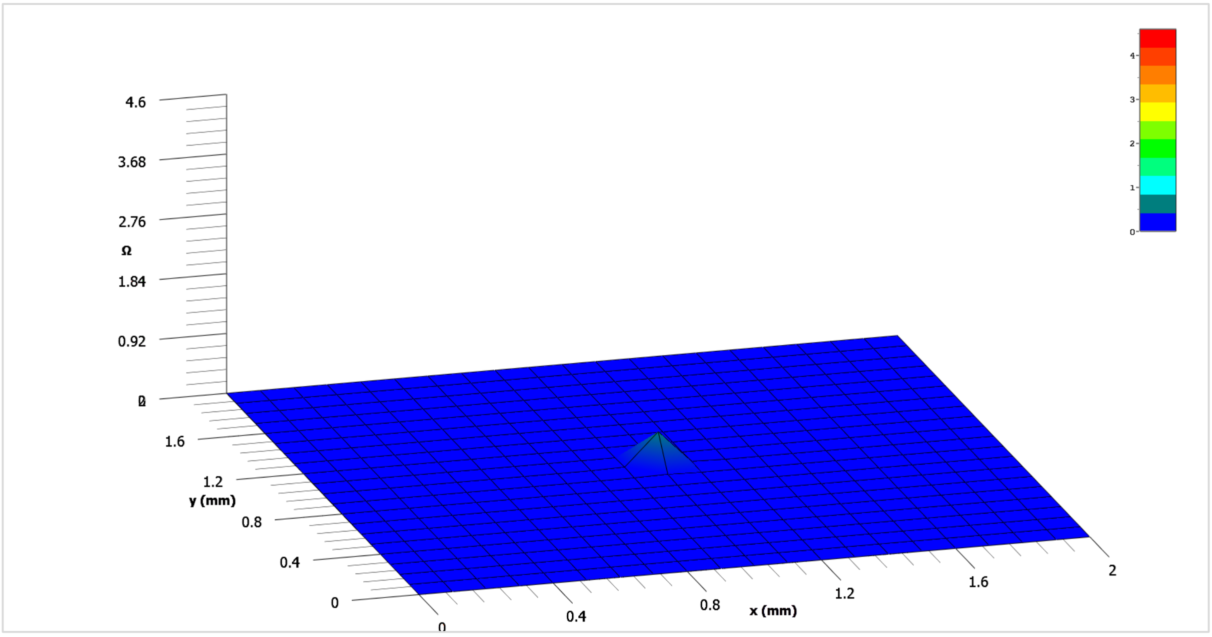


Figure 4-12: Temperature-Independent Thermal Parameter Thermal Damage Distribution

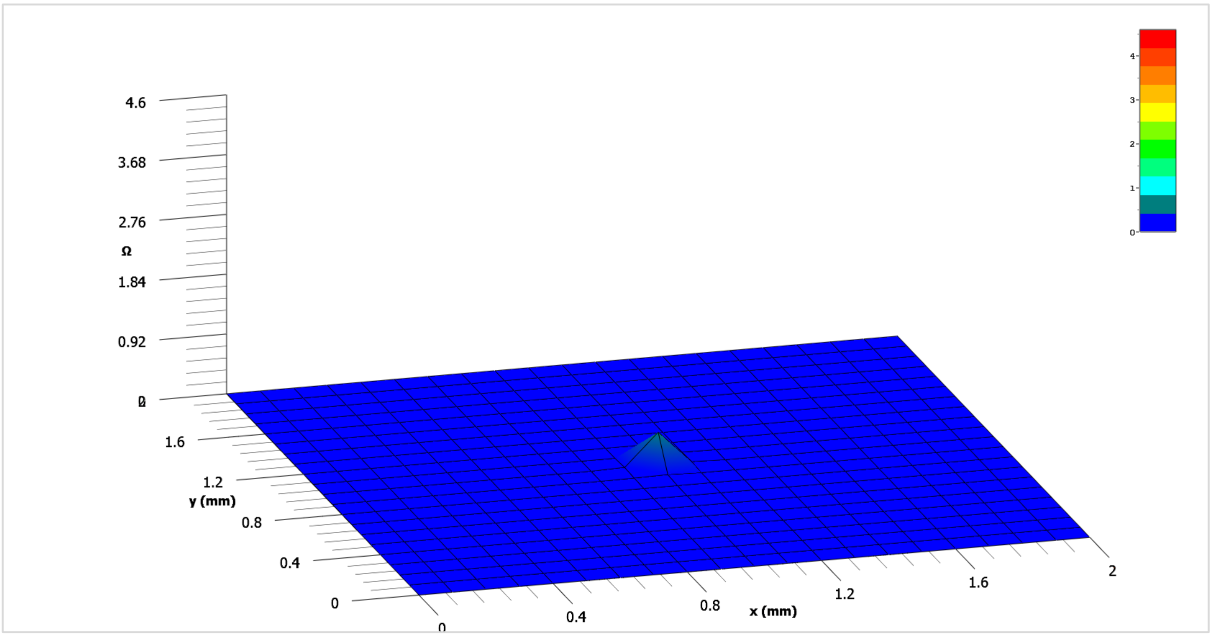


Figure 4-13: Temperature-Dependent Thermal Parameter Thermal Damage Distribution

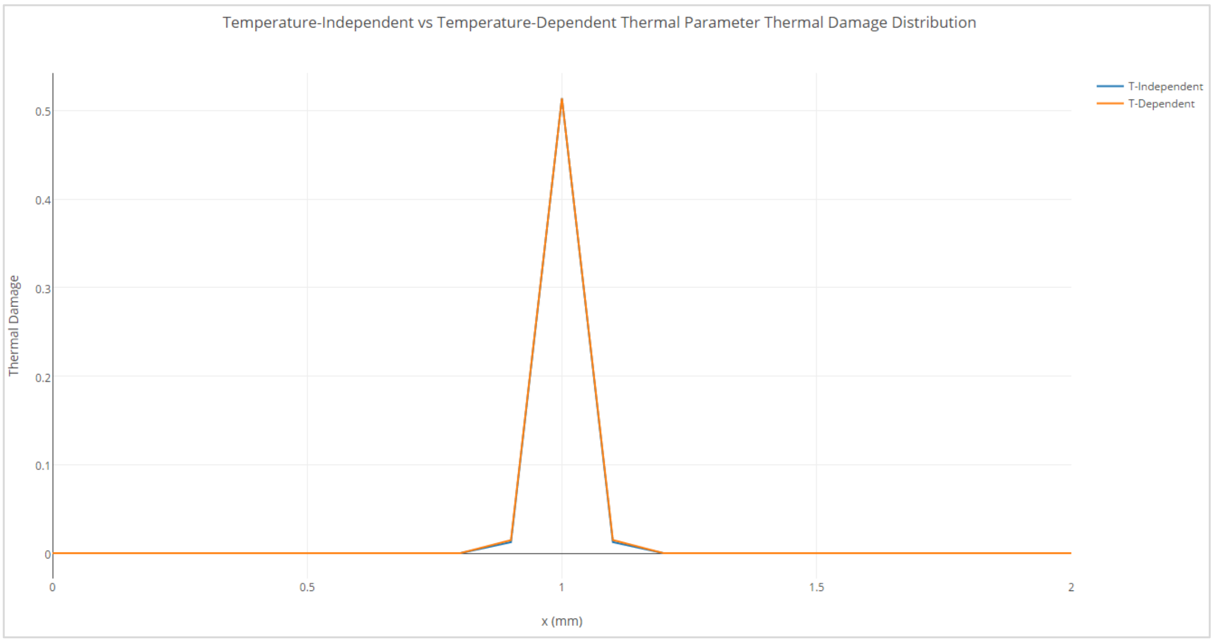


Figure 4-14: Temperature-Independent vs Temperature-Dependent Thermal Parameter Thermal Damage Distribution

Examining Figure 4-14, the difference between thermal damage for both scenarios indicates that, logically (since the temperature-dependent thermal parameter case yields higher temperatures), the temperature-dependent thermal parameter case yields greater thermal damage. Figure 4-15 shows the amount of extra thermal damage obtained by the tissue in the temperature-dependent thermal parameter case after only 2 seconds of heating. Considering that the total thermal damage obtained thus far by the nodes adjacent to the source temperature appears to be approximately $\Omega = 0.02$ (judging by Figure 4-14), and the difference in thermal damage between the cases is $\Omega = 0.002$ (i.e. 10%), then the slight difference in temperature produced by considering temperature-dependent tissue specific heat capacity and thermal conductivity can imply a reasonably faster rate of thermal damage.

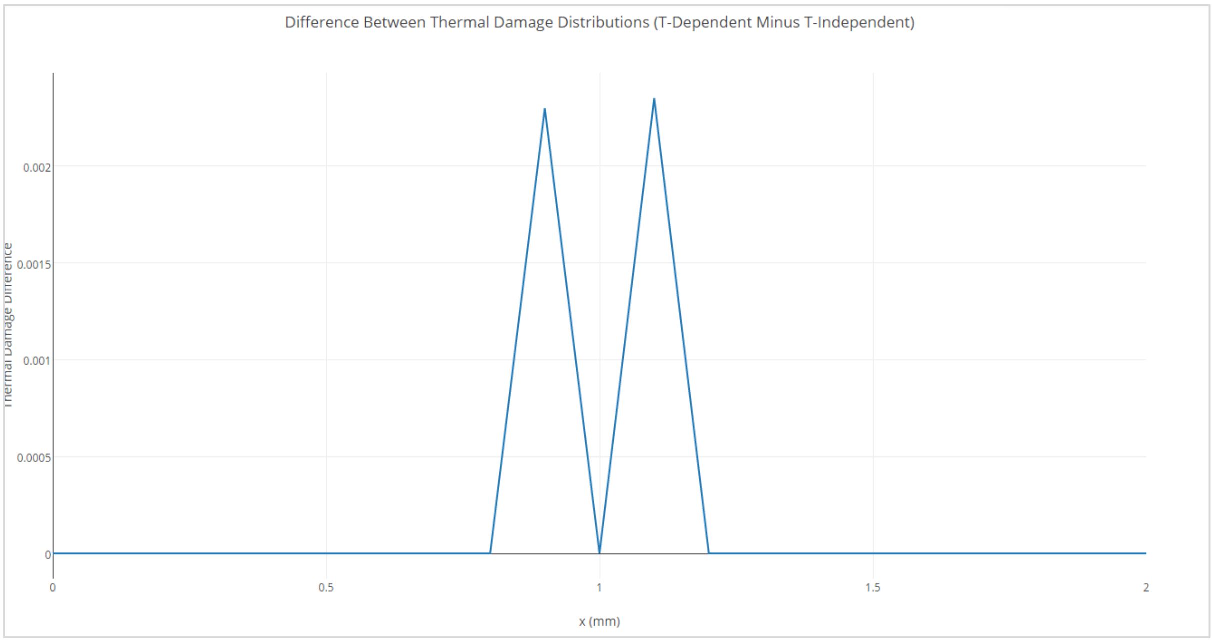


Figure 4-15: Difference between Thermal Damage Distributions (T-Dependent minus T-Independent)

4.2 Thermal Damage

4.2.1 Temperature Distribution

Simulation parameters for the results derived to analyse tissue thermal damage are as follows:

- Time step: 0.002 s
- Margin of error: 0.001
- Mesh resolution: 20×20×20 cubes: subdivided into 6 linear tetrahedra each
- Mesh element size: 0.1 mm (cube side length prior to subdivision)
- Source point temperature: 54 °C
- Duration of heating: 300 s. (Results presented at specific intervals)

The temperature distribution after 30 seconds is changing little if any; it has almost reached a steady state for this case. Nodes adjacent to the source temperature are above the threshold of 43 °C to receive thermal damage over a period of time.

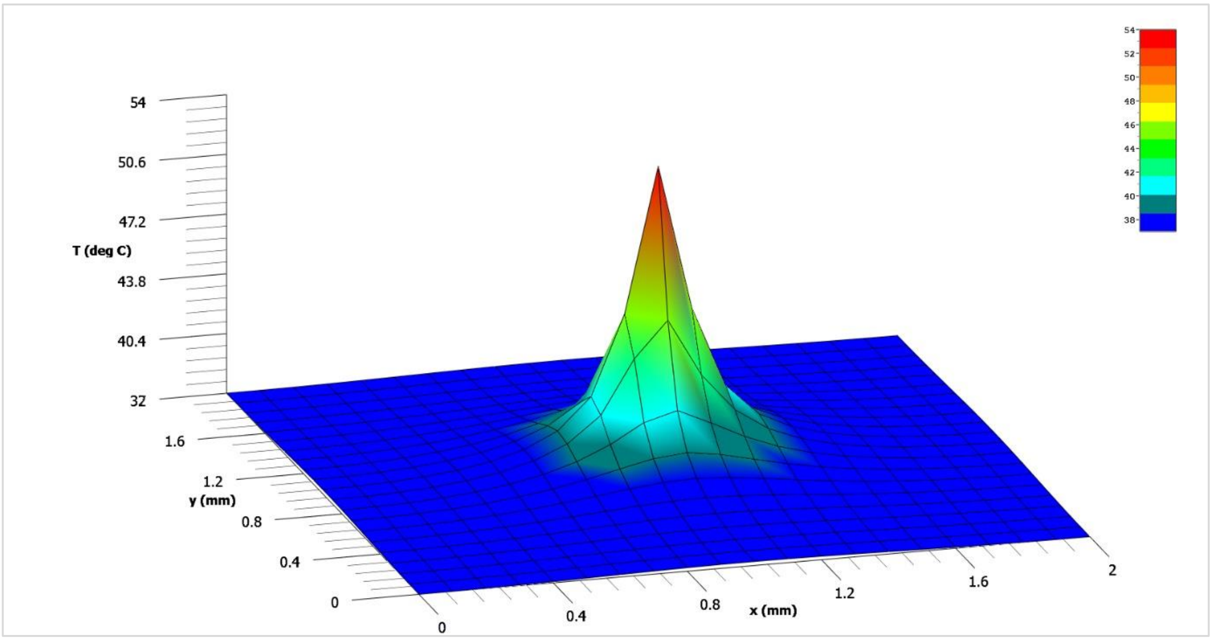


Figure 4-16: Temperature Distribution at 30s

4.2.2 Transient Thermal Damage

Due to the high falloff of temperature at a small distance from the point source temperature, the difference in temperature immediately at the source point and closely adjacent to it is quite significant. Consider Equation (2.2). As temperature increases, the rate of thermal damage rapidly increases. The result (as can be seen in Figure 4-18) is that the tissue at the immediate source temperature point necrotizes rapidly within 20 seconds, while immediately adjacent tissue has still not completely necrotized by five minutes. This indicates that while lower, safer treatment temperatures of 54 °C are able to rapidly necrotize tissue in a matter of seconds, it takes a number of minutes of continued application for adjacent tissue to necrotize, as discussed in section 2.5.3. The umbrella-like shape of the electrodes of the RFA needle instrument discussed in section 2.1.4 becomes more relevant. A large region of heat supply is necessary to raise temperatures of tissue in very close proximity to the instrument head. Higher treatment temperatures also imply that adjacent temperatures will be greater, increasing the coagulation/necrosis zone, and the rate at which the nearby tissue necrotizes. Figure 4-17 illustrates the time required to reach cellular necrosis ($\Omega = 4.6$) versus constant temperature applied during hyperthermia treatment. Refer to section 2.7.2.

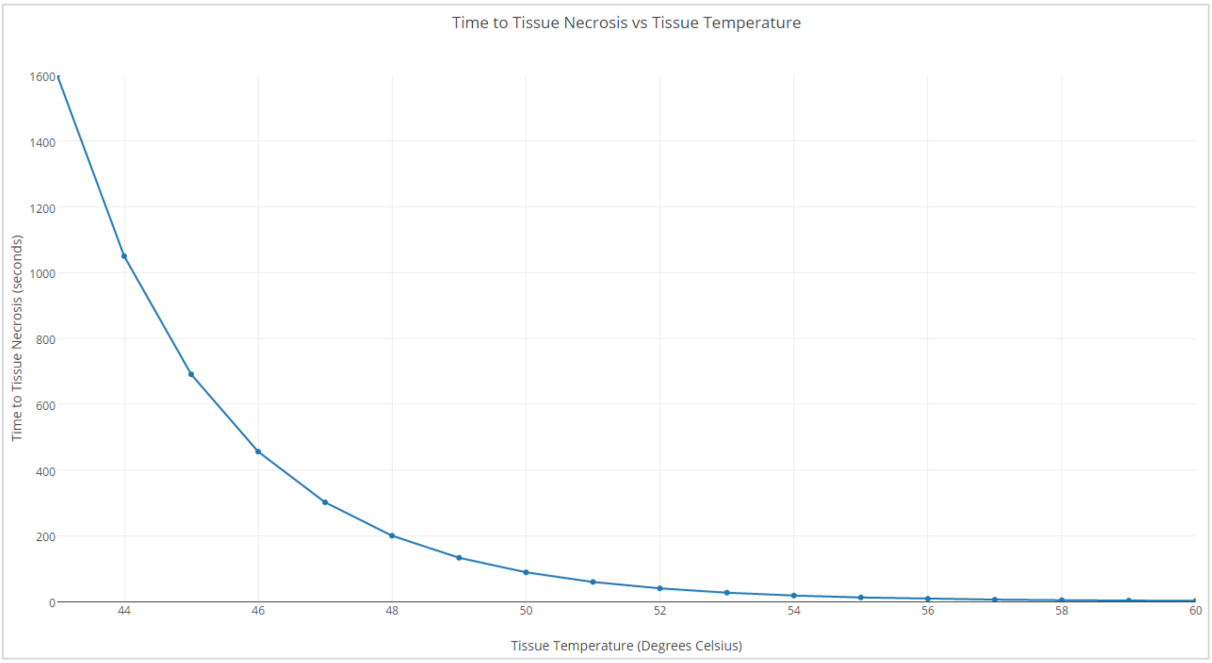


Figure 4-17: Time to Necrosis vs Tissue Temperature for Liver Tissue

Figure 4-19 shows the final thermal damage distribution for the tissue mid-plane. Note that two of the nodes diagonal to the source node, do not exceed 43 °C; possibly due to the non-symmetrical execution of shader groups during numerical solution, and thus thermal damage at these nodes is not activated.

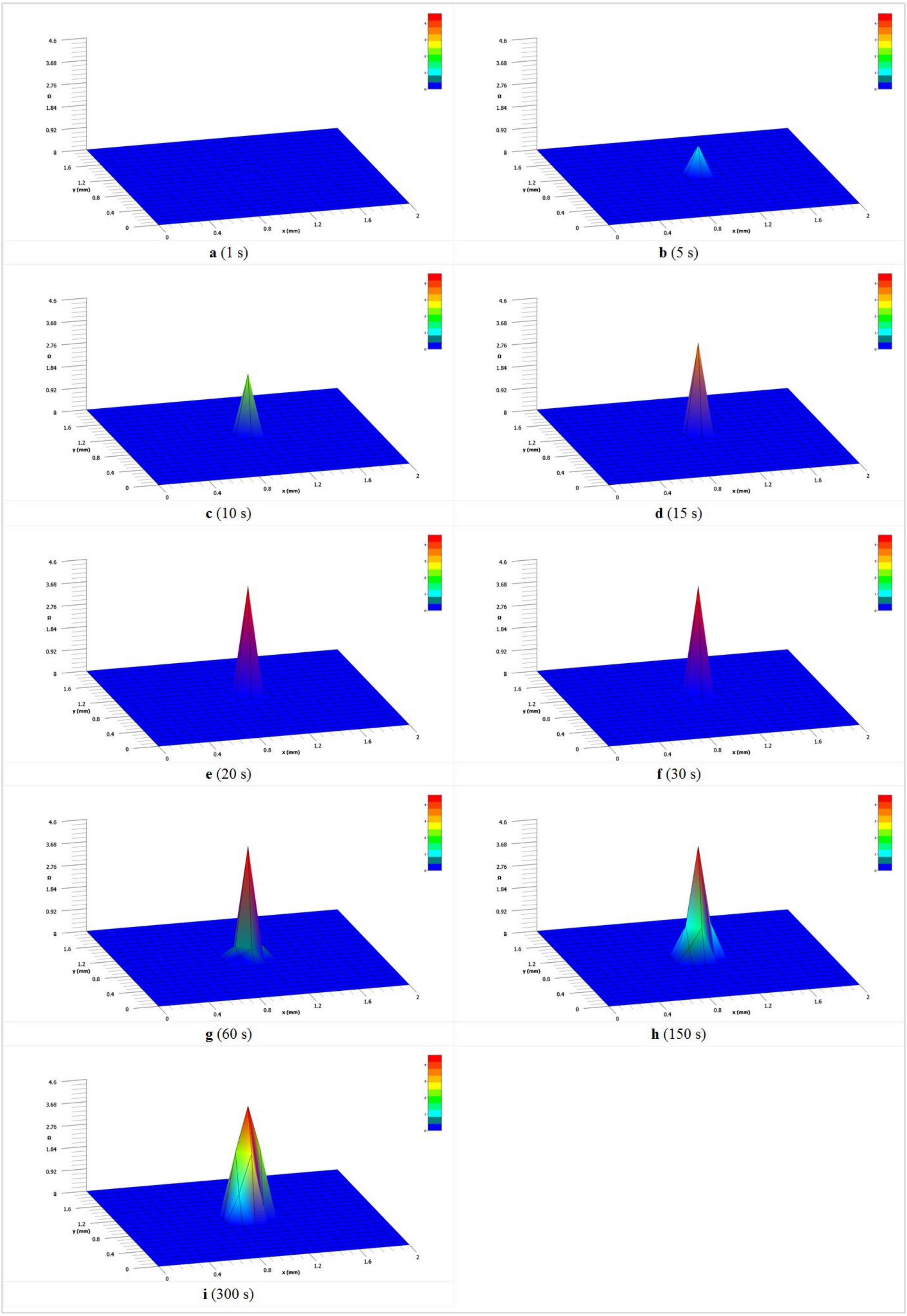


Figure 4-18: Transient Thermal Damage

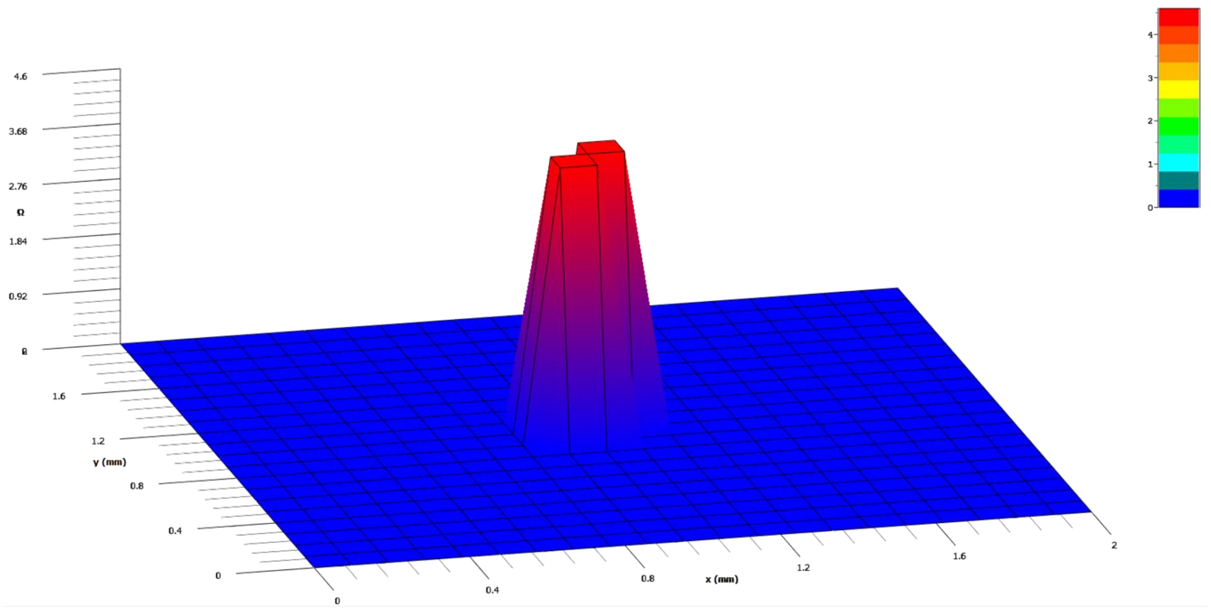


Figure 4-19: Thermal Damage Distribution (1200 s)

4.3 CPU vs GPU Comparison

4.3.1 CPU vs GPU Solution Times

In order to obtain computation times for simulation iterations on the CPU and GPU, simulations were operated at different mesh resolutions (varying global DoF), maintaining consistent parameters such as temperature-independent thermal parameters, margin of error, time step, and mesh element size/scale. Even though the margin of error can affect simulation duration due to influencing a greater number of numerical (Gauss-Seidel) solver iterations at a smaller margin (see Figure 3-11), the consistency of this parameter implies that the only other variable determining simulation duration is the mesh resolution (global DoF). For example, differing time steps or mesh element size/scale only affect the numerical output, not the computational solution process its self. Thus the following parameters were employed:

- Margin of error: 0.001
- Time step: 0.002 s
- Mesh element size: 0.5 mm

Despite the time step of 0.002 seconds, the same duration of simulation applies to a time step of 0.04 seconds (i.e. real-time), as explained above.

The duration for a single simulation iteration was evaluated on the GPU as the average of 1000 iterations for each case, while the duration on the CPU was evaluated as the average of 10 iterations, since each CPU iteration requires much longer to compute in comparison, and prove to be considerably consistent.

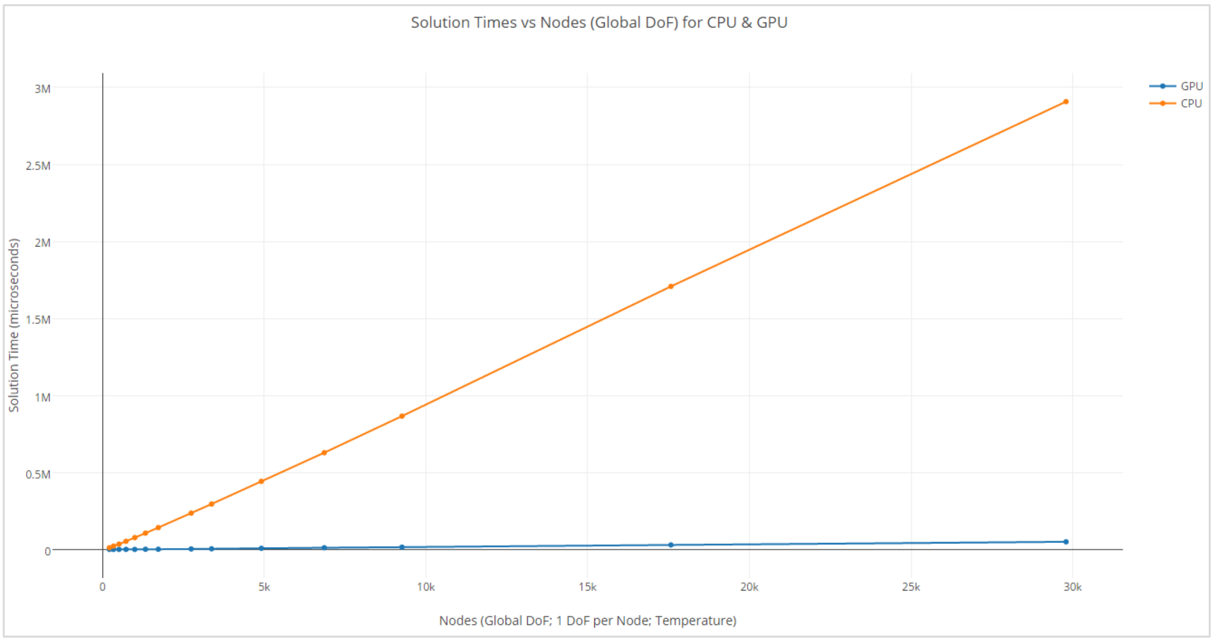


Figure 4-20: Solution Times vs Nodes (Global DoF) for CPU & GPU

Figure 4-20 shows a comparison of the average CPU solution time for a single simulation iteration vs that performed when accelerated by the GPU. Due to the highly parallelized processing nature of the GPU architecture and the parallelized implementation of the simulation upon it, the gradient of the relationship between solution time and global DoF on the GPU demonstrates increasingly greater efficiency and time savings as the mesh resolution increases.

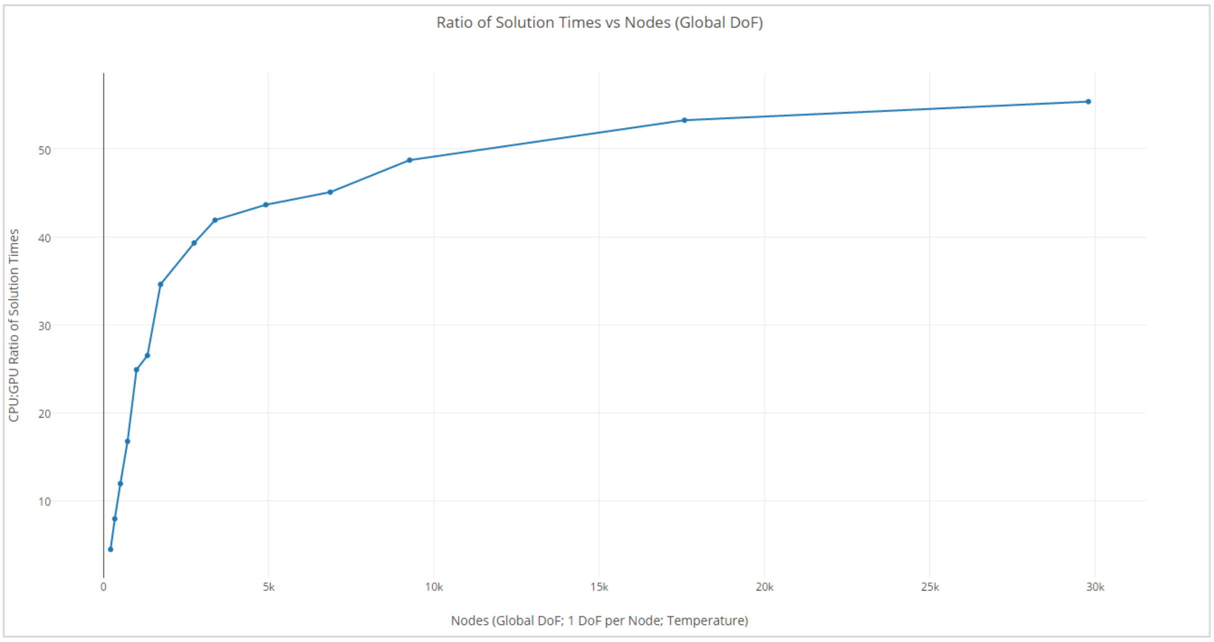


Figure 4-21: Ratio of Solution Times vs Nodes (Global DoF)

Figure 4-21 shows the ratio of solution times for bioheat transfer (and thermal damage) vs the number of nodes in the mesh (global DoF; 1 DoF per node – Temperature). The ratio is CPU:GPU; i.e. at 30,000 nodes, the GPU performs approximately 55.3 times faster than the CPU for the implementation of the simulation on the given hardware described in the methodology. The levelling-out of the relationship implies that at some large number of global DoF, the benefits of employing GPU acceleration will maximize. Even if the relationship continues to increase asymptotically, the performance gains may become insignificant. In actuality, the point at which the CPU:GPU solution times ratio begins to settle and tend asymptotically, coincides with what is dubbed the "saturation point" of the GPU hardware. In this case, not all of the GPU cores/streaming multiprocessors are utilized, (as: $\#cores - \#threads > 0$), until close to 3000 nodes/DoF in Figure 4-21, when the relative performance gains begin to diminish. The saturation point is reached when a number of threads (based on number of DoF/nodes) large enough to fully utilize the GPU hardware resources is dispatched. From this point onwards, there is still a benefit to using this solution implementation on the selected GPU hardware over the selected CPU hardware, however the benefit does not increase significantly after the saturation point.

This compares well with findings by Comas et al. (2008), as in Figure 2-3, where the benefits of GPU acceleration for real-time simulations increase dramatically at lower global DoF, and increase much more gradually at higher global DoF; after graphics hardware saturation.

4.3.2 GPU: Real-Time for Bioheat Transfer and Thermal Damage

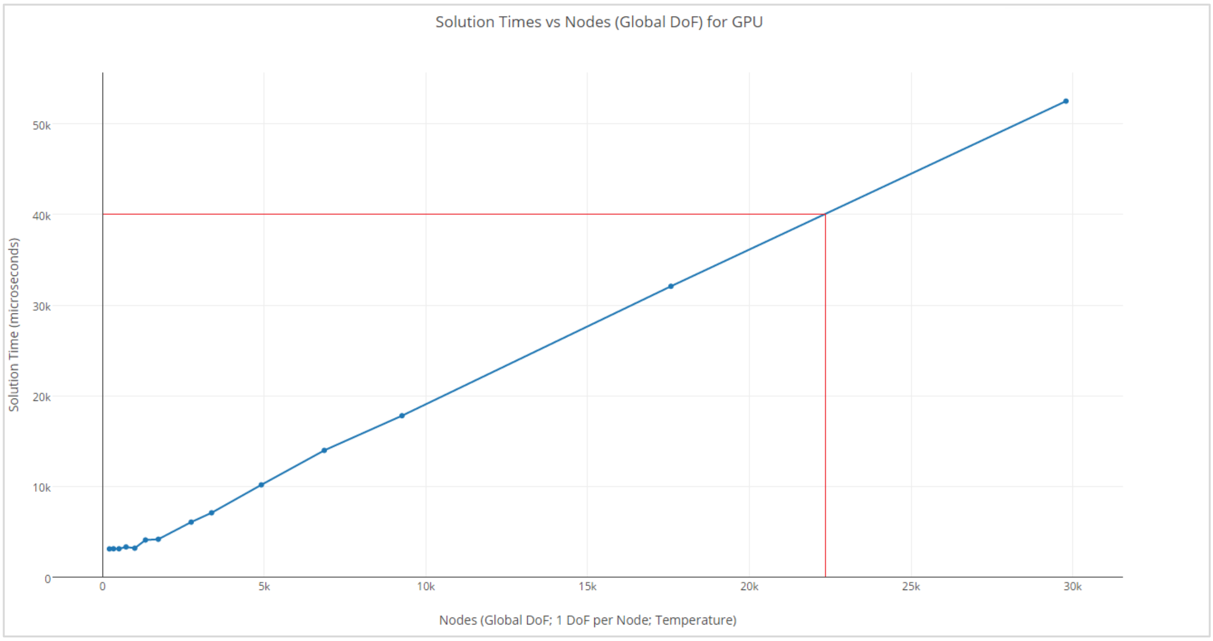


Figure 4-22: Solution Times vs Nodes (Global DoF) for GPU

Examining Figure 4-20, the considerably faster GPU solution times are difficult to evaluate in detail without being graphed on their own, as in Figure 4-22. On inspection here, the horizontal red line indicates the 40 millisecond (40,000 microsecond) mark. The intersection of this line with the GPU performance graph enables determination that approximately 22,000 nodes (global DoF) for bioheat transfer and thermal damage modelling will be solved fast enough to support real-time simulation, and thus close to 120,000 linear tetrahedra when considering the connectivity of the generated meshes (see section 3.4.3). After this number of DoF, greater numbers of global DoF will produce solution times greater than 0.04 seconds, and thus update frequencies of less than 25 Hz; insufficient for real-time simulation as defined in section 2.10.

Furthermore, it is predicted from these results that a temperature-dependent parameter scenario which requires multiple iterations will be limited to a much smaller maximum mesh resolution/global DoF for real-time simulation. Consider that for a temperature-dependent thermal parameter case, it is predicted that sufficient convergence of thermal parameters requires approximately 7 iterations (see Figure 3-11) for the given margin of error, thus a temperature-dependent parameter case should perform a single iteration in $\frac{1}{7} \times 40$ milliseconds. Actual simulation on the GPU for an average of 1000 steps for the temperature-dependent thermal parameter case indicated the following relationship shown in Figure 4-23. Examining the figure, the red line at 40 milliseconds indicates a mesh resolution of close to 3,000 nodes/global DoF.

Thus the resolution for real-time bioheat transfer and thermal damage for the temperature-independent thermal parameter case in the implementation undertaken in this research project is approximately 22,000 nodes/global DoF. For the temperature-dependent thermal parameter case, for the same implementation, the resolution is approximately 3,000 nodes/global DoF.

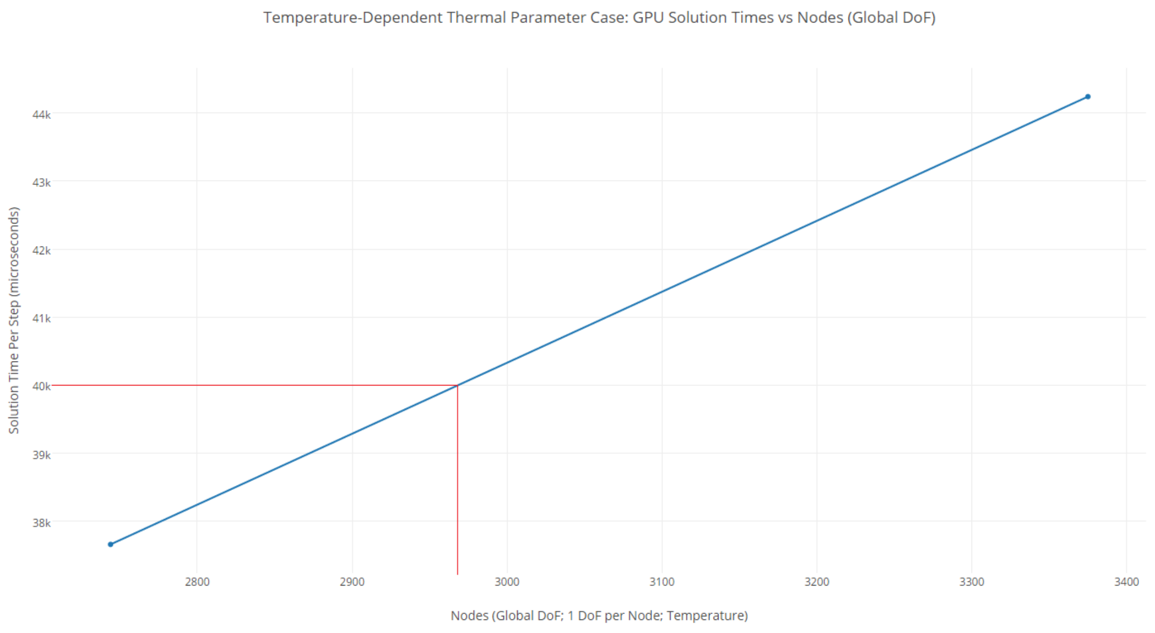


Figure 4-23: Temperature-Dependent Thermal Parameter Case: GPU Solution Times vs Nodes (Global DoF)

4.4 Bio-Thermal Stress-Strain

4.4.1 Temperature Distribution

As discussed in sections 3.4.3 and 3.5.2, the evaluation of thermal stress-strain takes place in third-party finite element software called LISA, which operates a 1300 node limit. This influenced the mesh resolution to be 10×10×8 cubes subdivided into 6 equally sized linear tetrahedra, or 11×11×9 nodes; for the sake of keeping the mesh as symmetrical as possible.

For modelling the initial temperature distribution, thermal parameters were maintained as temperature-independent. The time step was set to 0.002 seconds, and 1000 iterations were performed before obtaining the temperature distribution. A source temperature of 60 °C was applied to produce a slightly more diverse temperature gradient.

While the temperature distribution looks much larger for this case shown in Figure 4-24 as compared to Figure 4-16, note that the source temperature for this case is greater, thereby increasing the region of the coagulation/necrosis zone. The zone for potential thermal damage ($T \geq 43$ ℃) appears to extend very little over 0.5 mm from the source temperature location.

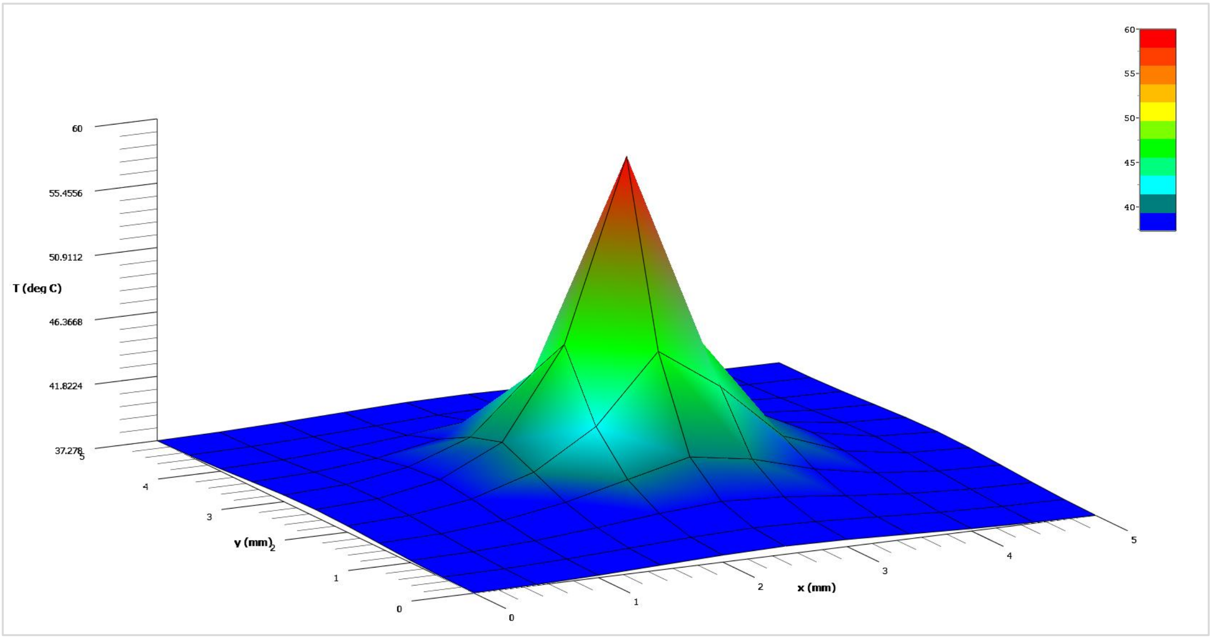


Figure 4-24: Temperature Distribution Inducing Thermal Stress/Strain

4.4.2 State of Strain

After exporting the tissue temperature distribution from the simulation application, and importing it into third party finite element analysis software (LISA), the temperature field was applied as thermal loading to the mesh structure, to influence the thermomechanical behaviour of the tissue. The outer boundary of the tissue block was constrained as a fixed support, and the thermal stress distributions calculated. Since LISA provided no option for viewing thermal strains, the thermal strain distribution for normal and shear strains was calculated backwards from the stresses. This was achieved, considering that LISA calculated a linear elastic state of stress based on a given elastic modulus parameter and Poisson's Ratio (see section 3.3.1 for both quantities), thus the state of strain was calculated for normal strains (xx, yy, zz) from Hooke's Law:

$$\sigma = E\varepsilon. \tag{4.1}$$

Furthermore, the shear strains (xy, yz, zx) were calculated backwards from the shear stresses, using Hooke's Law in shear, where:

$$\tau = G\gamma \tag{4.2}$$

where the shear modulus of elasticity *G* is related to the modulus of elasticity *E* and the Poisson's Ratio ν, by:

$$G = \frac{E}{2(1+\nu)}. \tag{4.3}$$

Examining Figure 4-25, Figure 4-26, Figure 4-27, Figure 4-28, Figure 4-29, and Figure 4-30, the resultant normal and shear strains due to the thermal loading produced by Pennes bioheat transfer for RFA application are shown for the xy mid-plane of the tissue block. Note how small these compressive thermal strains are, at most, a fraction of 1%, that is, thermally induced displacements are occurring on the scale of micrometres; considerably insignificant when compared to typical displacements induced by surgical instruments during the average surgical procedure.

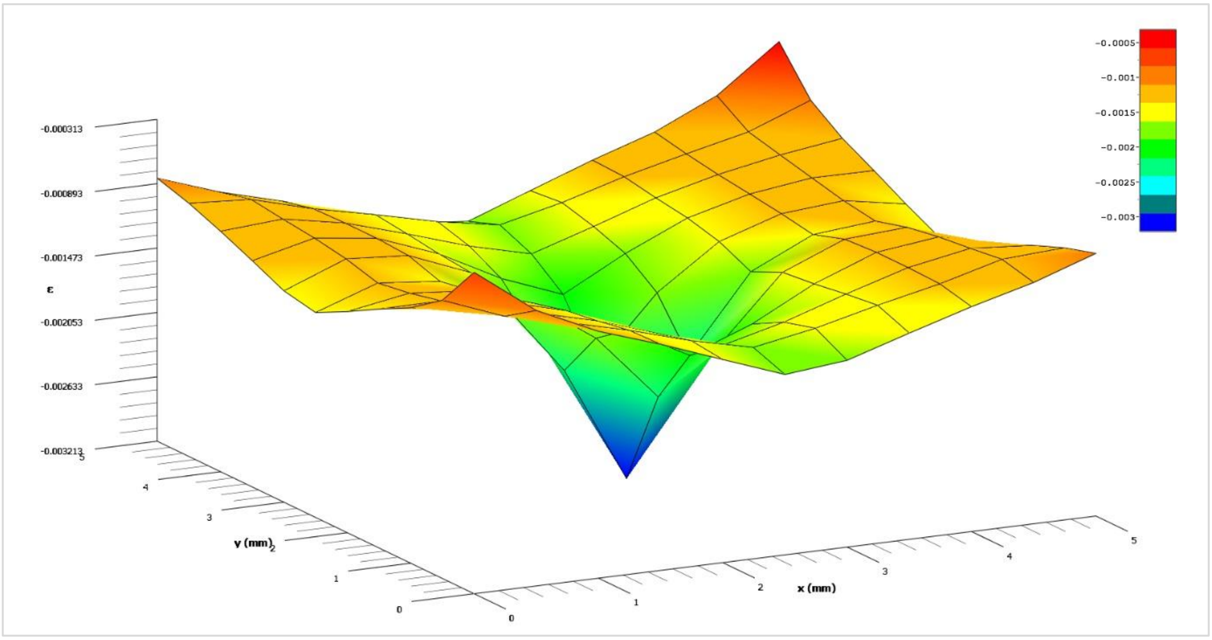


Figure 4-25: Tissue Mid-plane Normal Strain (xx)

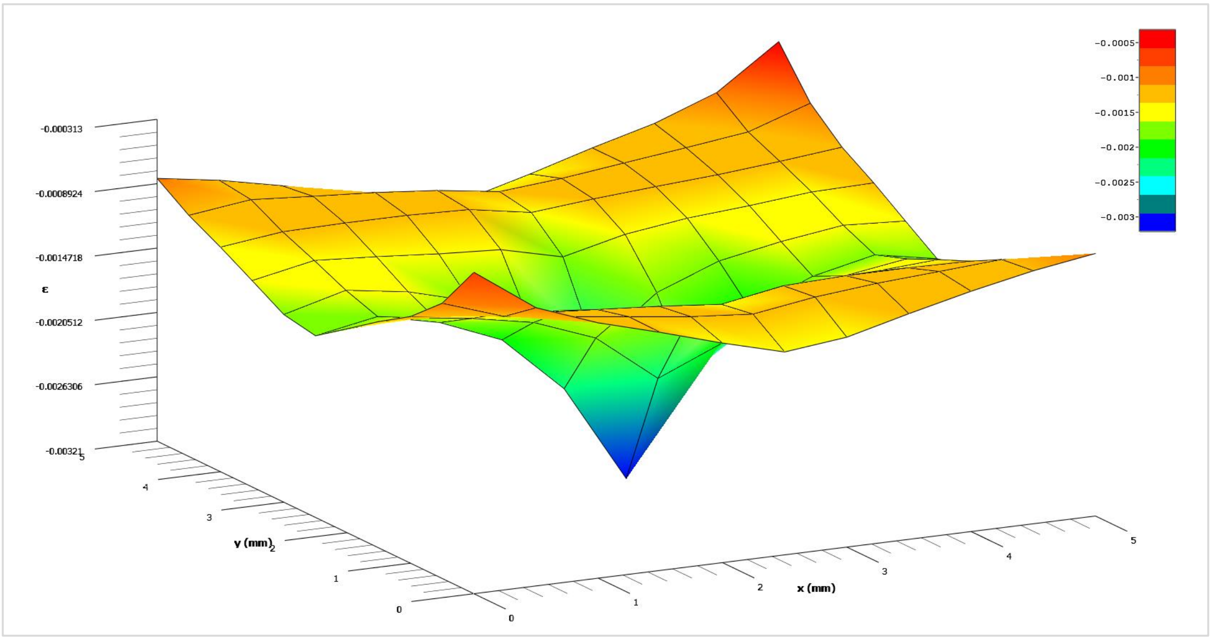


Figure 4-26: Tissue Mid-plane Normal Strain (yy)

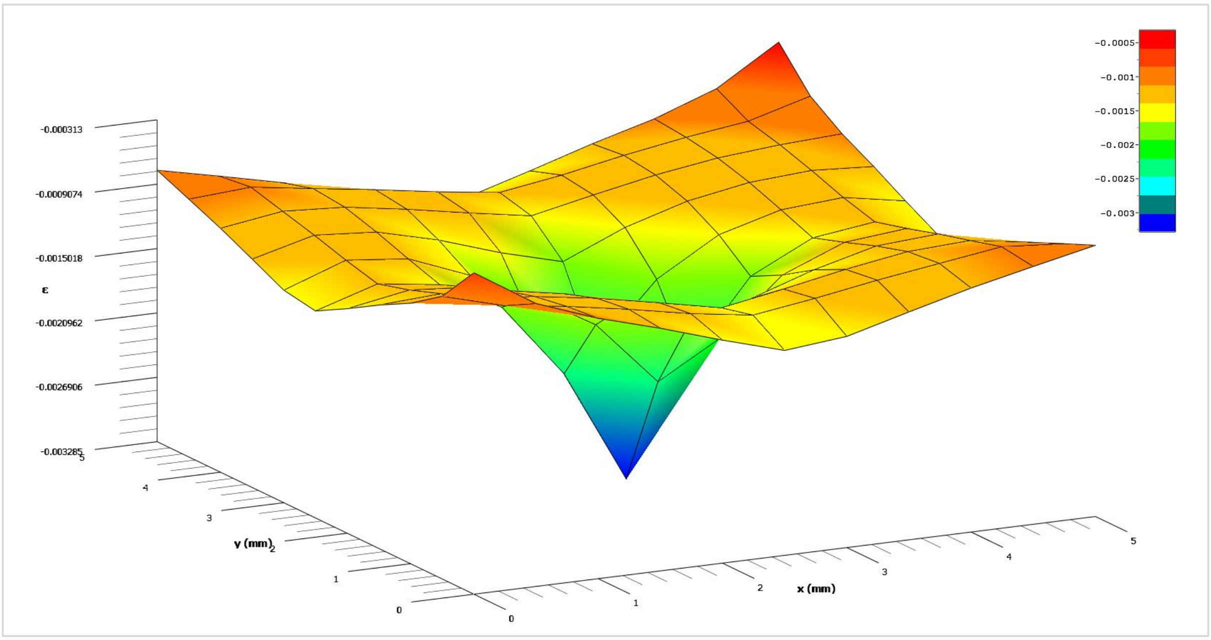


Figure 4-27: Tissue Mid-plane Normal Strain (zz)

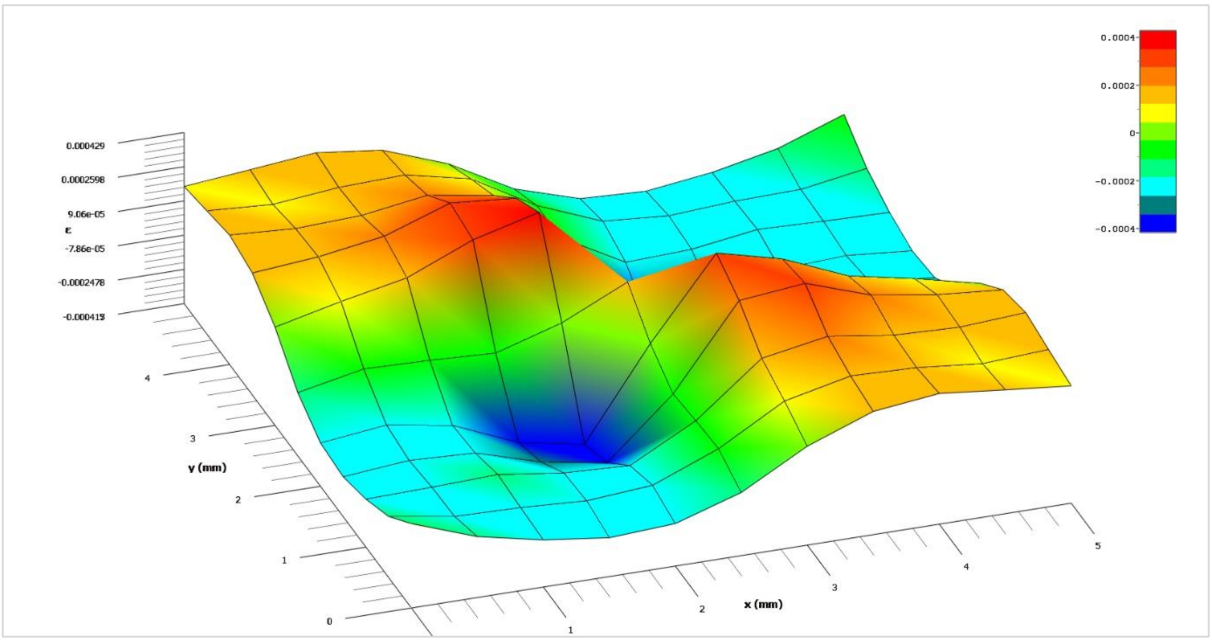


Figure 4-28: Tissue Mid-plane Shear Strain (xy)

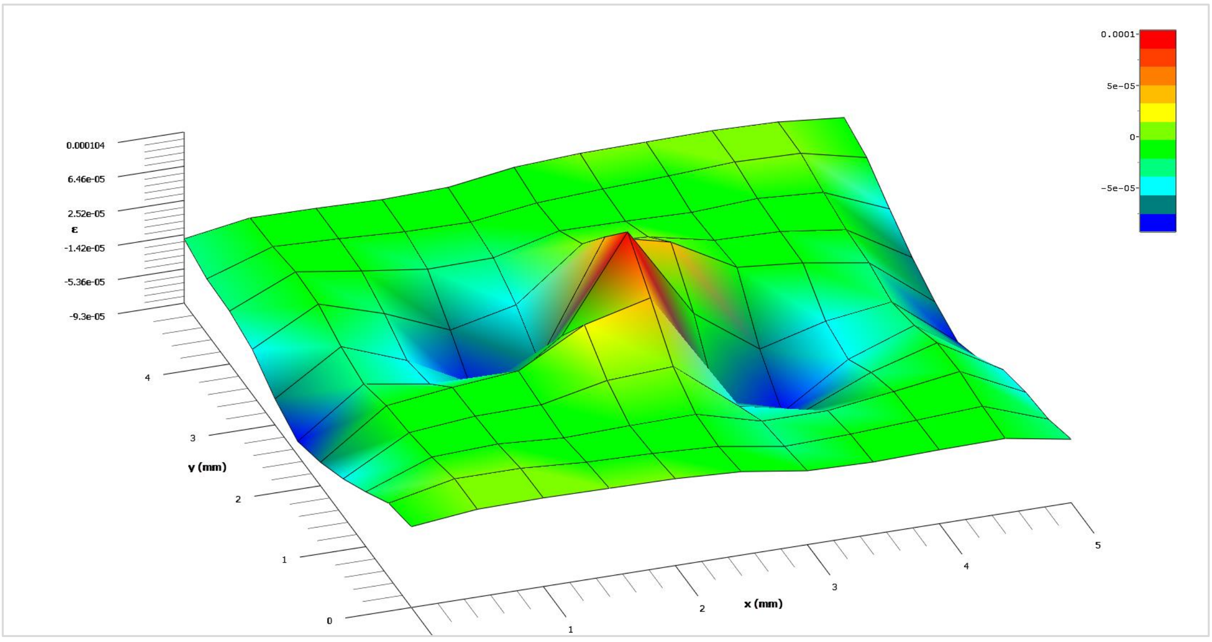


Figure 4-29: Tissue Mid-plane Shear Strain (yz)

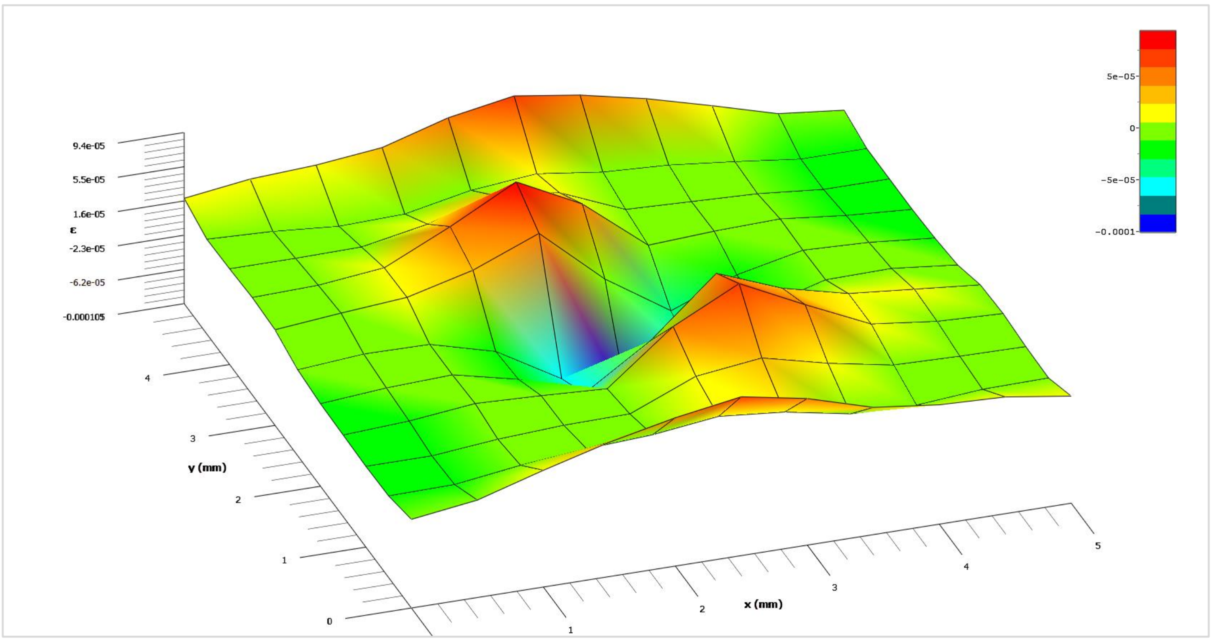


Figure 4-30: Tissue Mid-plane Shear Strain (zx)

The negative sign of the normal strains indicates compression in all three major axes, as expected. The tissue is heated by the instrument, and attempts to expand when no space is available to expand into. Thus the tissue remains in a compressed state. The shear strains shown in Figure 4-28 for the xy plane show the tissue block's tendency towards deforming to an oblique shape in the xy plane; a pair of positive strains at opposing corners, and a pair of negative strains also at opposing corners.

4.4.3 Temperature-Independent Linearly Elastic Stress-Strain: State of Stress

The state of stress for the linearly elastic case of a temperature-independent stress-strain relationship yields normal and shear stresses as expected in the state of strain in section 4.4.2. The modulus of elasticity used is as defined in section 3.3.1:

$$E = 152.5\ kPa$$

Compressive normal stresses are found in all three axes, peaking at the location of the highest temperature: the point source. The magnitude of these normal stresses is miniscule; a fraction of a kilopascal, considering the typical mechanical operational loading noted in section 2.5.3 is approximately 470 kPa. Note that furthermore, the shear strains are so small that their magnitude in the xy plane is little more than 20 Pa.

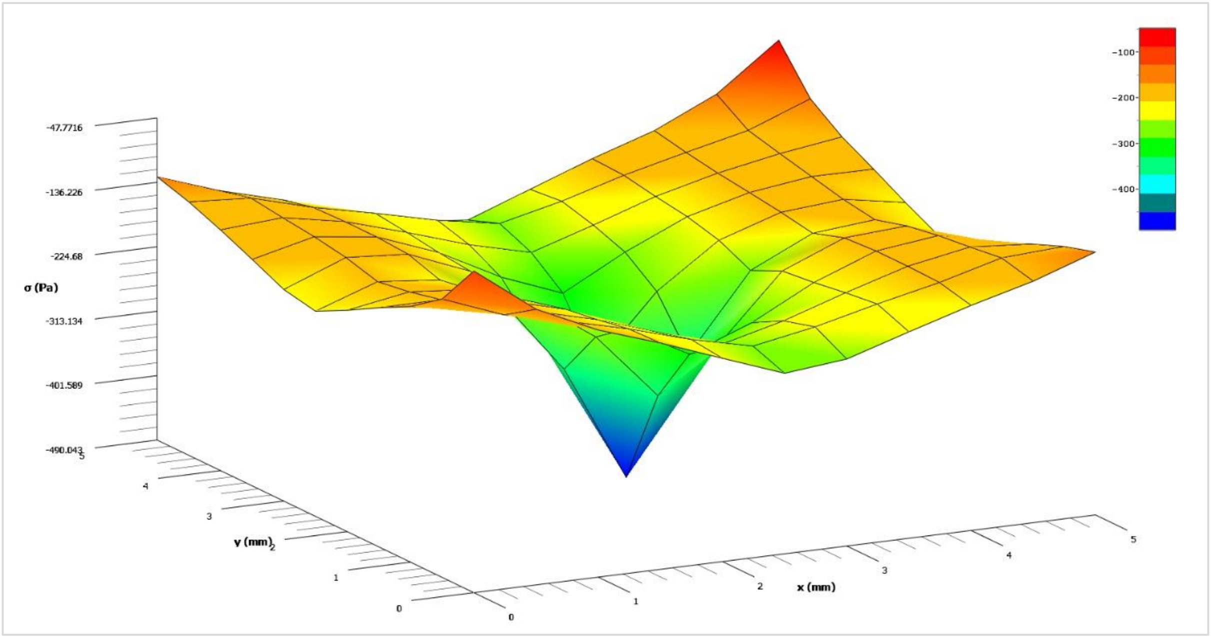


Figure 4-31: Temperature-Independent Linearly Elastic Stress-Strain Tissue Mid-plane Normal Stress (xx)

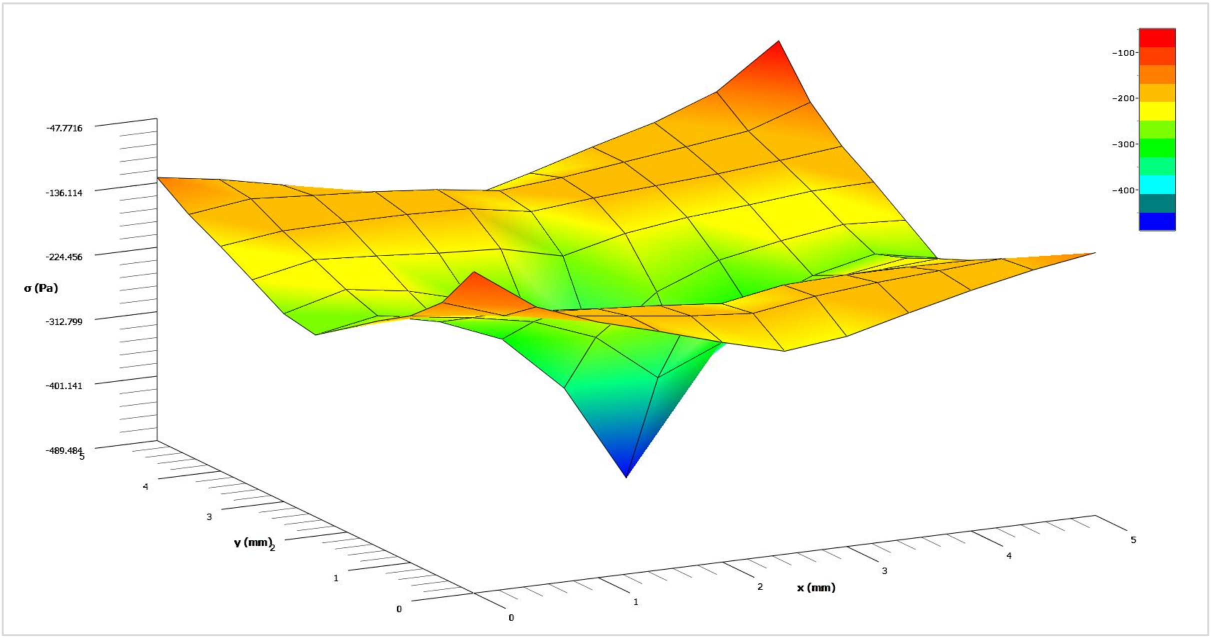


Figure 4-32: Temperature-Independent Linearly Elastic Stress-Strain Tissue Mid-plane Normal Stress (yy)

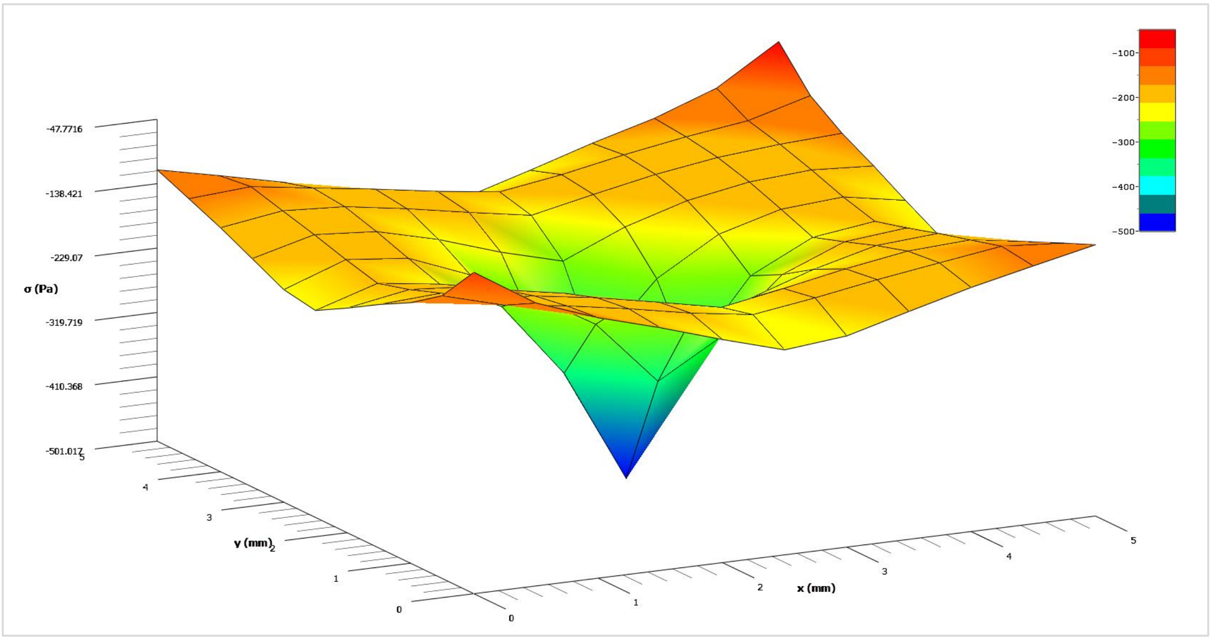


Figure 4-33: Temperature-Independent Linearly Elastic Stress-Strain Tissue Mid-plane Normal Stress (zz)

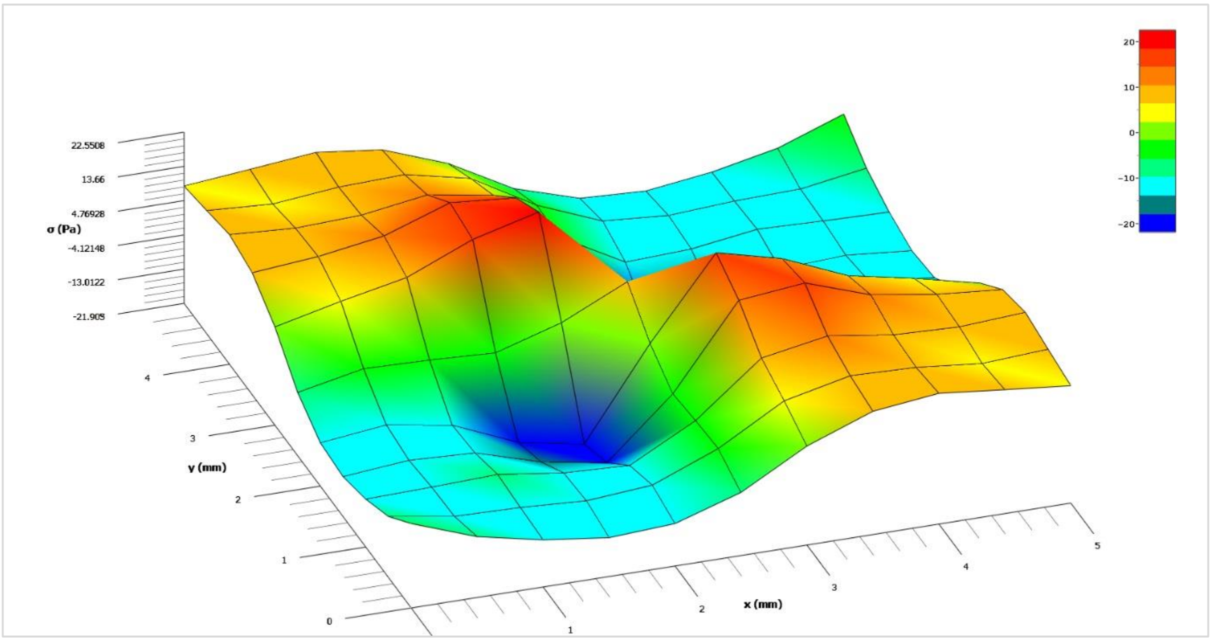


Figure 4-34: Temperature-Independent Linearly Elastic Stress-Strain Tissue Mid-plane Shear Stress (xy)

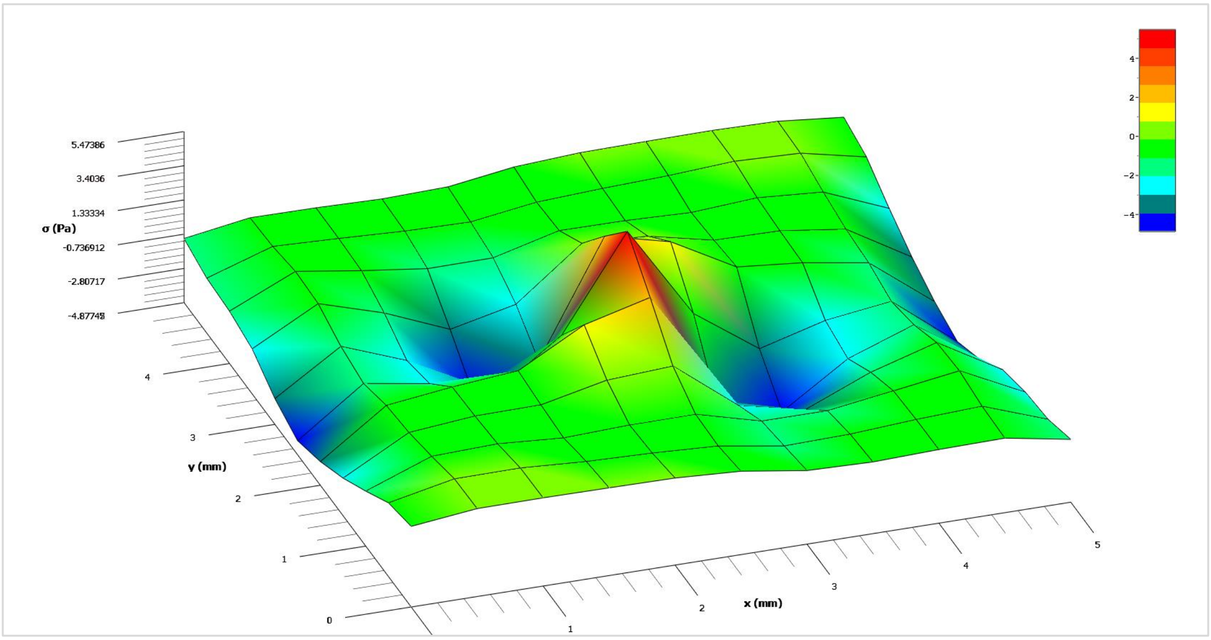


Figure 4-35: Temperature-Independent Linearly Elastic Stress-Strain Tissue Mid-plane Shear Stress (yz)

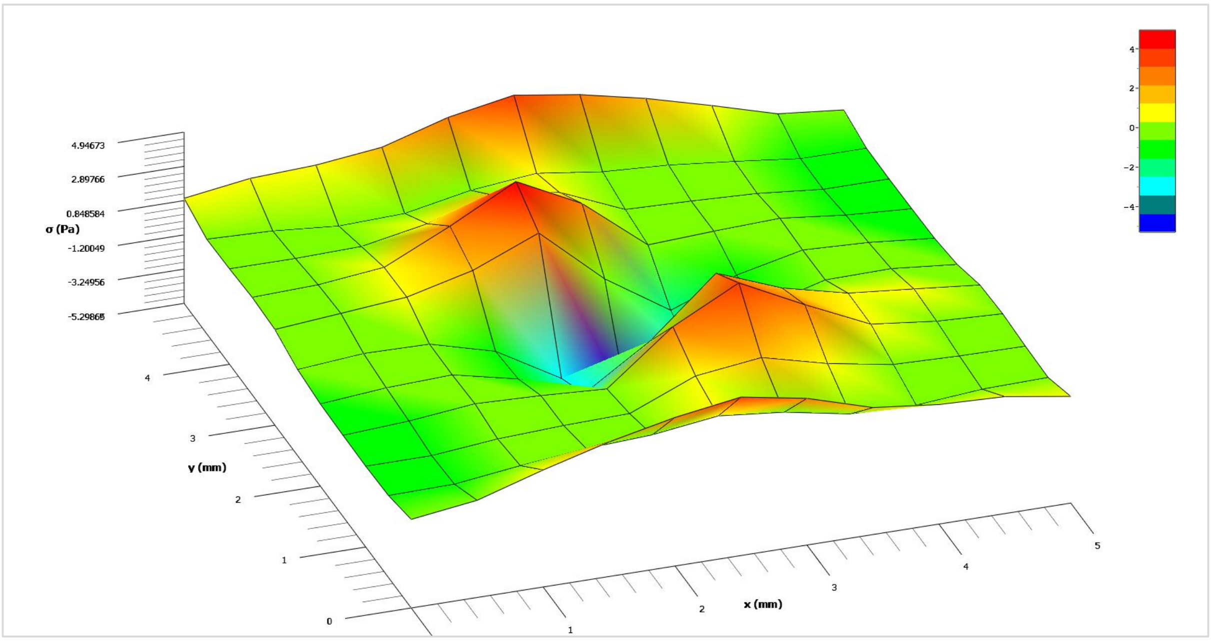


Figure 4-36: Temperature-Independent Linearly Elastic Stress-Strain Tissue Mid-plane Shear Stress (zx)

4.4.4 Temperature-Independent Non-Linearly Elastic Stress-Strain: Normal Stresses

The normal stresses for the case of nonlinearly elastic stress-strain (Equation (3.85) and coefficients for the 37 °C curve from Table 3.13) demonstrate a significant percentage decrease in compressive stresses, when the nonlinearity is accounted for.

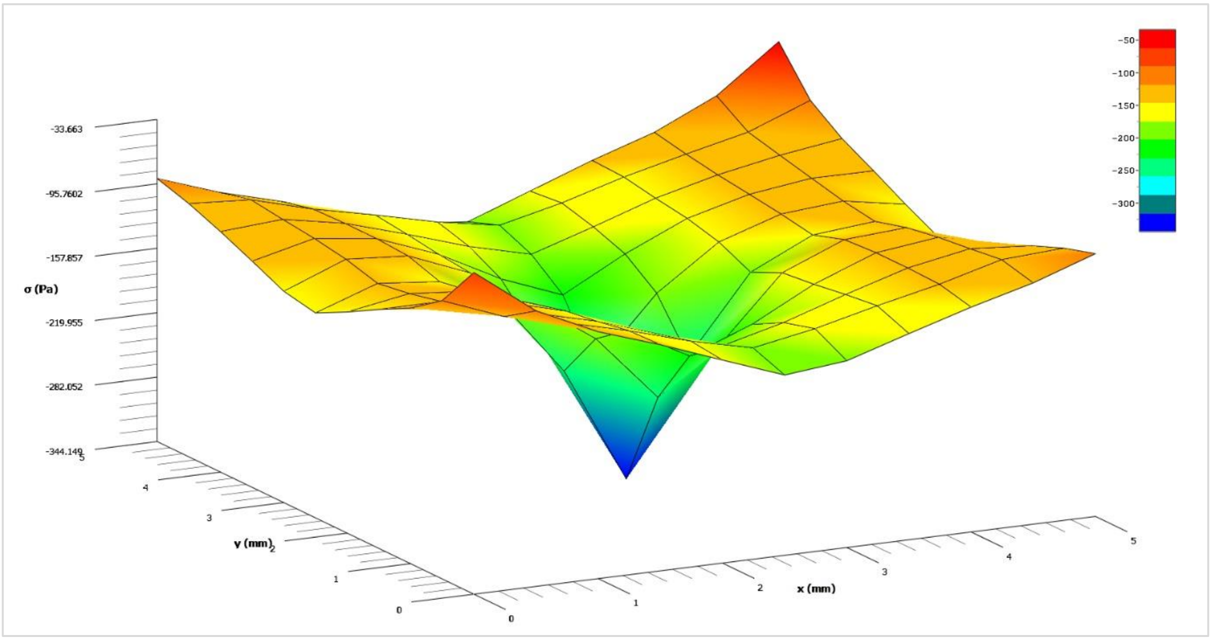


Figure 4-37: Temperature-Independent Nonlinearly Elastic Stress-Strain Tissue Mid-plane Normal Stress (xx)

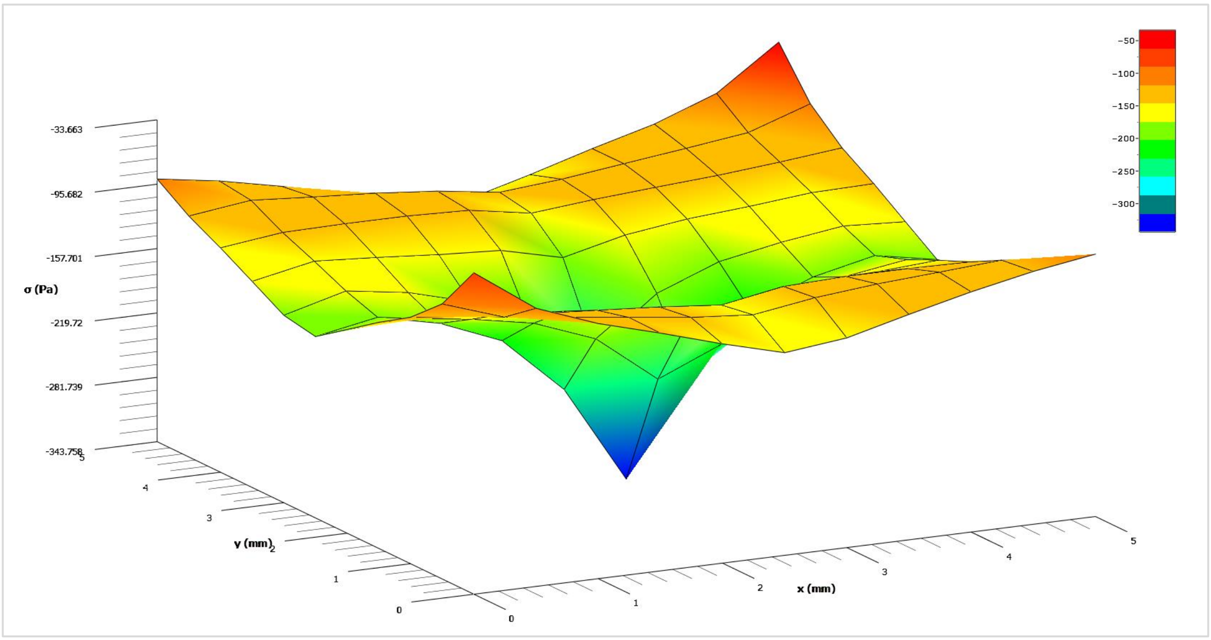


Figure 4-38: Temperature-Independent Nonlinearly Elastic Stress-Strain Tissue Mid-plane Normal Stress (yy)

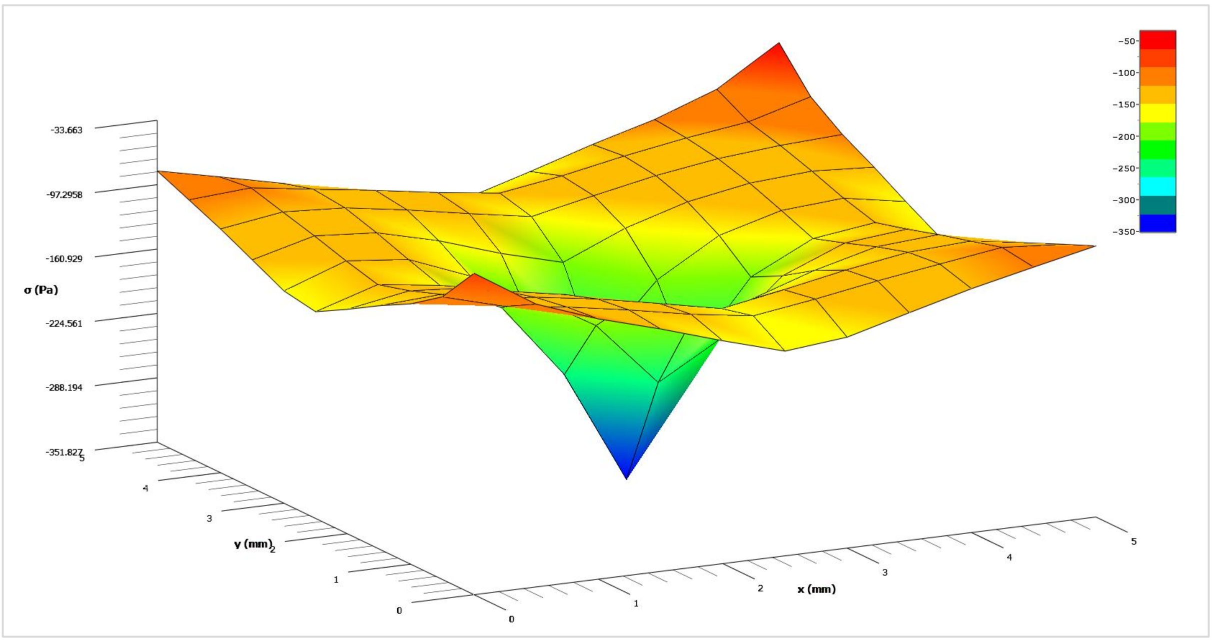


Figure 4-39: Temperature-Independent Nonlinearly Elastic Stress-Strain Tissue Mid-plane Normal Stress (zz)

4.4.5 Temperature-Dependent Non-Linearly Elastic Stress-Strain: Normal Stresses

A very significant decrease in compressive normal stresses occurs in the tissue when the temperature dependence of the stress-strain relationship is accounted for (Equation (3.85) and coefficients from Table 3.13). It appears that through the x and y axes (and one can deduce, the same through the z axis, due to the homogeneity of the material) the normal stresses are almost constant. The greater compressive thermal strains induced at the centre of the tissue block are at the same time increased due to the high temperature, while the tissue's lesser mechanical response due to higher temperatures (see Figure 2-14 and Figure 2-15) decreases the resultant stresses. Thus, the troughs (as opposed to valleys) of compressive stresses shown in Figure 4-40 and Figure 4-41 demonstrate a very consistent normal stress along each axis that is distributed evenly due to tissue bio-thermomechanical behaviour.

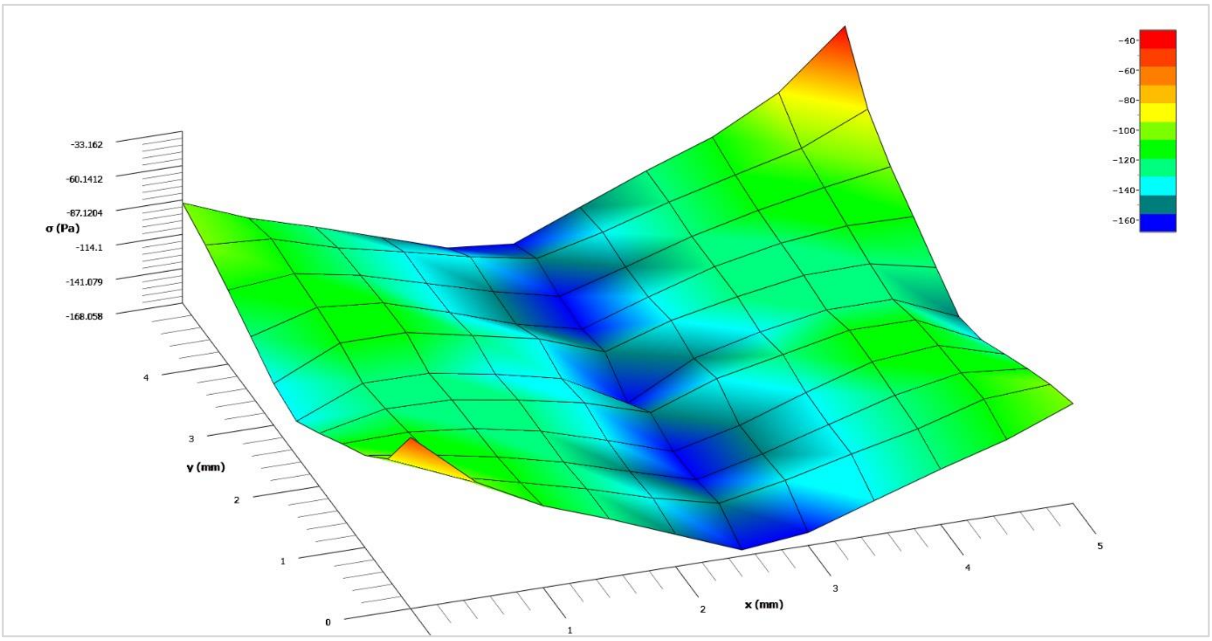


Figure 4-40: Temperature-Dependent Nonlinearly Elastic Stress-Strain Tissue Mid-plane Normal Stress (xx)

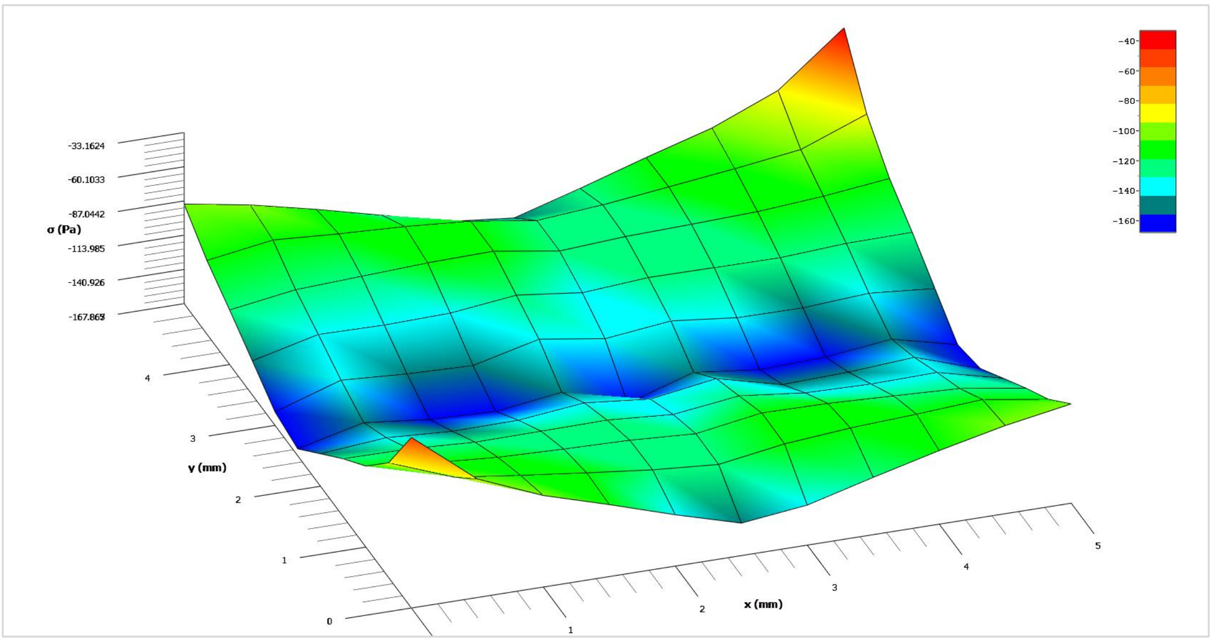


Figure 4-41: Temperature-Dependent Nonlinearly Elastic Stress-Strain Tissue Mid-plane Normal Stress (yy)

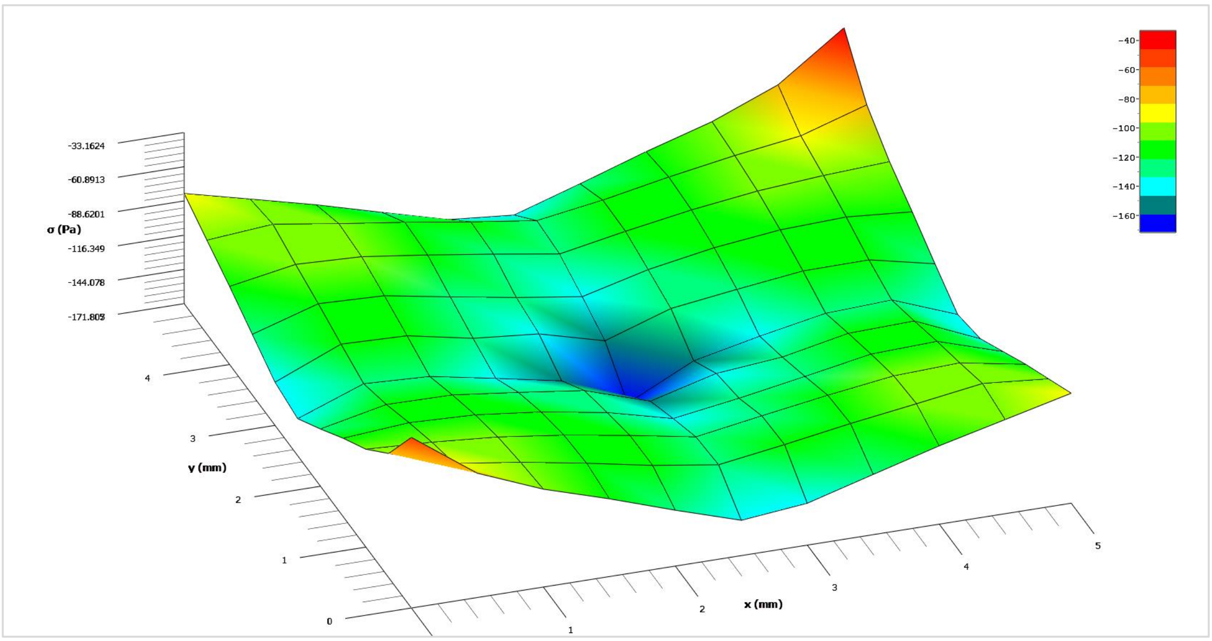


Figure 4-42: Temperature-Dependent Nonlinearly Elastic Stress-Strain Tissue Mid-plane Normal Stress (zz)

Examining Figure 4-43 and Figure 4-44, the evolution of the thermal stress derived from the temperature-dependent nonlinearly elastic stress-strain relationship can be observed, from a single iteration (1/25 s) up until 30 seconds. The thermal stresses at the nodal locations along the x-axis centreline of the x-y mid-plane are plotted at different times. The general cross-sectional shape of the 3D plot in Figure 4-40 can be appreciated, although for this transient example, a heat source of 54 °C was applied, thus the maximum compressive stress at the source location is less than that of Figure 4-40.

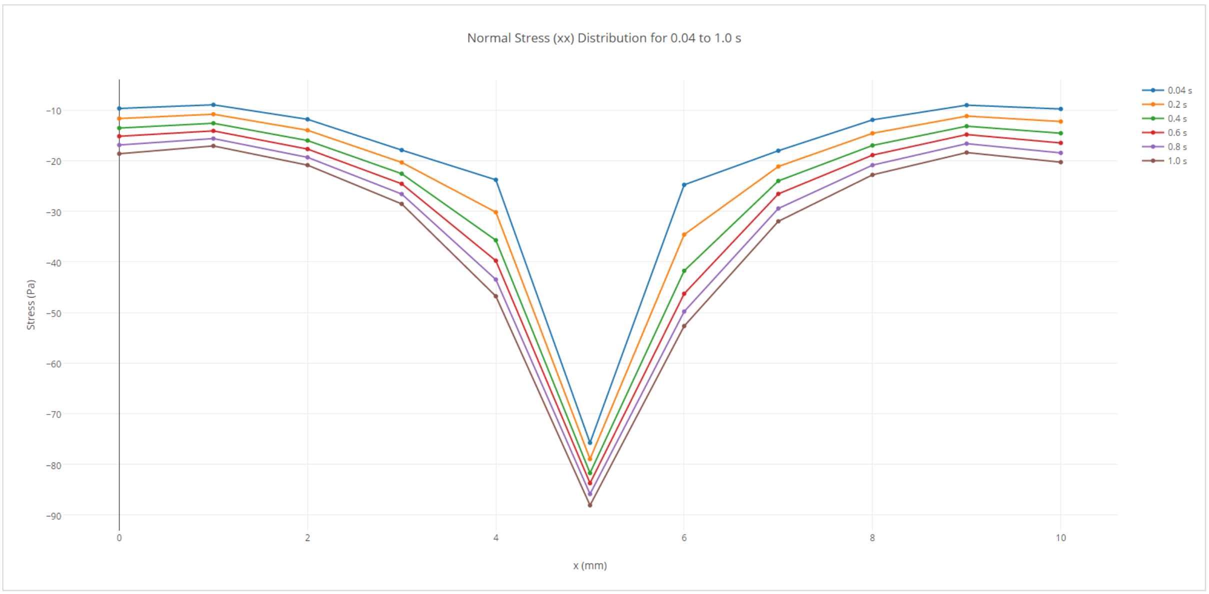


Figure 4-43: Thermal Stress Distribution from 0.04 s to 1.0 s

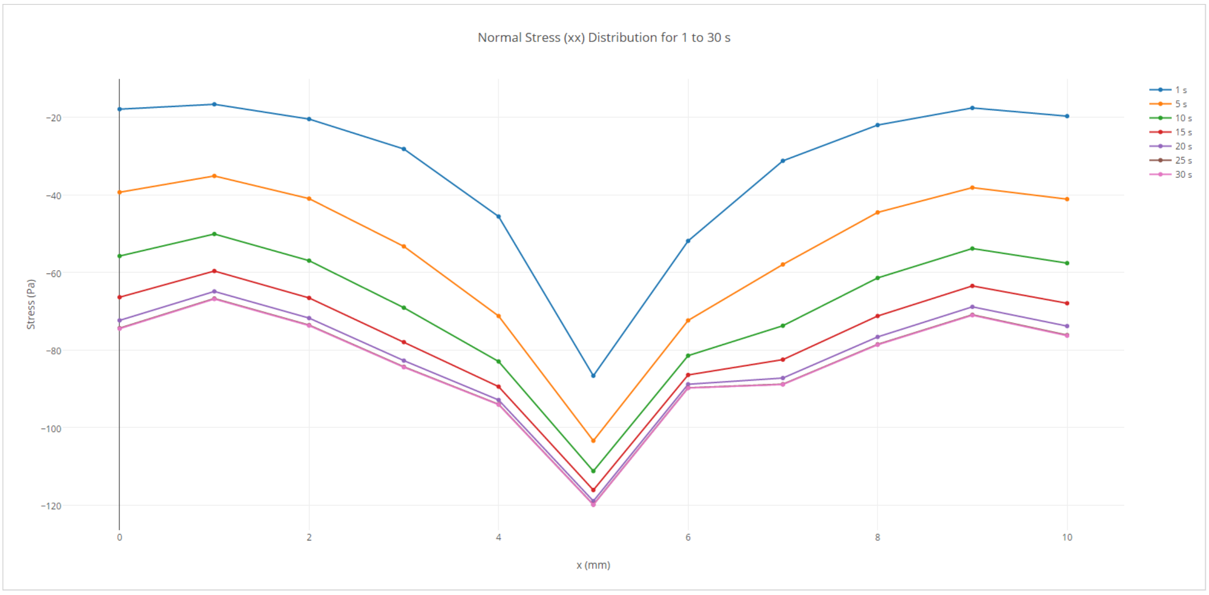


Figure 4-44: Thermal Stress Distribution from 1 s to 30 s

4.4.6 Normal Stress Comparison

When the xx, yy and zz normal stresses are compared along the x axis (centre of the xy mid-plane) for the three scenarios:

- Temperature-independent linear elasticity (blue)
- Temperature-independent nonlinear elasticity (orange)
- Temperature-dependent nonlinear elasticity (green)

It can be found that the greatest compressive stresses for the temperature-dependent nonlinear elastic stress-strain case are approximately 34%, or one third of magnitude of the stresses expected at the source location when temperature-dependence of stress-strain is not considered and linear elasticity is assumed. When compared to the temperature-independent case of nonlinearly elastic stress-strain, the stresses from the temperature-dependent stress-strain case are still approximately 50% or one half of the magnitude of the temperature-independent case, or rather the temperature-independent stress-strain cases differ to the temperature-dependent stress-strain one by 200-300%.

As discussed in section 4.4.5, the compressive thermal stresses for the temperature-dependent stress-strain case appear consistent; almost constant, throughout each axis, in comparison to the stress distributions for the temperature-independent stress-strain cases.

These results imply that as expected, the collagen of the tissue reduces in its capacity to bear loads at greater temperatures, thus the elastin component of the liver ECM becomes the main load-bearing contributor. It is to be expected that for a case where mechanical forces such as operational loads (see section 2.5.3) are applied, the percentage of difference between the temperature-dependent stress-strain case and the temperature-independent stress-strain cases will only continue to be magnified (see Figure 2-14 and Figure 2-15). Considering the magnitude of the vast difference in distributions when accounting solely for thermal stresses, the implications are that hyperthermia simulations should definitely account for temperature-dependence of the stress-strain relationship in biological tissue, as it very heavily impacts the resultant stresses and their distributions.

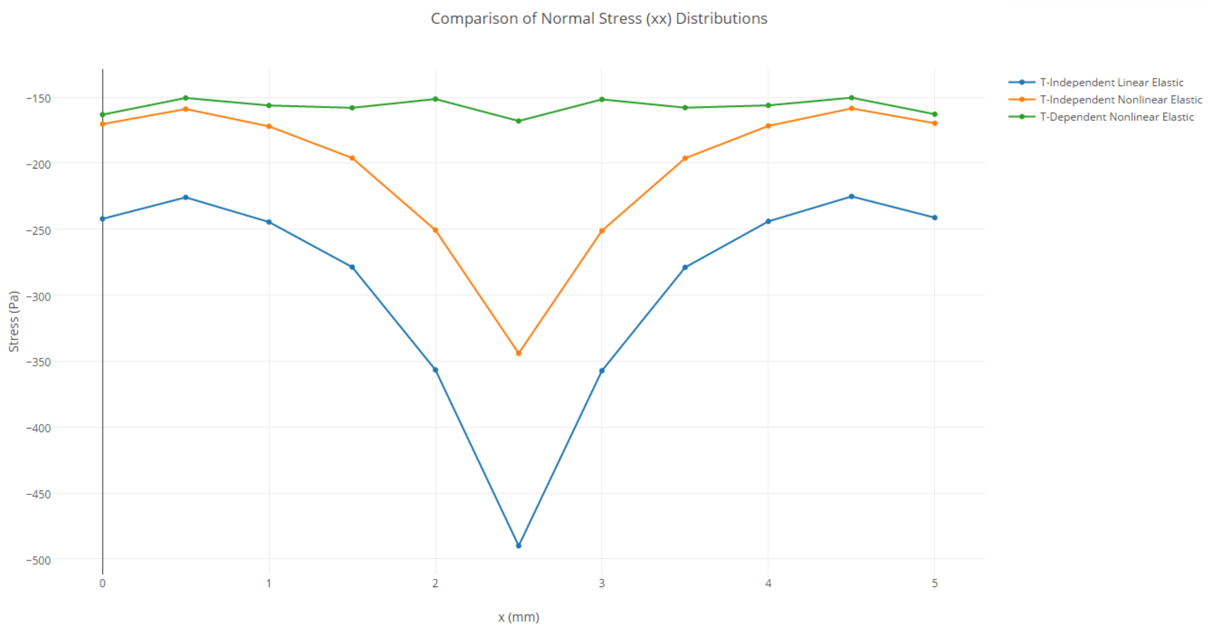


Figure 4-45: Comparison of Normal Stress (xx) Distributions

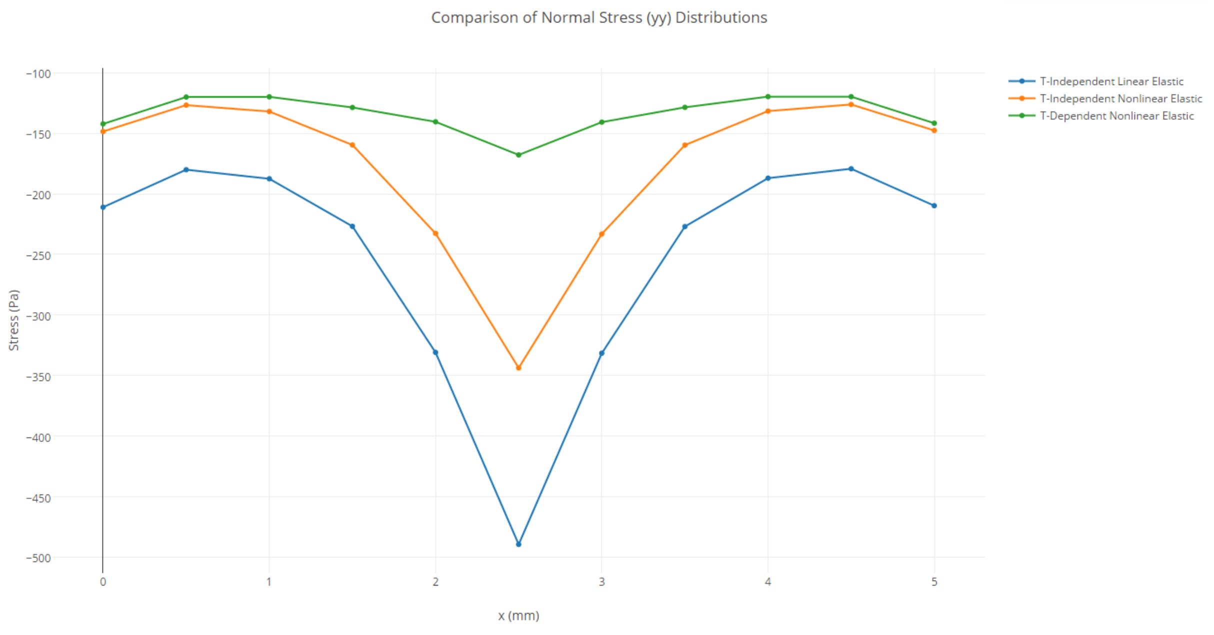


Figure 4-46: Comparison of Normal Stress (yy) Distributions

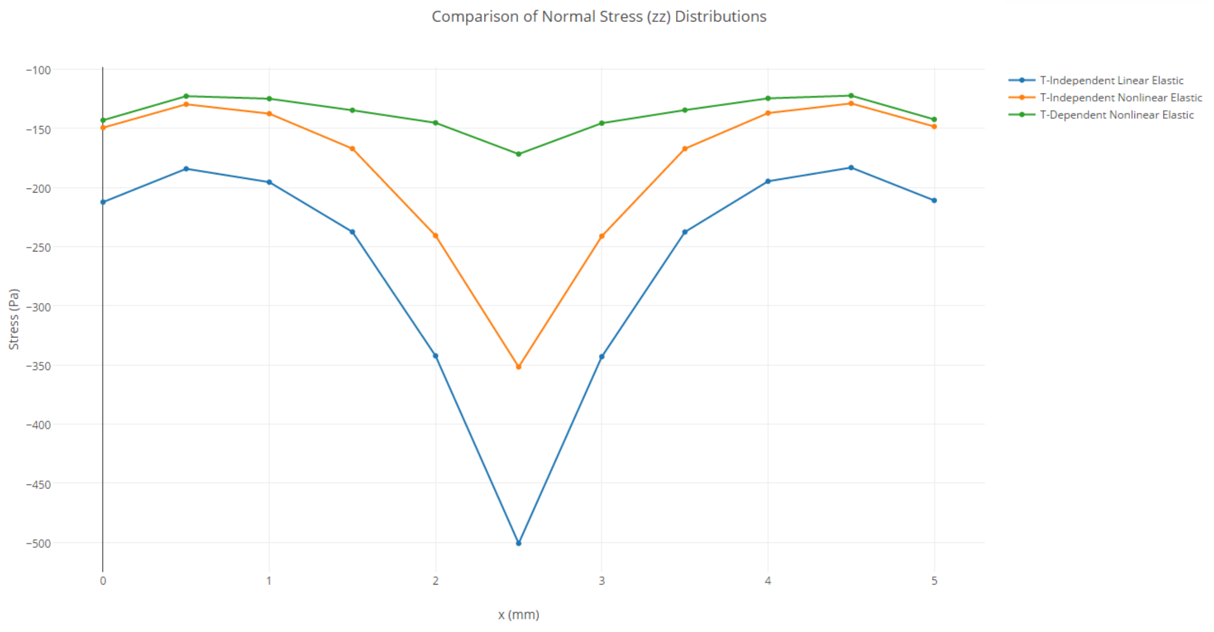


Figure 4-47: Comparison of Normal Stress (zz) Distributions

While it is highly recommended due to the results, that hyperthermia simulations account for temperature-dependent stress-strain, note that it should not be necessary to account for this in simple mechanical simulations. Procedures involving only cutting, suturing and excision actions would not benefit from the computational overhead required to account for thermally-dependent stress-strain, even for temperature distribution computation. In such cases however, it is still recommended that temperature-independent nonlinear-elastic stress-strain be preferred over temperature-independent linear-elastic stress-strain, as the nonlinear behaviour gives a closer approximation of realistic mechanical response of the tissue. For approximate mechanical simulations where iteration speed is critical over realism, temperature-independent linear-elastic stress-strain could be applied, using a simple Young's Modulus value. Above all though, for hyperthermia treatment and thermomechanical simulations, temperature-dependent nonlinear-elastic stress-strain should be preferred for realism.

4.5 Validation

4.5.1 Temperature Distribution Validation

Picinbono, Delingette and Ayache (2003) discuss outright, the difficulty of validating an in-vivo simulation. The Pennes bioheat equation approximates heat transfer due to blood perfusion, where the global weight of the liver in-vivo, is 50% comprised of blood. Ex-vivo tissues obviously do not behave in the same manner due to the absence of blood perfusion. For real-time surgery simulation, researchers often rely on feedback from surgeons and other professionals in their field, who can compare the simulation behaviour to the behaviour encountered during a real-life procedure. A clear and definitive means of validation of simulators towards real operational application is yet to be defined (Carter, TJ et al. 2005). Zhong et al. (2012) discuss that in-vivo validations of simulations raise ethical issues, and typically, empirical evidences of mechanical (and hence thermal) properties of living tissues are used for verifying simulation behaviour. All tissue properties in this research project are carefully sourced from literature, with parameters as close as possible to in-vivo human liver parameters, except for cases of stress-strain comparison, where mechanical parameters are only sufficiently supplied in literature for hog liver or that of other similar species.

Temperature profiles can be compared to the single point-source distribution shown by Karaa, Zhang and Yang (2005) in Figure 2-7. See Figure 4-48 below for a comparison of simulated temperature distribution for a point source temperature of 54 °C (typical to RFA in literature) to that of a 60 °C point source in Karaa, Zhang and Yang (2005). For a point source, the temperature falloff is typically very rapid, such that any potential thermal damage zone is minimal. Due to the rapid cooling effect of blood perfusion in the tissue (accommodated in the Pennes bioheat equation), the heated zone spans up to only approximately 1 millimetre. Note that the temperature distribution for Figure 4-16 compares reasonably well with that of literature for a point source as mentioned in Figure 2-7; as demonstrated in Figure 4-48. Variation of the tissue volume which is significantly affected by temperature (between shown temperature distributions) is significantly impacted by the source temperature and the thermal parameters employed.

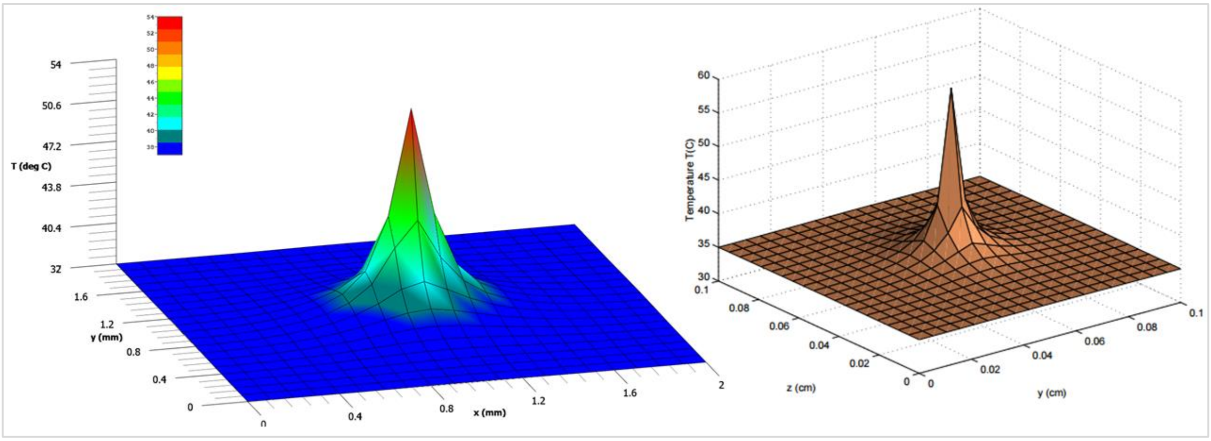


Figure 4-48: Point Source Tissue Temperature Distribution Comparison

Furthermore, commercial finite element analysis software (i.e. LISA) provides the following temperature distribution upon the mid-plane of a block of tissue heated by a 54 °C single point source at its centre; as shown in Figure 4-49.

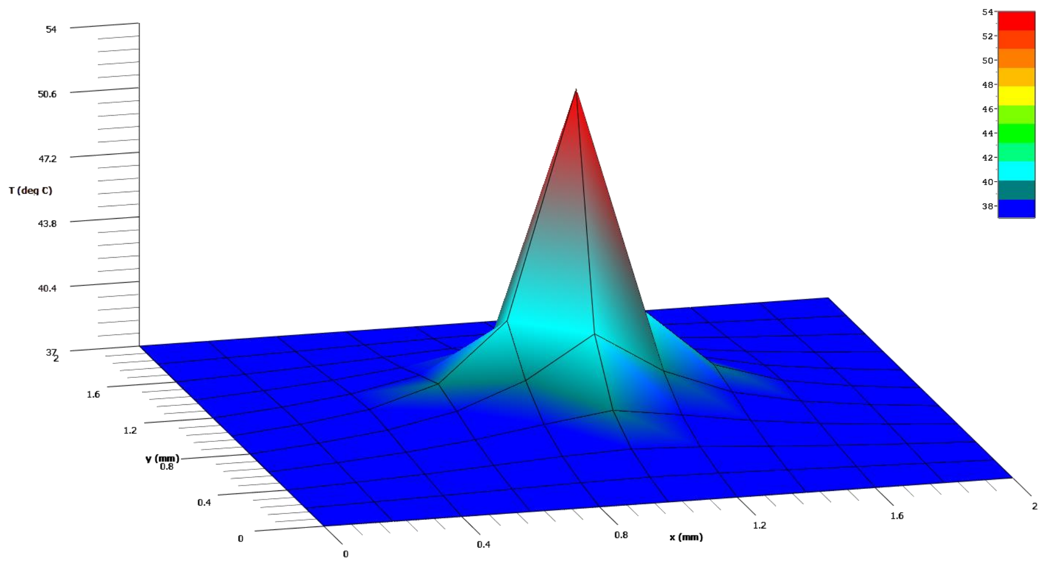


Figure 4-49: Point Source Tissue Temperature Distribution (LISA)

Note that while the mesh resolution is coarser (due to the maximum node limit available in LISA), the temperature profile is similar to that generated by the simulation developed for bioheat transfer. In the case in Figure 4-49, only conduction is considered throughout the tissue mesh block, and boundary conditions are Dirichlet (37 °C). Insulated (Neumann) boundary conditions are not appropriate for this implementation, since without accounting for the cooling effect of blood perfusion, the temperature distribution within the tissue reaches a steady-state where the tissue in its entirety becomes heated to the source temperature (54 °C). Thus Dirichlet boundary conditions at body temperature, help to produce an appropriate temperature distribution.

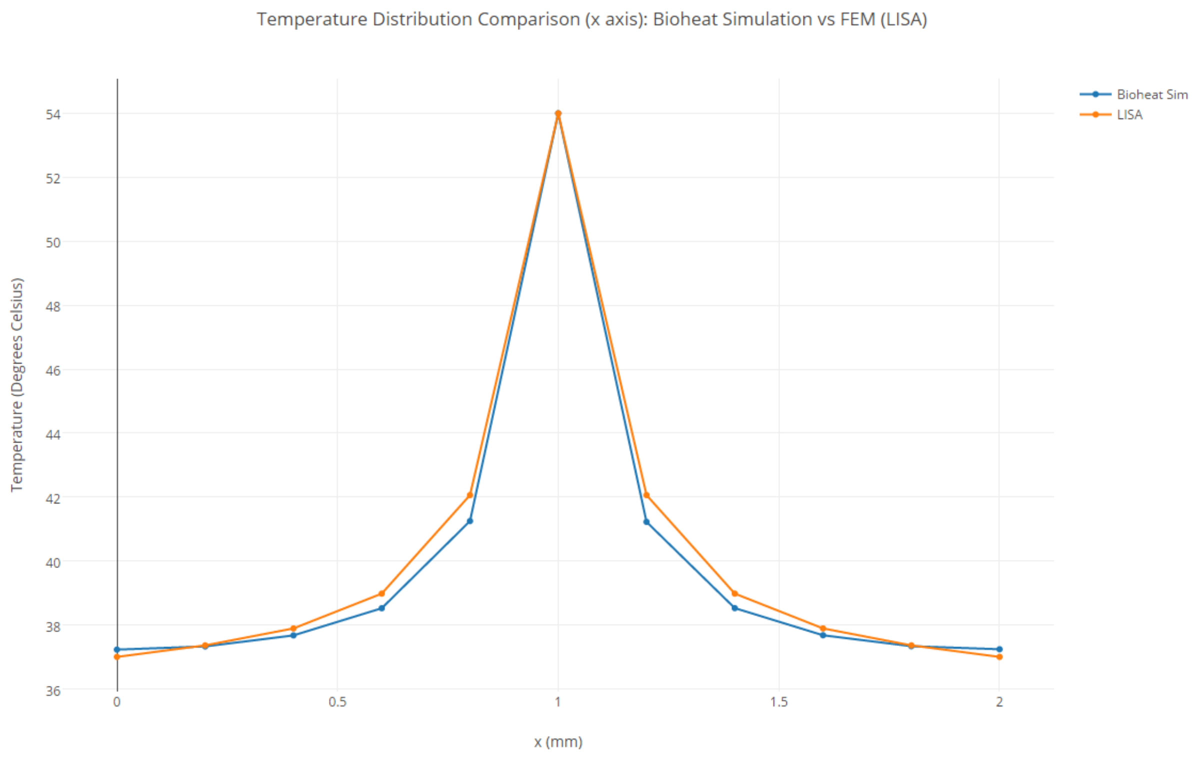


Figure 4-50: Simulation vs FEA Software (LISA) Temperature Distribution (x axis)

Also note Figure 4-50, where the x axis (mid plane) temperature distribution is compared between the simulation and commercial finite element analysis software (LISA). This comparison assists in validating the generated temperature distribution resulting from the developed simulation.

Since tissue thermal and mechanical parameters from literature are selected with consideration and comparison to multiple sources; and hence valid, then the thermal stress and strain distributions derived from use of these parameters and an appropriate valid temperature distribution can also be considered as valid. Consider further, the validation of the modelled temperature-dependent stress-strain functions in section 3.3.3, as compared to the original experimental data in literature shown in Figure 2-14. Section 4.5.2 addresses this comparison.

4.5.2 Temperature-Dependent Stress-Strain Relationship Validation

Figure 4-51, Figure 4-52 and Figure 4-53show the plotted stress-strain functions for low strains (0.0 to 0.4) based on the coefficients in Table 3.13 in section 3.3.3, against the experimental data from Xu and Lu (2009) as shown in Figure 2-14, from which these functions were derived. By applying these functions, and linear interpolation, as described in section 3.3.3, the temperature-dependent stress-strain-based thermomechanical (stress) response was modelled; the outcome of which is discussed and shown graphically in section 4.4.5.

Notice that each of the functions derived from the experimental data, fit precisely with the experimental data from which they were derived up until strains of 0.4, from which stage onwards, it is recommended that polynomial functions be modelled similarly for obtaining stress response, based on tabulated coefficient data in Table 3.12. Such strains would not be expected for purely thermally induced loading, however mechanical operational loading (section 2.5.3) would likely induce strains sufficient to warrant such functions.

The determination of tissue stress based on these functions and linear interpolation between said functions, is validated against this experimentally acquired temperature-dependent stress-strain data from literature.

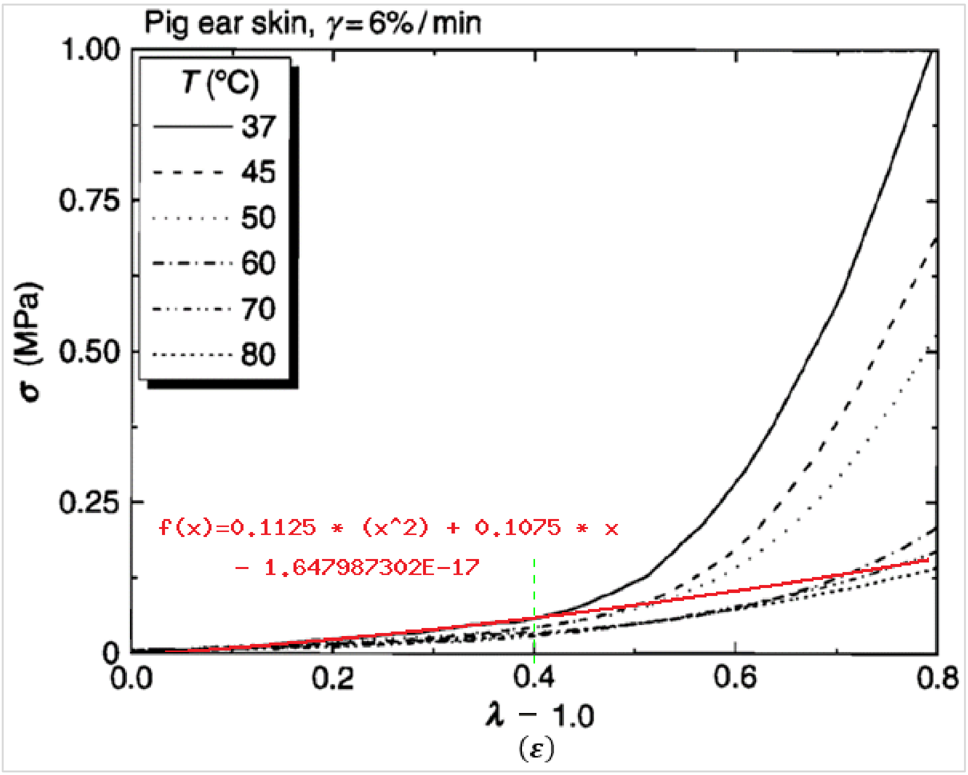


Figure 4-51: Function for Modelling Small Strains from 0 to 0.4 at 37 °C

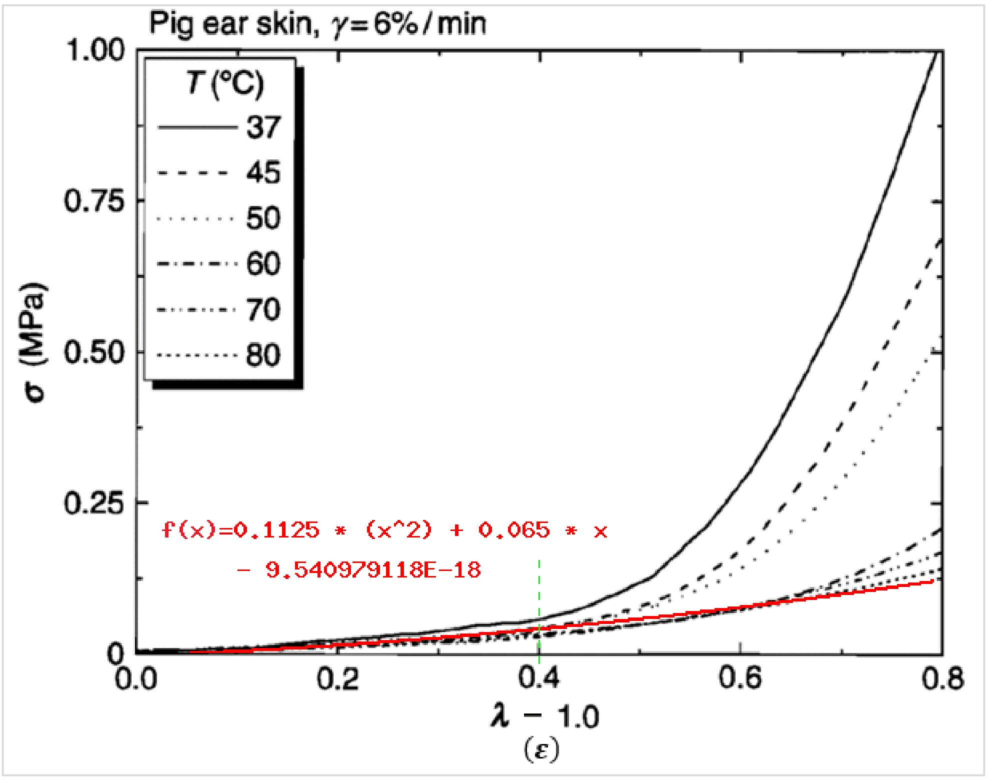


Figure 4-52: Function for Modelling Small Strains from 0 to 0.4 at 45 and 50 °C

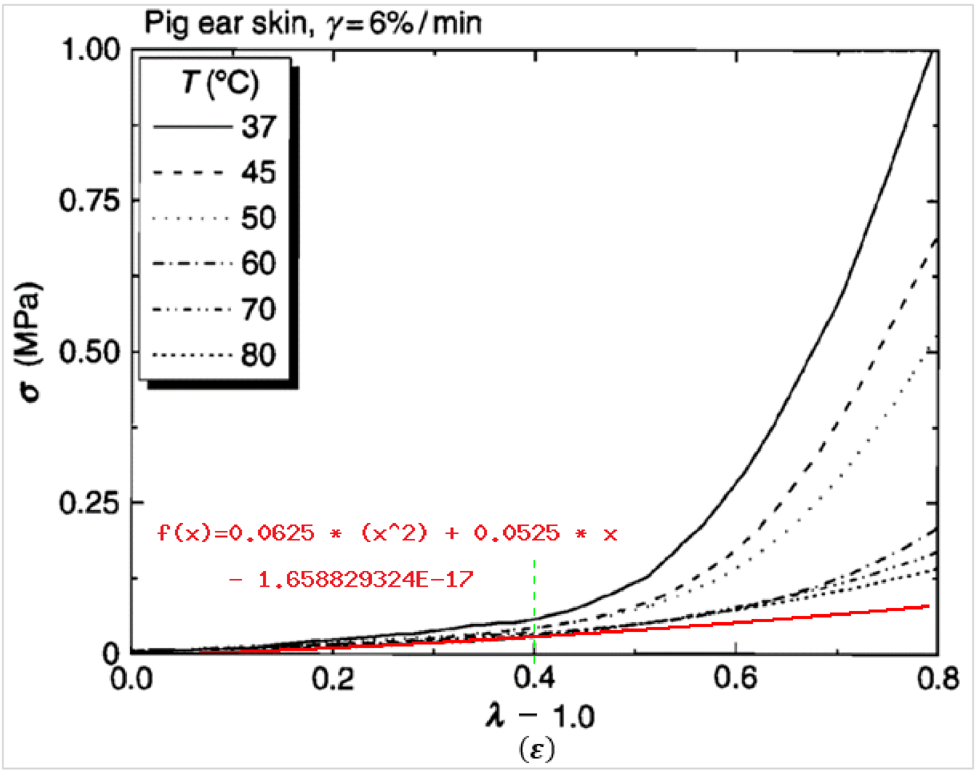


Figure 4-53: Function for Modelling Small Strains from 0 to 0.4 at 60, 70 and 80 °C

5 Conclusions

A combination of C++, HLSL, and the Direct3D and WIN32 APIs was used to program a GPU accelerated heat transfer and thermal damage simulation of liver tissue based on the Pennes bioheat equation and the Arrhenius Burn Integration, discretized spatially by Galerkin FEM and temporally by explicit FDM upon a 3D domain of linear tetrahedral elements, in visual real time (25 Hz). Thermomechanical response was modelled upon a combination of simulation temperature distribution, commercial FEM software (LISA) and temperature-dependent stress-strain functions modelled on data from literature (Xu & Lu 2009).

Analysis of simulation outcomes provided the following findings and derived conclusions:

- Temperature-dependent and temperature-independent thermal parameter bioheat transfer scenarios were simulated and compared against one another and against literature. Temperature-dependent tissue thermal conductivity and tissue specific heat capacity produce a greater temperature in the tissue temperature distribution than constant thermal parameters; contrary to literature (Watanabe et al. 2009). The literature in question can be seen to be poorly formulated in this regard, and implies comparison of two incomparable scenarios to obtain their conclusions.
- The subsequent simulated thermal damage distribution, modelled on the Arrhenius Burn Integration using the tissue temperature distribution, experienced a miniscule sub-millimetre falloff; similar to that of temperature. The small thermal damage distribution from the heat point source naturally indicated the purpose of the umbrella-like shape of the RFA needle, to increase coagulation volume during operation.
- GPU acceleration was applied to the bioheat transfer and thermal damage aspects of the simulation to maximize mesh resolution for real-time simulation. Visual real-time (25 Hz) performance of the implemented NVIDIA GTX 770M GPU enabled a 22,000 node (120,000 tetra) mesh to be computed upon for heat transfer and thermal damage, whereas the maximum single-core implemented CPU equivalent was limited to a 1,000 node mesh. For temperature-dependent thermal properties, the GPU supported suitable convergence for a mesh of 3,000 nodes. In general the GPU outperformed the CPU by greater than 50 times, concurring with Comas et al. (2008).
- Tissue normal thermal stresses were computed upon the liver tissue for temperature-independent linearly and nonlinearly elastic stress-strain relations, as well as for temperature-dependent nonlinearly elastic stress-strain relations. Compressive thermal strains and stresses were greater closer to the point heat source, producing up to 0.5 kPa internal stress. Temperature-dependent stress-strain relationships modelled based on Xu and Lu (2009) indicated greater than 200% difference in thermal stress as compared to temperature-independent stress-strain (and linear elasticity). This implies that if mechanical/deformation-based strains and thermal strains can be superposed as indicated in literature (Cook 1995), then after accounting for realistic mechanical/deformation stresses in operational settings (470 kPa; (Rosen et al. 2008)), the difference in stress state of the tissue could vary by a vast and highly significant magnitude compared to temperature-independent stress-strain. Thus, it is highly recommended that further heat-based procedures account for the temperature-dependence of stress-strain, whether purely mechanical/deformation-based response is considered, or also involving thermomechanical response. The inclusion of this dependence is vital to prevention of erring of accuracy by vast amounts (over 200%).
- Temperature distribution from a point source in the tissue showed a rapid falloff, validated as concurring with literature (Karaa, Zhang & Yang 2005). Temperature-dependent stress-strain functions modelled on experimental data from Xu and Lu (2009) matched very well at low

strains (0.0 to 0.4), validating the functions implemented to simulate and produce thermal stresses.

Sufficient work has still not been accomplished in literature, regarding reasonable experimentally obtained temperature-dependent stress-strain data of in-vivo tissues, and that of other parameters (mechanical and thermal); particularly of human organs. Ethics approvals and reasonable clearances for such experimentation as to obtain these relationships are inherently difficult to obtain, thus the available data is limited. Were such parameters and relationships able to be obtained for in-vivo human liver tissue, the simulation of RFA could be more accurately positioned. Furthermore, real-time simulation of thermomechanical tissue response has not yet been implemented, and it is unknown to what degree of resolution a mesh can be supported for real-time visual feedback on modern hardware; even if GPU accelerated.

6 Future Work

Several specific improvements and extensions are recommended for building upon the work demonstrated in this research project.

Firstly, the real-time component of the simulation should be expanded to encompass the calculation of thermal stresses and strains at the node locations upon the domain. It is recommended that the coefficient of (linear) thermal expansion be used in conjunction with the temperature change to determine the thermal strain component that each element exerts on its nodes. Revisiting section 2.4; Cook (1995); it must be noted that for each element, to restrain all nodal DoF (displacement) of each node, and compute loads applied by the element upon those nodes due to change in temperature. From these thermal strains, thermal stresses could be calculated within the simulation by implementing functions based on published temperature-dependent stress-strain data of collagenous tissue, as done in this project (see section 3.3.3). Similar models if applicable, could be used to determine the thermal stresses within the tissue. The thermal stress must be a function of both the strain and the tissue temperature. Furthermore, purely mechanical loads can be superposed upon thermal loads to produce a total loading condition. If the thermal component of strain, and the mechanical/displacement-induced component of strain can be superposed (as suggested by Cook (1995)), then the temperature-dependent stress calculation could be applied.

Collagenous tissue exhibits viscoelastic behaviour, as discussed in section 2.5.2. Thus time-dependent response should be accounted for, such as the relaxation of the tissue. For example, non-Newtonian fluids/putties exhibit viscoelastic behaviour, providing resistance to loading initially, but tend heavily towards relaxation over time, and thus a reduction in load response. Thus, a time-based response of soft collagenous tissue may result in an initial peak of mechanical stress response, however decline gradually, tending to some lesser amount as the collagen fibres align and slip since the application of loading. Hysteresis is the time-dependent response of the tissue to loading based not only on the current loading, but a loading history of the tissue. Brown, Jeffrey D et al. (2003) discuss that the point at which repeated loading results in a stabilized response pattern is called conditioning. Tissues used experimentally are pre-conditioned in order to produce consistent results, however in an operational environment, tissues are not preconditioned, and thus a realistic simulation should account for this. These time-dependent responses of the tissue would best be suited to be implemented after mechanical loading superposition is implemented.

Temperature-dependent blood perfusion is strongly advised by Drizdal et al. (2010), as the temperature profile in skin, fat and muscle tissue does not very well match between constant (temperature-independent) and temperature-dependent blood perfusion scenarios. Temperature-dependent scaling functions are developed each for skin, fat and muscle tissues, with which to scale the perfusion rate in the tissues. This is applied to the blood perfusion rate of the Pennes bioheat transfer model for determining the temperature distribution. It is recommended that as far as possible, the parameters of the Pennes bioheat equation be modelled as being temperature-dependent to improve the accuracy and realism of the temperature distribution in the soft tissue. This implementation would result in an iterative process at each time step, requiring calculation of the tissue temperature field, calculation of the temperature-dependent thermal parameters, recalculation of the tissue temperature field, and so on, repeating until suitable convergence criteria are met for the thermal parameters. It has already been demonstrated that temperature-dependence of thermal parameters causes the resultant temperature distribution to differ significantly (see section 4.1.1).

Tumour tissue properties and tissue heterogeneity have not been accounted for in this research project. The liver tissue has been treated as an isotropic, homogeneous domain, whilst the tumour tissue its self has not been separately defined or discerned from the healthy live tissue. This has been due to the additional complexity required in order to represent separate tissues with varying properties. In order to improve realism, the simulation could be extended to support the application of different materials and properties to different parts of the domain. Song et al. (1984) describe the blood flow in tumours as reducing as temperature is raised, due to newly-formed tumour blood vessels which are easily damaged, thus reducing blood perfusion. This means that while heat dissipates quickly from healthy tissue, tumour tissue may retain heat much better than healthy tissue, to a significant degree. Accounting for this behaviour of heat transport may improve the realism of the simulation. Future simulated implementations should consider treating specific regions and mesh elements as tumour tissue, and other regions as healthy tissue. To achieve this, element (tetra) properties could be added to define tissue type, or to store parametric information such as thermal conductivity, specific heat capacity, or blood perfusion rate. Upon solution computation, each elements parameters would determine the heat transfer between nodes forming the element its self. The temperature field would then indicate the boundary between healthy and tumour tissues, due to increasingly distinct temperature distribution and falloff behaviours.

Water content of soft tissue is the final recommendation for consideration of the extension of this project. Deshan et al. (2007) discuss the behaviour of water within the tissue during heating. Upon heating, water vaporises, diffuses through tissue to regions of lower pressure (and lower heat), and condenses in these locations, depositing latent heat energy as it cools, thus heating these cooler regions. The heated tissue, upon reduction of water content, could be further supposed to shrink (as well as due to the denaturation of collagen (Xu, Lu & Seffen 2008; Xu, Seffen & Lu 2008a)), perhaps even negating the effect of thermal expansion in the tissue due to the heating (as investigated in this research project).

Thus in summary, for temperature distribution, temperature-dependent blood perfusion (and other parameters of the bioheat equation), tumour tissue thermal properties and tissue water content should be accounted for to improve the realism of the solution of temperature distribution. Upon this distribution, thermal strains can be calculated due to thermal expansion as well as a component due to tissue shrinkage (from collagen denaturation). These thermally induced strain components can be superposed, and thus the thermal strain can be superposed with mechanical deformation-based strain, which is based on viscoelastic response behaviour. From this, the total resultant stress state upon the tissue can be calculated using a temperature-dependent stress-strain model (such as the one implemented in this research project). Following these recommendations, a more realistic and accurate albeit computationally expensive simulation could be developed. As a further recommendation, the computational implementation should be investigated further, using known methods to optimize shader code and execution on the GPU, as well as minimizing data transfer between GPU and CPU, thus maximizing the mesh resolution able to be computed upon in visual real time. Finally, haptic interfacing may be implemented, however a different implementation is advised in such a case, such that the very high update frequency required by haptic interfacing can be achieved.

7 References

Bittar, E 2004, *The Liver in Biology and Disease: Liver Biology in Disease, Hepato - Biology in Disease*, vol. 15, Principles of Medical Biology, Elsevier.

Boresi, AP, Chong, K & Lee, JD 2010, *Elasticity in Engineering Mechanics*, Wiley.

Borsic, A, Hoffer, E & Attardo, EA 2014, 'GPU-Accelerated real time simulation of Radio Frequency Ablation thermal dose', paper presented to Bioengineering Conference (NEBEC), 2014 40th Annual Northeast, 25-27 April 2014.

British Medical Association 1990, *Complete Family Health Encyclopedia*, Dorling Kindersly, Ltd, London.

Brown, JD, Rosen J Fau - Chang, L, Chang L Fau - Sinanan, MN, Sinanan Mn Fau - Hannaford, B & Hannaford, B 2004, 'Quantifying surgeon grasping mechanics in laparoscopy using the Blue DRAGON system', no. 0926-9630 (Print).

Brown, JD, Rosen, J, Sinanan, MN & Hannaford, B 2003, *In-Vivo and Postmortem Compressive Properties of Porcine Abdominal Organs*, Springer Berlin Heidelberg, 978-3-540-39899-8, <http://link.springer.com/chapter/10.1007%2F978-3-540-39899-8_30>.

Carter, FJ, Frank, TG, Davies, PJ, McLean, D & Cuschieri, A 2001, 'Measurements and modelling of the compliance of human and porcine organs', *Medical Image Analysis*, vol. 5, no. 4, pp. 231-6.

Carter, TJ, Sermesant, M, Cash, DM, Barratt, DC, Tanner, C & Hawkes, DJ 2005, 'Application of soft tissue modelling to image-guided surgery', *Med Eng Phys*, vol. 27, no. 10, pp. 893-909.

Comas, O, Taylor, Z, Allard, J, Ourselin, S, Cotin, S & Passenger, J 2008, 'Efficient Nonlinear FEM for Soft Tissue Modelling and Its GPU Implementation within the Open Source Framework SOFA', in F Bello & PJE Edwards (eds), *Biomedical Simulation*, Springer Berlin Heidelberg, vol. 5104, pp. 28-39.

Cook, RD 1995, *Finite Element Modeling for Stress Analysis*, John Wiley & Sons, Inc., Toronto, Canada.

Courtecuisse, H, Jung, H, Allard, J, Duriez, C, Lee, DY & Cotin, S 2010, 'GPU-based real-time soft tissue deformation with cutting and haptic feedback', *Prog Biophys Mol Biol*, vol. 103, no. 2-3, pp. 159-68.

Delingette, H & Ayache, N 2004, 'Soft tissue modeling for surgery simulation', *Computational models for the human body*, pp. 453-550.

Deshan, Y, Converse, MC, Mahvi, DM & Webster, JG 2007, 'Expanding the Bioheat Equation to Include Tissue Internal Water Evaporation During Heating', *Biomedical Engineering, IEEE Transactions on*, vol. 54, no. 8, pp. 1382-8.

Dillenseger, JL & Esneault, S 2010, 'Fast FFT-based bioheat transfer equation computation', *Comput Biol Med*, vol. 40, no. 2, pp. 119-23.

Drizdal, T, Togni, P, Visek, L & Vrba, J 2010, 'Comparison of constant and temperature dependent blood perfusion in temperature prediction for superficial hyperthermia', *Radioengineering*, vol. 19, no. 2, pp. 281-9.

Eipel, C 2010, 'Regulation of hepatic blood flow: The hepatic arterial buffer response revisited', *World Journal of Gastroenterology*, vol. 16, no. 48, p. 6046.

Eisenberg, MA & Malvern, LE 1973, 'On finite element integration in natural co-ordinates', *International Journal for Numerical Methods in Engineering*, vol. 7, no. 4, pp. 574-5.

Frank, AO, Twombly, IA, Barth, TJ & Smith, JD 2001, 'Finite element methods for real-time haptic feedback of soft-tissue models in virtual reality simulators', paper presented to Virtual Reality, 2001. Proceedings. IEEE, 17-17 March 2001.

Gupta, PK, Singh, J & Rai, KN 2010, 'Numerical simulation for heat transfer in tissues during thermal therapy', *Journal of Thermal Biology*, vol. 35, no. 6, pp. 295-301.

Hasgall PA, Di Gennaro F, Baumgartner C, Neufeld E, Gosselin MC, Payne D, Klingenböck A & N, K 2013, *IT'IS Database for thermal and electromagnetic parameters of biological tissues Version 2.5*, viewed 14th December 2013, <www.itis.ethz.ch/database>.

Holzapfel, GA 2000, *Biomechanics of Soft Tissue*, Graz University of Technology, Graz, Austria.

Karaa, S, Zhang, J & Yang, F 2005, 'A numerical study of a 3D bioheat transfer problem with different spatial heating', *Mathematics and Computers in Simulation*, vol. 68, no. 4, pp. 375-88.

Kattan, P 2008, *MATLAB Guide To Finite Elements*, 2 edn, Springer, Heidelberg.

Keangin, P, Wessapan, T & Rattanadecho, P 2011, 'Analysis of heat transfer in deformed liver cancer modeling treated using a microwave coaxial antenna', *Applied Thermal Engineering*, vol. 31, no. 16, pp. 3243-54.

Kolchanov, NA 2006, *Liver*, GeneNetWorks - Institute of Cytology and Genetics, viewed 1 January 2014, <http://wwwmgs.bionet.nsc.ru/mgs/gnw/trrd/thesaurus/Di/liver.html>.

Kup-Sze, C, Hanqiu, S & Pheng-Ann, H 2003, 'Interactive deformation of soft tissues with haptic feedback for medical learning', *Information Technology in Biomedicine, IEEE Transactions on*, vol. 7, no. 4, pp. 358-63.

Liszka, T & Orkisz, J 1980, 'The finite difference method at arbitrary irregular grids and its application in applied mechanics', *Computers & Structures*, vol. 11, no. 1–2, pp. 83-95.

Liu, Z & Bilston, LE 2002, 'Large deformation shear properties of liver tissue', *Biorheology*, vol. 39, no. 6, pp. 735-42.

Luna, FD 2012, *Introduction to 3D Game Programming With DirectX 11*, Mercury Learning and Information LLC, Dulles, VA.

Manescu, P, Azencot, J, Beuve, M, Ladjal, H, Saadé, J, Morreau, Jean-Michel, Giraud, P & Shariat, B 2012, 'Material density mapping on deformable 3D models of human organs', *World Academy of Science, Engineering & Technology*, vol. June 2012, no. 66, pp. 131-40.

McGahan, J & Raalte, V 2005, 'History of Ablation', in E vanSonnenberg, W McMullen & L Solbiati (eds), *Tumor Ablation*, Springer New York, pp. 3-16.

Miller, K 2000, 'Constitutive modelling of abdominal organs', *Journal of Biomechanics*, vol. 33, no. 3, pp. 367-73.

Misra, S, Ramesh, KT & Okamura, AM 2010, 'Modelling of non-linear elastic tissues for surgical simulation', *Comput Methods Biomech Biomed Engin*, vol. 13, no. 6, pp. 811-8.

Nakayama, A & Kuwahara, F 2008, 'A general bioheat transfer model based on the theory of porous media', *International Journal of Heat and Mass Transfer*, vol. 51, no. 11-12, pp. 3190-9.

Nefti, S & Abulgasem, E 2009, 'Neural networks approach for soft tissue modelling', paper presented to Emerging Technologies, 2009. ICET 2009. International Conference on, 19-20 Oct. 2009.

Nikishkov, GP 2010, 'Finite Element Equations for Heat Transfer', in *Programming Finite Elements in Java™*, Springer London, pp. 13-9.

Okazaki, I, Ninomiya, Y, Kyuichi, T & Friedman, SI 2003, *Extracellular Matrix and The Liver: Approach to Gene Therapy*, Academic Press.

Özisik, MN 1994, *Finite Difference Methods in Heat Transfer*, CRC Press (Taylor & Francis Group), Boca Raton, Florida.

Peng, T, O’Neill, DP & Payne, SJ 2011, 'A two-equation coupled system for determination of liver tissue temperature during thermal ablation', *International Journal of Heat and Mass Transfer*, vol. 54, no. 9-10, pp. 2100-9.

Picinbono, G, Delingette, H & Ayache, N 2003, 'Non-linear anisotropic elasticity for real-time surgery simulation', *Graphical Models*, vol. 65, no. 5, pp. 305-21.

Prakash, P 2010, 'Theoretical Modeling for Hepatic Microwave Ablation', *The Open Biomedical Engineering Journal*, vol. 4, no. 5, pp. 27-38.

Prakash, P & Diederich, CJ 2012, 'Considerations for theoretical modelling of thermal ablation with catheter-based ultrasonic sources: Implications for treatment planning, monitoring and control', *International Journal of Hyperthermia*, vol. 28, no. 1, pp. 69-86.

Roland W. Lewis, Perumal Nithiarasu & Seetharamu, KN 2004, *Fundamentals of the Finite Element Method for Heat and Fluid Flow*, John Wiley & Sons Inc., West Sussex, England.

Rosen, J, Brown, JD, De, S, Sinanan, M & Hannaford, B 2008, 'Biomechanical properties of abdominal organs in vivo and postmortem under compression loads', *J Biomech Eng*, vol. 130, no. 2, p. 021020.

Saad, Y 2003, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, viewed February 15, 2015, <http://www-users.cs.umn.edu/~saad/IterMethBook_2ndEd.pdf>.

Satava, RM 1996, 'Medical virtual reality. The current status of the future', *Stud Health Technol Inform*, vol. 29, pp. 100-6.

Sewell, R 2010, *The RITA Treatment for Liver Tumors*, Dr. Robert Sewell, YouTube, <http://www.youtube.com/watch?v=UsSVAHE-CfM>.

Shen, W, Zhang, J & Yang, F 2005a, 'Modeling and numerical simulation of bioheat transfer and biomechanics in soft tissue', *Mathematical and Computer Modelling*, vol. 41, no. 11–12, pp. 1251-65.

---- 2005b, *Skin Thermal Injury Prediction with Strain Energy*, 427-05, University of Kentucky, Lexington, Kentucky.

---- 2005c, 'Three-dimensional model on thermal response of skin subject to laser heating', *Comput Methods Biomech Biomed Engin*, vol. 8, no. 2, pp. 115-25.

Song, CW, Lokshina, A, Rhee, JG, Patten, M & Levitt, SH 1984, 'Implication of Blood Flow in Hyperthermic Treatment of Tumors', *Biomedical Engineering, IEEE Transactions on*, vol. BME-31, no. 1, pp. 9-16.

Stout, R & Billings, D 2002, 'Accuracy and Time Resolution in Thermal Transient Finite Element Analysis', paper presented to 2002-Int-ANSYS-Conf-91, <http://ansys.com/staticassets/ANSYS/>.

Terzopoulos, D, Platt, J & Fleischer, K 1991, 'Heating and melting deformable models', *The Journal of Visualization and Computer Animation*, vol. 2, no. 2, pp. 68-73.

University of Rochester Medical Center 2014, *The Liver: Anatomy and Functions*, URMC, viewed 1st January 2014, <http://www.urmc.rochester.edu/Encyclopedia/Content.aspx?ContentTypeID=85&ContentID=P00676>.

Venkatasubramanian, P 2012, 'Imaging the pancreatic ECM', in P Grippo & H Munshi (eds), *Pancreatic Cancer and Tumor Microenvironment*, Transworld Research Network, Trivandrum (India).

Watanabe, H, Kobayashi, Y, Hashizume, M & Fujie, MG 2009, 'Modeling the temperature dependence of thermophysical properties: Study on the effect of temperature dependence for RFA', paper presented to Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE, 3-6 Sept. 2009.

Xu, F & Lu, TJ 2009, 'Chapter 3 Skin Biothermomechanics: Modeling and Experimental Characterization', in A Hassan & G Erik van der (eds), *Advances in Applied Mechanics*, Elsevier, vol. Volume 43, pp. 147-248.

Xu, F, Lu, TJ & Seffen, KA 2008, 'Biothermomechanics of skin tissues', *Journal of the Mechanics and Physics of Solids*, vol. 56, no. 5, pp. 1852-84.

Xu, F, Seffen, KA & Lu, TJ 2008a, 'Non-Fourier analysis of skin biothermomechanics', *International Journal of Heat and Mass Transfer*, vol. 51, no. 9–10, pp. 2237-59.

---- 2008b, 'Temperature-Dependent Mechanical Behaviors of Skin Tissue', *IAENG International Journal of Computer Science*, vol. 35, no. 1, p. 92+.

Xu, J, Jia, Z-z, Song, Z-j, Yang, X-d, Chen, K & Liang, P 2010, 'Three-dimensional ultrasound image-guided robotic system for accurate microwave coagulation of malignant liver tumours', *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 6, no. 3, pp. 256-68.

Zhong, Y, Shirinzadeh, B, Smith, J & Gu, C 2012, 'Soft tissue deformation with reaction-diffusion process for surgery simulation', *Journal of Visual Languages & Computing*, vol. 23, no. 1, pp. 1-12.

Zink, J, Pettineo, M & Hoxley, J 2011, *Practical Rendering and Computation with Direct3D 11*, Taylor & Francis.

Zolfaghari, A & Maerefat, M 2011, *Bioheat Transfer, Developments in Heat Transfer*, InTech, <http://www.intechopen.com/books/developments-in-heat-transfer/bioheat-transfer>.