Université Lille1
Sciences et Technologies

Thesis submitted to obtain the title of
Doctor of Philosophy

Doctoral School of Engineering Science
Field: Computer Science

Real-time Soft Tissue Modelling on GPU for Medical Simulation

Prepared by Olivier Comas at
INRIA Lille, Shaman Team and CSIRO ICT Brisbane, AEHRC

DRAFT - Thu 30th of September 2010 at 21:26

Jury:

Advisor:	Stéphane Cotin	-	INRIA (Shaman team), Lille, France
Reviewers:	Thomas Sangild Sørensen	-	University of Aarhus, Denmark
	Georges Dumont	-	INRIA (Bunraku team), Rennes, France
President:	Laurent Grisoni	-	INRIA (Mint team), Lille, France
Examinators:	Tamy Boubekeur	-	Telecom ParisTech, Paris, France
	Bernard	-	INRIA Lille, au bout du couloir

Contents

Part I

Introduction

Chapter 1

Medical simulation

A short abstract for the upcoming chapter

1.1 General context

1.1.1 Simulators for everything: we started by simulating what is simple

1.1.2 As complexity increases, applications to medical training, patient-specific planning and per-operative guidance

1.2 Challenges

1.2.1 Photorealistic rendering

1.2.2 Accurate haptic feedback

1.2.3 Physically realist organ modelling in real-time

Chapter 2

Background in continuum mechanics for soft-tissue modelling

As seen in the previous chapter, realistic modelling of organ deformation is a challenging research field that opens the door to new clinical applications including: medical training and rehearsal systems, patient-specific planning of surgical procedures and per-operative guidance based on simulation. In all these cases the clinician needs fast updates of the deformation model to obtain a real-time display of the computed deformations. If for medical training devices the haptic feedback from touching organs merely needs to feel real, the accuracy of the information provided to the clinician in the cases of planning or per-operative guidance is crucial. Therefore, a substantial comprehension of the mechanics involved and a knowledge of the physical properties of anatomical structures are both mandatory in our quest to realistically model the deformation of organs. This chapter will start by introducing the main concepts of continuum mechanics that are fundamental to study the mechanical response of organs. It will then present mathematical models able to describe the different mechanical aspects of materials and briefly discuss the mechanical characterisation of tissues.

2.1 Introduction

In our everyday life, matter appears smooth and continuous: from the wood used to build your desk to the water you drink. But this is just illusion. The concept that matter is composed of discrete units has been around for millennia. In fact, we know with certainty that our world is composed of microscopic atoms and molecules separated by empty space since the beginning of the twentieth century (Lautrup, 2005). However, certain physical phenomena can be predicted with theories that pay no attention to the molecular structure of materials. Consider for instance the deformation of the horizontal board of a bookshelf under the weight of books. The bending of the shelf can be modelled without considering its molecular composition. The branch of physics in which materials are treated as continuous is known as *continuum mechanics*. Continuum mechanics studies the response of materials to different loading conditions. In this theory, matter is assumed to exist as a continuum, meaning that the matter in the body is continuously distributed and fills the entire region of space it occupies (Lai et al., 1996). This assumption is generally valid if the length scales of interest are large compared with the length scales of discrete molecular structure, but whether the approximation of continuum mechanics is actually justified in a given situation is merely a matter of experimental test.

Modelling anatomical structures requires an understanding of the deformation and stresses caused by the different interactions that occur during medical procedures. A sufficient knowledge of continuum mechanics is therefore essential to follow the rest of this manuscript. Continuum mechanics can be divided into two main parts: general principles common to all media (analysis of deformation, strain and stress concepts) and constitutive equations defining idealised materials. This chapter will not only deal with those two aspects but will also discuss the different aspects of the mechanical behaviour that we find in anatomical structures. Note that this chapter will mostly follow the notation used by Bonnet and Wood (1997) and Reddy (2007). The interested reader may refer to these books for more details.

2.2 Description of motion

Let us consider a body $\boldsymbol{B}$ of known geometry in a three-dimensional Euclidian space $\mathbb{R}^3$. For a given geometry and loading, $\boldsymbol{B}$ will undergo a set of macroscopic changes which is called *deformation*. The region of space occupied by the body at a given time t is termed a *configuration*. A change in the configuration of a continuum body results in a displacement. The displacement of a body has two components: a rigid-body displacement and a deformation. A rigid-body displacement consists of a simultaneous translation and rotation of the body without changing its shape or size. In constrast, a change in shape and/or size of the body from an initial configuration κ_0 to a new configuration κ is called a deformation. This new configuration κ may be designated by the *current* or *deformed configuration*.

Let us now consider a given particle of the body that we call X. What we will call particle in the following is in fact an infinitesimal volume of material. We denote

the position it occupies in the initial configuration $\mathbf{X}$ and note its position in the deformed configuration $\mathbf{x}$, both expressed in the chosen frame of reference. The mapping χ defined as the following:

$$\chi : \boldsymbol{B}_{\kappa_0} \rightarrow \boldsymbol{B}_{\kappa} \tag{2.1}$$

is called the *deforming mapping* of the body $\boldsymbol{B}$ from κ_0 to κ. When analysing the deformation of a continuous body, it is necessary to describe the evolution of configurations through time. Its mathematical description follows one of the two approaches: the material description or the spatial description. The material description is known as *Lagrangian description* whereas the spatial description is called *Eulerian description*. These two approaches are detailed next.

2.2.1 Lagrangian description

In the Lagrangian description, the position and physical properties of the particles are referred to a reference configuration κ_R, often chosen to be the undeformed configuration κ_0. Consequently, the current coordinates ($\mathbf{x} \in \kappa$) depend on the reference coordinates ($\mathbf{X} \in \kappa_0$):

$$\mathbf{x} = \chi(\mathbf{X}, t), \quad \chi(\mathbf{X}, 0) = \mathbf{X}, \tag{2.2}$$

and a typical variable ϕ defined over the body is expressed in terms of the coordinates $\mathbf{X}$ and time t:

$$\phi = \phi(\mathbf{X}, t). \tag{2.3}$$

Let us consider a particule X whose position in the reference configuration is $\mathbf{X}$. From (2.3), we note that the value of ϕ associated with this fixed particle X changes over time. Hence, the Lagrangian description focuses its attention on the particles of the continuous body and it is usually used in solid mechanics.

2.2.2 Eulerian description

In constrast, the Eulerian description is interested in changes at fixed locations. This time, the motion and ϕ are described with respect to the current position ($\mathbf{x} \in \kappa$):

$$\phi = \phi(\mathbf{x}, t), \quad \mathbf{X} = \mathbf{X}(\mathbf{x}, t). \tag{2.4}$$

For a fixed value of $\mathbf{x} \in \kappa$, $\phi(\mathbf{x}, t)$ gives the value of ϕ associated with the particle occupying the position $\mathbf{x} \in \kappa$, which may very well be a different particle for each new time t. Because a change in time t implies that a different value ϕ is observed at the same spatial location $\mathbf{x} \in \kappa$, now probably occupied by a different particle, the Eulerian description is focused on a spatial position. This approach is convenient for the study of fluid flow where the kinematic property of greatest interest is the rate at which change is taking place rather than the shape of the body of fluid at a reference time (Spencer, 1980). Because we are only interested in the study of solid bodies, the Lagrangian description will be used in the rest of the text.

2.2.3 Displacement field

The displacement $\mathbf{u}$ of a particle X is called *displacement vector* and its expression in the Lagrangian description is given by the following:

$$\mathbf{u}(\mathbf{X}, t) = \mathbf{x}(\mathbf{X}, t) - \mathbf{X}. \tag{2.5}$$

A *displacement field* is a vector field of all displacement vectors for all particles in the body. Thus, the deformed configuration κ may be obtained from the undeformed configuration κ_0 by merely adding the displacement field: κ_0: $\chi(\mathbf{X}, t) = \mathbf{X} + \mathbf{u}(\mathbf{X})$.

2.3 Analysis of deformation

2.3.1 Deformation gradient tensor

The displacement field tells us how a particule displaces from the reference to the deformed configuration. More interestingly, we would like to know how a material line would deform (stretch and rotation) within this displacement field. Since the length of a material line $d\mathbf{X}$ can change when going to the deformed configuration as well as its orientation, we can say that $d\mathbf{X}$ deforms into $d\mathbf{x}$. We now seek the relation existing between $d\mathbf{x}$ in the deformed configuration and $d\mathbf{X}$ of the reference configuration.

Consider two particles P_1 and P_2 in a continuous body separated by an infinitesimal distance $d\mathbf{X}$:

$$d\mathbf{X} = \mathbf{X}_{P_2} - \mathbf{X}_{P_1}. \tag{2.6}$$

After deformation, the two particles have deformed to their current positions given by the mapping χ (2.1) as:

$$\mathbf{x}_{P_1} = \chi(\mathbf{X}_{P_1}, t) \quad \text{and} \quad \mathbf{x}_{P_2} = \chi(\mathbf{X}_{P_2}, t). \tag{2.7}$$

Using (2.6) the distance $d\mathbf{x}$ between P_1 and P_2 can then be expressed as:

$$d\mathbf{x} = \mathbf{x}_{P_2} - \mathbf{x}_{P_1} = \chi(\mathbf{X}_{P_1} + d\mathbf{X}, t) - \chi(\mathbf{X}_{P_1}, t). \tag{2.8}$$

The *deformation gradient* $\mathbf{F}$ can be defined as:

$$\mathbf{F} = \frac{\partial \chi}{\partial \mathbf{X}} \tag{2.9}$$

and the vector $d\mathbf{x}$ can then be obtained in terms of $d\mathbf{X}$ as:

$$d\mathbf{x} = \mathbf{F}\, d\mathbf{X}. \tag{2.10}$$

Note that $\mathbf{F}$ transforms vectors from the reference configuration into vectors in the current configuration and is therefore a second-order tensor.

Knowing that $\chi(\mathbf{X}, t)$ is of course $\mathbf{x}$, the deformation gradient may also be written as:

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \nabla_0 \mathbf{x} = \nabla_0 \mathbf{u} + \mathbf{I}, \tag{2.11}$$

where ∇_0 is the gradient operator with respect to $\mathbf{X}$ and $\mathbf{u}$ the displacement vector. In indicial notation in a Cartesian coordinate system, (2.11) can be explicited as:

$$[F] = \begin{bmatrix} \frac{\partial \mathbf{x}_1}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{x}_1}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{x}_1}{\partial \mathbf{X}_3} \\ \frac{\partial \mathbf{x}_2}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{x}_2}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{x}_2}{\partial \mathbf{X}_3} \\ \frac{\partial \mathbf{x}_3}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{x}_3}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{x}_3}{\partial \mathbf{X}_3} \end{bmatrix}. \tag{2.12}$$

2.3.2 Change of volume

At this point, we have seen how a deformation can affect a vector. We will now look into its effect on volumes. Our motivation comes from the need to write global equilibrium statements that involve integrals over volumes. Let us consider a volume in the reference configuration formed by three non-coplanar line elements $d\mathbf{X}^{(1)}$, $d\mathbf{X}^{(2)}$ and $d\mathbf{X}^{(3)}$ as can be seen on Fig. 2.1.

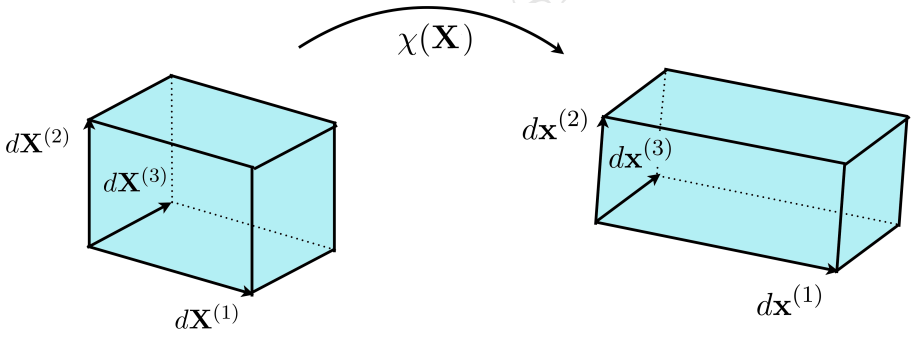


Figure 2.1: Transformation of a volume element under a deformation mapping.

The three vectors after deformation $d\mathbf{x}^{(1)}$, $d\mathbf{x}^{(2)}$ and $d\mathbf{x}^{(3)}$ can be obtained with:

$$d\mathbf{x}^{(i)} = \mathbf{F}\, d\mathbf{X}^{(i)}, \quad i = 1, 2, 3. \tag{2.13}$$

The volume of the parallelepiped that we will note dV can be calculated using the triple product between the three vectors:

$$\begin{aligned} dV &= d\mathbf{X}^{(1)} \cdot d\mathbf{X}^{(2)} \times d\mathbf{X}^{(3)} = (\hat{\mathbf{N}}_1 \cdot \hat{\mathbf{N}}_2 \times \hat{\mathbf{N}}_3)\, dX^{(1)} dX^{(2)} dX^{(3)} \\ &= dX^{(1)} dX^{(2)} dX^{(3)}, \end{aligned} \tag{2.14}$$

where $\hat{\mathbf{N}}_i$ is the unit vector along $d\mathbf{X}^i$. Similarly, we can compute the corresponding

volume in the deformed configuration by:

$$\begin{aligned} dv &= d\mathbf{x}^{(1)} \cdot d\mathbf{x}^{(2)} \times d\mathbf{x}^{(3)} \\ &= (\mathbf{F} \cdot \hat{\mathbf{N}}_1) \cdot (\mathbf{F} \cdot \hat{\mathbf{N}}_2) \times (\mathbf{F} \cdot \hat{\mathbf{N}}_3)\, dX^{(1)} dX^{(2)} dX^{(3)} \quad \text{by (2.13)} \\ &= \det \mathbf{F}\, dX^{(1)} dX^{(2)} dX^{(3)}. \end{aligned} \tag{2.15}$$

The determinant of $\mathbf{F}$ is called the *Jacobian* and it is denoted by $J = \det \mathbf{F}$. Therefore, the relation between the deformed volume dv and the undeformed volume dV may be written as:

$$dv = J dV. \tag{2.16}$$

It is worth noting that the Jacobian J has the physical meaning of being the local ratio of current to reference volume of a material volume element.

2.3.3 Change of surface

For similar reasons, let us find the relation between an element of area in the reference configuration $d\mathbf{A}$ which becomes $d\mathbf{a}$ in the deformed configuration. Considering an arbitrary material line $d\mathbf{L}$ in the reference configuration and noting $d\mathbf{l}$ the same line after deformation, the reference and current volumes are given respectively by:

$$d\mathbf{V} = d\mathbf{L} \cdot d\mathbf{A} \tag{2.17}$$

$$\text{and} \quad d\mathbf{v} = d\mathbf{l} \cdot d\mathbf{a}. \tag{2.18}$$

Using (2.16) which relates the reference and deformed volumes and recalling that $d\mathbf{l} = \mathbf{F}\, d\mathbf{L}$, we have:

$$J\, d\mathbf{L} \cdot d\mathbf{A} = (\mathbf{F}\, d\mathbf{L}) \cdot d\mathbf{a}. \tag{2.19}$$

Since this expression is valid for any vector $d\mathbf{L}$ and by using the property $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b}$, the sought relation between an element of area in the reference configuration $d\mathbf{A}$ and its corresponding area $d\mathbf{a}$ in the deformed configuration may be expressed as:

$$d\mathbf{a} = J\, \mathbf{F}^{-T}\, d\mathbf{A}. \tag{2.20}$$

2.3.4 Volumetric and isochoric components

We recall that the Jacobian J has the physical meaning of being the local ratio of current to reference volume of a material volume element. Therefore, if $J = 1$ the volume does not change locally during the deformation and the latter is qualified as *isochoric* at this given particle P. If $J = 1$ everywhere in the body, the deformation of the body is isochoric.

When dealing with incompressible and nearly incompressible materials it is necessary to separate the volumetric from the isochoric components of the deformation. Such a separation must ensure that the isochoric component $\hat{\mathbf{F}}$ does not imply any change in volume. This condition can be achieved by choosing $\hat{\mathbf{F}}$ as:

$$\hat{\mathbf{F}} = J^{-1/3} \mathbf{F}. \tag{2.21}$$

Indeed,

$$\begin{aligned}\det \hat{\mathbf{F}} &= \det(J^{-1/3}\mathbf{F}) \\ &= (J^{-1/3})^3 \det \mathbf{F} \\ &= J^{-1}J \\ &= 1. \end{aligned} \tag{2.22}$$

The deformation gradient $\mathbf{F}$ can now be expressed in terms of the volumetric and isochoric component J and $\hat{\mathbf{F}}$ as:

$$\mathbf{F} = J^{1/3}\hat{\mathbf{F}}. \tag{2.23}$$

2.4 Strain measures

The length of a material curve from the reference configuration can change when displaced to a curve in the deformed configuration. If all the curves do not change length, it is said that a rigid body displacement occurred. The concept of strain is used to evaluate how much a given displacement differs locally from a rigid body displacement (Lubliner, 2006). Therefore, although we know how to transform vectors from the reference configuration into vectors in the current configuration using the deformation gradient, it is more useful to find a measure of the change in length of $d\mathbf{X}$. Many measures of strains can be defined and the most common ones will now be introduced.

2.4.1 Cauchy-Green deformation tensors

Let us consider two particles P_1 and P_2 in the neighbourhood of each other, separated by $d\mathbf{X}$ in the reference configuration. In the deformed configuration P_1 and P_2 occupy the positions $\tilde{P}_1$ and $\tilde{P}_2$ and they are separated by $d\mathbf{x}$. We are interested in the change of distance between the two points P_1 and P_2 as the body deforms.

The squared distances between P_1 and P_2 and $\tilde{P}_1$ and $\tilde{P}_2$ are respectively given by:

$$(dS)^2 = d\mathbf{X} \cdot d\mathbf{X} \tag{2.24}$$

$$\begin{aligned}(ds)^2 &= d\mathbf{x} \cdot d\mathbf{x} = (\mathbf{F}\, d\mathbf{X}) \cdot (\mathbf{F}\, d\mathbf{X}) \\ &= (\mathbf{F}\, d\mathbf{X})^T(\mathbf{F}\, d\mathbf{X}) = (d\mathbf{X}^T\, \mathbf{F}^T)(\mathbf{F}\, d\mathbf{X}) = d\mathbf{X}^T(\mathbf{F}^T\mathbf{F}\, d\mathbf{X}) \\ &= d\mathbf{X} \cdot (\mathbf{F}^T\mathbf{F}\, d\mathbf{X}). \end{aligned} \tag{2.25}$$

using the property that the dot product between two vectors a and b can also be expressed as the simple product between the transpose of a and b ($a \cdot b = a^T b$). We define the *right Cauchy-Green deformation tensor* $\mathbf{C}$ as:

$$\mathbf{C} = \mathbf{F}^T\mathbf{F}. \tag{2.26}$$

Thus, the change of distance between the two points P_1 and P_2 after deformation of the continuous body may be written as:

$$(ds)^2 = d\mathbf{X} \cdot (\mathbf{C}\, d\mathbf{X}). \tag{2.27}$$

By definition, $\mathbf{C}$ is a symmetric second-order tensor. The transpose of $\mathbf{C}$ is denoted $\mathbf{B}$ and is called the *left Cauchy-Green deformation tensor*:

$$\mathbf{B} = \mathbf{F}\mathbf{F}^T. \tag{2.28}$$

2.4.2 Green-Lagrange strain tensor

Using (2.24) and (2.27), the difference in the squared lengths between the reference and the current configuration may be evaluated as:

$$\begin{aligned}(ds)^2 - (dS)^2 &= d\mathbf{X} \cdot (\mathbf{C}\, d\mathbf{X}) - d\mathbf{X} \cdot d\mathbf{X} \\ &= d\mathbf{X} \cdot (\mathbf{C} - \mathbf{I})\, d\mathbf{X}.\end{aligned} \tag{2.29}$$

Let us define the *Green-Lagrange strain tensor* $\mathbf{E}$ as:

$$\mathbf{E} = \frac{1}{2}(\mathbf{C} - \mathbf{I}) \tag{2.30}$$

so we can write:

$$(ds)^2 - (dS)^2 = 2\, d\mathbf{X} \cdot \mathbf{E}\, d\mathbf{X}. \tag{2.31}$$

By definition, the Green-Lagrange strain tensor is a symmetric second-order tensor. Moreover, we observe that the change in squared lengths is zero if and only if the Green-Lagrange strain tensor $\mathbf{E} = \mathbf{0}$. Using (2.11), the Green strain tensor may be expanded as the following:

$$\begin{aligned}\mathbf{E} &= \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I}) \\ &= \frac{1}{2}\left[(\nabla_0\mathbf{u} + \mathbf{I})^T(\nabla_0\mathbf{u} + \mathbf{I}) - \mathbf{I}\right] \\ &= \frac{1}{2}\left[(\nabla_0\mathbf{u})^T + \nabla_0\mathbf{u} + (\nabla_0\mathbf{u})^T(\nabla_0\mathbf{u})\right].\end{aligned} \tag{2.32}$$

The Green strain tensor can be expressed in terms of its components in any coordinate system. In particular, in the Cartesian coordinate system (X_1, X_2, X_3) the components E_{ij} of $\mathbf{E}$ are the following:

$$E_{i,j} = \frac{1}{2}\left(\frac{\partial u_i}{\partial Xj} + \frac{\partial u_j}{\partial Xi} + \frac{\partial u_k}{\partial Xi}\frac{\partial u_k}{\partial Xj}\right), \quad i = 1, 2, 3. \tag{2.33}$$

The components E_{11}, E_{22} and E_{33} are called *normal strains* and it can be shown that they are in fact the ratio of the change in length to the original length along each of the three unit vectors. The components E_{12}, E_{23} and E_{13} are called *shear strains* and they can be interpreted as a measure of the change in angle between line elements that were perpendicular to each other in the undeformed configuration.

2.4.3 Cauchy and Euler tensor

The change in the squared lengths during the body deformation can also be expressed relative to the current length. The length dS can be written in terms of $d\mathbf{x}$ as:

$$(dS)^2 = d\mathbf{X} \cdot d\mathbf{X} = d\mathbf{x} \cdot (\mathbf{F}^{-T}\mathbf{F}^{-1}) d\mathbf{x} = d\mathbf{x} \cdot \tilde{\mathbf{B}} d\mathbf{x} \tag{2.34}$$

where $\tilde{\mathbf{B}} = \mathbf{F}^{-T}\mathbf{F}^{-1}$ is called the *Cauchy strain tensor*. $\tilde{\mathbf{B}}$ is in fact the inverse of the left Cauchy-Green tensor $\mathbf{B}$ introduced previously.

In a similar way we defined the Green strain tensor, we can write the change in the squared lengths but relative to the current length:

$$(ds)^2 - (dS)^2 = 2\, d\mathbf{x} \cdot \mathbf{e}\, d\mathbf{x}. \tag{2.35}$$

where $\mathbf{e}$ is called *Almansi-Hamel strain tensor* or simply *Euler strain tensor*.

2.4.4 Principal strains and invariants

The components ε_{ij} of the strain tensor depend on the coordinate system at the point under consideration. However, the strain tensor itself is a physical quantity and as such, it is independent of the coordinate system chosen to represent it. Certain operations on strain tensors give the same result independently of the coordinate system chosen to represent the components of strain. As an example, a vector is a simple tensor of rank one. In three dimensions, it has three components. The value of these components will depend on the coordinate system chosen to represent the vector, but the length of the vector is a physical quantity (a scalar) and is independent of the coordinate system chosen to represent the vector. Similarly, every second rank tensor (such as the strain tensors) has three independent invariant quantities associated with it. One set of such *invariants* are the principal strains of the strain tensor, which are just the eigenvalues of the strain tensor. Because they are independent of any coordinate system, they are very convenient to define strain energy density functions (see section 2.6.5). The most commonly used invariants are:

$$\begin{aligned} I_1 &= tr(\varepsilon) \\ I_2 &= \frac{1}{2}\left\{tr(\varepsilon^2) - [tr(\varepsilon)]^2\right\} \\ I_3 &= det(\varepsilon). \end{aligned} \tag{2.36}$$

It can be shown that in the coordinate system $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$ formed by the three eigenvectors, the expression of the strain tensor is:

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 & 0 & 0 \\ 0 & \varepsilon_2 & 0 \\ 0 & 0 & \varepsilon_3 \end{bmatrix}. \tag{2.37}$$

Since there are no shear strain components in this particular coordinate system, the principal strains represent the maximum and minimum stretches of an elemental volume. Because of the obvious simplicity of the strain tensor's expression, this coordinate system is often used in mechanics to derive and solve equations.

2.4.5 Infinitesimal strain tensor

In some cases, it is possible to simplify the expression of the Green strain tensor defined in (2.32). Indeed, when the displacement gradients are small (that is, $|\nabla \mathbf{u}| \ll 1$) we can neglect the non-linear terms in the definition. In the case of infinitesimal strains, the Green-Lagrange strain tensor and the Eulerian strain tensor are approximately the same and can be approximated by the infinitesimal strain tensor denoted $\boldsymbol{\varepsilon}$ and is given by:

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left[\nabla \mathbf{u} + (\nabla \mathbf{u})^T\right]. \tag{2.38}$$

Its Cartesian components ε_{ij} are the following:

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial X_j} + \frac{\partial u_j}{\partial X_i}\right). \tag{2.39}$$

The strain quantities defined in the previous sections are non-linear expressions in terms of the mapping χ and will lead to non-linear governing equations. In solid mechanics, whenever the hypothesis is acceptable, it is common practice to assume that the displacements are small and the infinitesimal strain tensor is then used as a measure of the deformation.

2.5 Stress

Stress is a measure of the intensity of the internal forces acting between particles of a deformable body across imaginary internal surfaces. These internal forces are produced between the particles in the body as a reaction to external forces applied on the body. The SI unit for stress is pascal (symbol Pa), which is equivalent to one newton (force) per square metre (unit area). As with strain, stress can be measured per unit of deformed area (section 2.5.1) or undeformed area (sections 2.5.2 and 2.5.3). In general, the stress is not uniformly distributed over the cross section of a material body, and consequently the stress at a point on a given area is different from the average stress over this entire area. Therefore, it is necessary to define the stress not over a given area but at a specific point in the body. The stress at a point in a three-dimensional continuum is fully defined by nine quantities, three per plane, on three mutually perpendicular planes at the point. The *stress tensor* is defined as the second-order tensor which components are these nine quantities.

2.5.1 Cauchy stress

The Euler-Cauchy's stress principle states that on any surface (real or imaginary) that divides the body, the action of one part of the body on the other is equivalent to the system of distributed forces and couples on the surface dividing the body (Truesdell and Toupin, 1960). With that consideration in mind, let us consider a plane S that passes through an arbitrary internal point P which has a unit normal

vector $\mathbf{n}$. This plane separates the body into two regions and we are interested in the forces applied by one region onto the other. First we will introduce the true stress, defined as the stress in the deformed configuration χ measured per unit area of the deformed configuration. The force $\Delta\mathbf{f}(\mathbf{n})$ acting on a small element of area Δa in a continuous medium depends not only on the size of the area but also depends on the orientation of this area $\mathbf{n}$. The *Cauchy stress vector* at P on S can be defined as:

$$\mathbf{t}(\mathbf{n}) = \lim_{\Delta a \to 0} \frac{\Delta \mathbf{f}(\mathbf{n})}{\Delta a}. \tag{2.40}$$

This equation means that the stress vector depends on its location in the body and the orientation of the plane on which it is acting. In general, the stress vector is not perpendicular to that plane and it can be separated into two components: one component normal to the plane called *normal stress* and one component tangent to the plane named *shear stress*. Since $\mathbf{t}$ depends on $\mathbf{n}$ but is not in the direction of $\mathbf{n}$, it may be interesting to look into the relationship between $\mathbf{t}$ and $\mathbf{n}$.

In the following we make use of a Cartesian coordinate system and we consider an infinitesimal tetrahedron as shown on Fig. 2.2. We note $-\mathbf{t}_1$, $-\mathbf{t}_2$, $-\mathbf{t}_3$ and $\mathbf{t}$ the

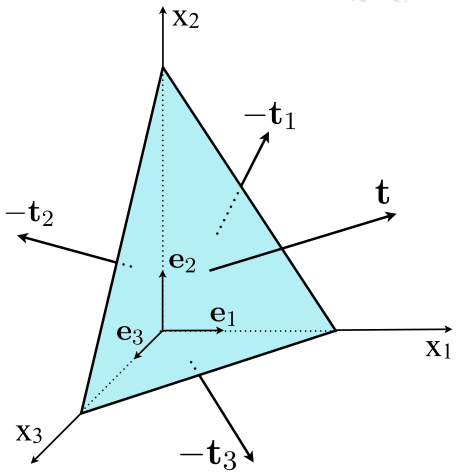


Figure 2.2: Tetrahedral element in Cartesian coordinates.

stress vectors in the outwards directions on the faces of the tetrahedron whose areas are respectively Δa_1, Δa_2, Δa_3 and Δa. For the mass m inside the tetrahedron, we have by Newton's second law:

$$\sum \mathbf{F} = \mathbf{t}\Delta a - \mathbf{t}_1\Delta a_1 - \mathbf{t}_2\Delta a_2 - \mathbf{t}_3\Delta a_3 + \Delta v\mathbf{f} = m\mathbf{a}, \tag{2.41}$$

where $\mathbf{f}$ is the force per unit volume acting on the body, Δv the volume of the tetrahedron and $\mathbf{a}$ the acceleration. The areas Δa_1, Δa_2 and Δa_3 being the projections of Δa, they are related to Δa in the Cartesian coordinate system $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ by:

$$\Delta a_i = (\mathbf{n} \cdot \mathbf{e}_i)\Delta a, \quad i = 1, 2, 3. \tag{2.42}$$

Dividing (2.41) by Δa, using (2.42) and noting that $\Delta v/\Delta a \to 0$ when the tetrahedron shrinks to a point yields:

$$\begin{aligned}\mathbf{t} &= (\mathbf{n}\cdot\mathbf{e}_1)\mathbf{t}_1 + (\mathbf{n}\cdot\mathbf{e}_2)\mathbf{t}_2 + (\mathbf{n}\cdot\mathbf{e}_3)\mathbf{t}_3 \\ &= \mathbf{n}\cdot(\mathbf{e}_1\mathbf{t}_1 + \mathbf{e}_2\mathbf{t}_2 + \mathbf{e}_3\mathbf{t}_3).\end{aligned} \tag{2.43}$$

We define the *Cauchy stress tensor* $\sigma = \mathbf{e}_1\mathbf{t}_1 + \mathbf{e}_2\mathbf{t}_2 + \mathbf{e}_3\mathbf{t}_3$ and we have:

$$\mathbf{t}(\mathbf{n}) = \mathbf{n}\cdot\sigma. \tag{2.44}$$

The stress vector $\mathbf{t}$ represents the vectorial stress on a plane whose normal is $\mathbf{n}$. As we demonstrated, the stress tensor is a property of the medium that is independent of $\mathbf{n}$. It represents the current force per unit of deformed area: $d\mathbf{f} = \mathbf{t}da = \sigma\cdot d\mathbf{a}$.

The matrix form of (2.44) in a Cartesian coordinate system is given by:

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} \sigma_{11} & \sigma_{21} & \sigma_{31} \\ \sigma_{12} & \sigma_{22} & \sigma_{32} \\ \sigma_{13} & \sigma_{23} & \sigma_{33} \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix}. \tag{2.45}$$

The Cauchy stress tensor is used for stress analysis of material bodies experiencing small deformations. This stress is basically defined as force/(unit deformed area). The strain measure that is appropriate to use with the Cauchy stress tensor is therefore the infinitesimal strain tensor (see section 2.4.5). Indeed, because the Cauchy stress tensor is defined by unit of deformed area, it is not appropriate for analysing materials undergoing large deformation where the area in the deformed configuration is often unknown. Consequently, we require alternative stress tensors based on the reference configuration. We will introduce two new stress measures: the first Piola-Kirchhoff stress tensor and the second Piola-Kirchhoff stress tensor.

2.5.2 First Piola-Kirchhoff stress tensor

We know that the Cauchy stress tensor represents the current force per unit of deformed area: $d\mathbf{f} = \mathbf{t}(\mathbf{n})da = \sigma\cdot d\mathbf{a}$. We state that we can find a stress vector $\mathbf{T}$ over the area element $d\mathbf{A}$ in the underformed configuration which results in the same total force.

$$d\mathbf{f} = \mathbf{t}(\mathbf{n})da = \mathbf{T}(\mathbf{N})dA. \tag{2.46}$$

The two stress vectors have the obviously the same direction but different magnitudes since they are not applied to the same area.

In a similar way we defined the Cauchy stress σ with $\mathbf{t}(\mathbf{n}) = \sigma\cdot\mathbf{n}$, we introduce a stress tensor $\mathbf{P}$ such that $\mathbf{T}(\mathbf{n}) = \mathbf{P}\cdot\mathbf{N}$. We call $\mathbf{P}$ the *first Piola-Kirchhoff stress tensor* and gives the current force per unit undeformed area.

The relation between the first Piola-Kirchhoff stress tensor and the Cauchy stress tensor can be obtained as follows. From (2.46) we have:

$$\mathbf{T} = \frac{da}{dA}\mathbf{t}. \tag{2.47}$$

Using $d\mathbf{f} = \mathbf{t}da$ and the relation (2.20) between $d\mathbf{A}$ and $d\mathbf{a}$, we have:

$$\mathbf{T} = J\,\mathbf{t}\,\mathbf{F}^{-T}. \tag{2.48}$$

Finally we can express the first Piola-Kirchhoff stress tensor $\mathbf{P}$ as

$$\mathbf{P} = J\,\sigma\,\mathbf{F}^{-T}. \tag{2.49}$$

2.5.3 Second Piola-Kirchhoff stress tensor

Let us consider the stress tensor $\mathbf{S}$ resulting in the force $d\mathbf{F}$ in the undeformed area $d\mathbf{A}$ which corresponds to the force $d\mathbf{f}$ in the deformed area $d\mathbf{a}$. We define this stress tensor $\mathbf{S}$ as the *second Piola-Kirchhoff stress tensor*.

$$d\mathbf{F} = \mathbf{S}\,d\mathbf{A}. \tag{2.50}$$

In other words, the second Piola-Kirchhoff stress tensor gives the transformed current force per unit undeformed area. Similar to the relationship (2.13) between $d\mathbf{x}$ and $d\mathbf{X}$, the force $d\mathbf{f}$ is related to the force $d\mathbf{F}$ via the deformation gradient tensor:

$$\begin{aligned} d\mathbf{F} &= \mathbf{F}^{-1}\,d\mathbf{f} \\ &= \mathbf{F}^{-1}\,(\mathbf{P}\,d\mathbf{A}) \\ &= \mathbf{S}\,d\mathbf{A}. \end{aligned} \tag{2.51}$$

Therefore, the second Piola-Kirchhoff stress tensor is related to the first Piola-Kirchhoff stress tensor as follows:

$$\mathbf{S} = \mathbf{F}^{-1}\mathbf{P} = J\,\mathbf{F}^{-1}\,\sigma\,\mathbf{F}^{-T}. \tag{2.52}$$

2.5.4 Principal stresses and invariants

In the same way that we defined principal strains in section 2.4.4 and like any second-rank tensor, the stress tensor has at least three invariants and we can find a coordinate system $(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$ in which the expression of the stress tensor is:

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix}. \tag{2.53}$$

This result means that at every point in a stressed body there are at least three planes called *principal planes*, with normal vectors $\mathbf{n}_1$, $\mathbf{n}_2$ and $\mathbf{n}_3$ called *principal directions* (in fact, the eigenvectors), where the corresponding stress vector is perpendicular to the plane and where there are no normal shear stresses. The three stresses normal σ_1, σ_2 and σ_3 to these principal planes are called *principal stresses*.

2.6 Constitutive equations

So far, all the relations that we have established are valid for every continuum. Indeed, the derivations carried out so far made no mention of any material. Therefore, we know that these relations are not sufficient to describe the response of a material to a given loading. Moreover, because the stresses result from the deformation of the material, they may be expressed in terms of some measure of this deformation such as the strain. This relationship, which obviously depends on the type of material under consideration, is what we call a *constitutive equation*. Constitutives equations are an attempt to mathematically characterise the behaviour of materials observed from experimentals results.

Naturally, the relationship between strain and stress is conditioned by the constitution of the material of interest. Because the mechanics involved in materials can be quite complex (tissue mechanics in particular), many simplifying assumptions are required to derive a constitutive equation. In practice, a given constitutive equation can only cover a subset of the actual mechanical behaviour and could not possibly model all aspects of tissue behaviour under any type of loading accurately. For instance, some tissues behave very differently under small loads as compared to large loads. The behaviour may also depends on loading rates and damage may even occur under large loads. Moreover, not only constitutive equations need to deal with the complexity of material mechanics but they must remain simple enough to measure all the included material constants experimentally. This section will now introduce the most common constitutive models relevant to soft-tissue modelling.

2.6.1 Elasticity

If under applied loads a material stores but does not dissipate energy, and it returns to its orginal shape when the loads are removed, we call that material *elastic*. The work done by the stress is generally dependent on the deformation path. However, for elastic materials the stress does not depend on the path of deformation and therefore the state of stress in the deformed configuration is determined only by the state of deformation. It means that the stress does not depend on the time taken to achieve the deformation or the rate at which the state of deformation is reached.

In addition, if there exists a *strain energy density* function $U_0(\varepsilon)$ such that

$$\boldsymbol{\sigma} = \frac{\partial U_0}{\partial \varepsilon}, \tag{2.54}$$

the material is said to be hyperelastic. However, for *imcompressible* materials the stress tensor is not completely determined by the deformation. The hydrostatic pressure[1] affects the stress and the above equation is then written as:

$$\boldsymbol{\sigma} = -p\mathbf{I} + \frac{\partial U_0}{\partial \varepsilon}, \tag{2.55}$$

[1] pressure exerted by a fluid at equilibrium due to the force of gravity

where p is the hydrostatic pressure. In order to develop a mathematical model of an hyperelastic material, U_0 is expanded in Taylor's series about $\varepsilon = 0$:

$$U_0 = C_0 + C_{ij}\varepsilon_{ij} + \frac{1}{2!}\hat{C}_{ijkl}\varepsilon ij\varepsilon_{kl} + \frac{1}{3!}\hat{C}_{ijklmn}\varepsilon_{ij}\varepsilon_{kl}\varepsilon_{mn} + \ldots, \tag{2.56}$$

where C_0, C_{ij} and $\hat{C}$ are material stiffnesses. The form of U_0 may varied for different materials. A material is said to be *linear* if the relationship between strain and stress is also linear. Based on this definition, U_0 is a cubic or higher-order function of the strains for non-linear materials whereas U_0 is a quadratic function of strain for linear materials. We will start by discussing linear materials for the case of infinitesimal deformations (sections 2.6.2 to 2.6.4). Consequently, we will not make any distinction between the various measures of stress and strain and use $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$ for stress and strain tensors respectively.

2.6.2 Linear materials: generalised Hooke's law

The linear constitutive model for infinitesimal deformations is called the generalised Hooke's law. To derive the relation between stress and strain for a linear material, we assume the quadratic form of U_0:

$$U_0 = C_0 + C_{ij}\varepsilon_{ij} + \frac{1}{2!}\hat{C}_{ijkl}\varepsilon ij\varepsilon_{kl}. \tag{2.57}$$

From there and using (2.54) it can be shown that there exists some coefficients C_{ijkl} such as:

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl}. \tag{2.58}$$

The C_{ijkl} are the coefficients of the fourth-order tensor $\mathbf{C}$ that relates the second-order tensors of strain and stress. The 81 scalar components of this fourth-order tensor ($= 3^4$) can be reduced to 21 independent coefficients using existing symmetries. Recalling that both strain and stress tensors are symmetric yields:

$$\begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \sigma_{23}\\ \sigma_{13}\\ \sigma_{12}\end{bmatrix} = \begin{bmatrix} C_{1111} & C_{1122} & C_{1133} & C_{1123} & C_{1113} & C_{1112}\\ C_{1122} & C_{2222} & C_{2233} & C_{2223} & C_{2213} & C_{2212}\\ C_{1133} & C_{2233} & C_{3333} & C_{3323} & C_{3313} & C_{3312}\\ C_{1123} & C_{2223} & C_{3323} & C_{2323} & C_{2313} & C_{2312}\\ C_{1113} & C_{2213} & C_{3313} & C_{2313} & C_{1313} & C_{1312}\\ C_{1112} & C_{2212} & C_{3312} & C_{2312} & C_{1312} & C_{1212}\end{bmatrix} \begin{bmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ \varepsilon_{33}\\ \varepsilon_{23}\\ \varepsilon_{13}\\ \varepsilon_{12}\end{bmatrix}. \tag{2.59}$$

2.6.3 Orthotropic materials

Further reduction in the number of independent stiffness parameters can be achieved using the so-called material symmetry. This symmetry is due to the internal structure of the material, that is the arrangement of molecules (crystallographic form). We note that the symmetry under discussion is a directional property and not a positional property. Thus, a material may have certain elastic symmetry at every

point of a material body and the properties may vary from point to point. Positional dependence of material properties is what we called the inhomogeneity of the material.

Orthotropic materials are characterised by three mutually orthogonal planes of material symmetry, that is, three mutually orthogonal preferred directions. In this case, the number of coefficients is reduced to 9 and the stress-strain relation for an orthotropic material is the following:

$$\begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \sigma_{23}\\ \sigma_{13}\\ \sigma_{12}\end{bmatrix} = \begin{bmatrix} C_{1111} & C_{1122} & C_{1133} & 0 & 0 & 0\\ C_{1122} & C_{2222} & C_{2233} & 0 & 0 & 0\\ C_{1133} & C_{2233} & C_{3333} & 0 & 0 & 0\\ 0 & 0 & 0 & C_{2323} & 0 & 0\\ 0 & 0 & 0 & 0 & C_{1313} & 0\\ 0 & 0 & 0 & 0 & 0 & C_{1212}\end{bmatrix} \begin{bmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ \varepsilon_{33}\\ \varepsilon_{23}\\ \varepsilon_{13}\\ \varepsilon_{12}\end{bmatrix}. \tag{2.60}$$

Most often, the material properties are determined in a laboratory in terms of the engineering constants such as Young's modulus, shear modulus, and so on. These constants are measured using simple tests like uniaxial tension test or pure shear test. Because of their direct and obvious physical meaning, engineering constants are used in place of the more abstract stiffness coefficients C_{ijkl}. Therefore, we will now reformulate (2.60) with respect to those engineering constants. Noting E_i the Young's modulus in the direction x_i and G_{ij} the shear moduli in the $x_i - x_j$ plane, it can be shown that:

$$\begin{bmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ \varepsilon_{33}\\ \varepsilon_{23}\\ \varepsilon_{13}\\ \varepsilon_{12}\end{bmatrix} = \begin{bmatrix} \frac{1}{E_1} & -\frac{\nu_{21}}{E_2} & -\frac{\nu_{31}}{E_3} & 0 & 0 & 0\\ -\frac{\nu_{12}}{E_1} & \frac{1}{E_2} & -\frac{\nu_{32}}{E_3} & 0 & 0 & 0\\ -\frac{\nu_{13}}{E_1} & -\frac{\nu_{23}}{E_2} & \frac{1}{E_3} & 0 & 0 & 0\\ 0 & 0 & 0 & \frac{1}{G_{23}} & 0 & 0\\ 0 & 0 & 0 & 0 & \frac{1}{G_{13}} & 0\\ 0 & 0 & 0 & 0 & 0 & \frac{1}{G_{12}}\end{bmatrix} \begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \sigma_{23}\\ \sigma_{13}\\ \sigma_{12}\end{bmatrix}. \tag{2.61}$$

2.6.4 Isotropic materials

A material is said *isotropic* if its properties are independent of directions. Therefore, we have the following relations:

$$E_1 = E_2 = E_3 = E, \quad G_{12} = G_{13} = G_{23} = G, \quad \nu_{12} = \nu_{23} = \nu_{13} = \nu. \tag{2.62}$$

The stress-strain relations may be written:

$$\sigma_{ij} = \frac{E}{1+\nu}\varepsilon_{ij} + \frac{\nu E}{(1+\nu)(1-2\nu)}\varepsilon_{kk}\delta_{ij}. \tag{2.63}$$

where the summation of repeated indices is implied (Einstein's notation) and δ_{ij} is the Kronecker delta[2]. The inverse relations are:

$$\varepsilon_{ij} = \frac{1+\nu}{E}\sigma_{ij} - \frac{\nu}{E}\sigma_{kk}\delta_{ij}, \tag{2.64}$$

recalling that $G = E/[2(1+\nu)]$.

2.6.5 Non-linear materials

For most materials, the linear relation between stress and strain is only valid for small deformation. Beyond a certain threshold, Hooke's law is no longer valid. Note that it does mean that the material is not elastic, it may well recover all its deformation after removal of the loads. However, materials often exibit an elastic threshold in the non-linear range after which permanent deformation occurs. This range of deformation is qualified of *plastic*.

The strain energy density function takes different forms for different materials. It is often expressed as a linar combination of unknown coefficients (determined experimentally) and principal invariants of Green strain tensor $\mathbf{E}$, left Cauchy-Green strain tensor $\mathbf{B}$ and deformation gradient tensor $\mathbf{F}$ (see section 2.4.4 for more information on invariants). The constitutive equations of two well-known non-linear elastic materials (Mooney-Rivlin and neo-Hookean) will now be introduced.

Mooney-Rivlin

The Mooney-Rivlin model was proposed by Melvin Mooney and Ronald Rivlin in two independent papers in 1952. For an incompressible Mooney-Rivlin material, the strain energy density function U_0 is taken as a linear function of the principal invariants of the left Cauchy-Green strain tensor $\mathbf{B}$ and is written as:

$$U_0 = C_1(I_B - 3) + C_2(II_B - 3), \tag{2.65}$$

where C_1 and C_2 are two constants and I_B and II_B are the two principal invariants of $\mathbf{B}$. For those materials, it can be shown that the stress tensor has the form

$$\sigma = -p\mathbf{I} + \alpha\mathbf{B} + \beta\mathbf{B}^{-1}, \tag{2.66}$$

where ρ is the material density and α and β are given by

$$\alpha = 2\rho\frac{\partial U_0}{\partial I_B} = 2\rho C_1, \quad \beta = -2\rho\frac{\partial U_0}{\partial II_B} = -2\rho C_2. \tag{2.67}$$

The Mooney-Rivlin incompressible model is often used to represent the behaviour of rubber-like materials.

[2] δ_{ij} is equal to 1 if $i = j$ and to 0 if $i \neq j$

Neo-Hookean

The neo-Hookean model was proposed by Ronald Rivlin in 1948. It can be derived from the Mooney-Rivlin model in the special case where $C_2 = 0$ in the strain energy density function. Therefore, we have:

$$U_0 = C_1(I_B - 3). \tag{2.68}$$

and the constitutive equation for the stress may be written as:

$$\sigma = -p\mathbf{I} + 2\rho C_1 \mathbf{B}. \tag{2.69}$$

The neo-Hookean model provides a reasonable approximation of rubber-like material for moderate strains. TODO: Do I keep this range of validity, which seems somewhat limited... It is typically accurate for strains less than 20 % (Gent, 2001).

2.6.6 Viscoelastic materials

We know that materials for which the mechanical behaviour depends solely on the state of deformation are called elastic. In contrast, the mechanical behaviour of *viscous* materials is only a function of the current rate of deformation (like honey for instance). Viscoelastic materials have elements of both of these properties and, as such, exhibit time dependent strain. Depending on the change of strain rate versus stress inside a material the viscosity can be categorised as having a linear or non-linear response. When a material exhibits a linear response it is categorised as a Newtonian material. In this case the stress is linearly proportional to the strain rate. If the material exhibits a non-linear response to the strain rate, it is categorised as a Non-Newtonian material.

Viscoelastic materials present a few characteristics:

1. if the stress is held constant, the strain increases with time (creep)
2. if the strain is held constant, the stress decreases with time (relaxation)
3. the effective stiffness depends on the rate of application of the load
4. if cyclic loading is applied, hysteresis occurs, leading to a dissipation of mechanical energy

In other words, the resistance of a viscoelastic material to a given force depends on the velocity to which this force is applied.

2.7 Tissue characterisation

If continuum mechanics does teach us how to model idealised continuous structures, modelling soft tissues remains very complex. Indeed, in practice soft tissues are complex structures: they do not consist of a simple continuous piece of material,

they are made of various constituents. Mechanical properties of soft tissues not only depends on the properties of these constituents, but how they are arranged relative to each other. This elaborate arrangement often leads to material properties for the global structure that are non-homogenous, non-linear, anisotropic and even viscoelastic.

As we know, the constitutive equations are used to define idealised materials, which represent different aspects of the mechanical behaviour of natural materials. However, in practice, no real material is known to behave exactly as one of these mathematical models. In particular, few biological tissues obey Hooke's law. If because of their high stiffness (and therefore small deformations) cortical bones are accurately described by a linear model, Hooke's law becomes insufficient to describe the mechanical behaviour of the liver with precision for instance. In fact, in vivo experiments conducted by Melvin et al. (1973) suggest that mechanical behaviour of liver is non-linear and viscoelastic. The brain also appears to be non-linear and viscoelastic (Miller and Chinzei, 1997). The liver has the additional property of anisotropy, more precisely, it is transversely isotropic according to Chui et al. (2007).

These various mechanical aspects of materials make realistic modelling of anatomical structures fairly challenging. The development of algorithms that allow the interactive computation of the deformation of soft tissues is very difficult. Therefore, in medical simulation, where modelling must occur in real-time or close to real-time, very often simplifications have to be made and only the key characteristics of the material that are relevant for the objective of the simulation are considered and simulated. Experiments have to be performed to check the relevancy of the chosen constitutive model and to determine its parameters. At last, an optimised implementation of the model is required to perform in real-time.

One of the main obstacles in developing realistic organ models is the lack of data on the material properties of live organ tissues. Measuring and characterising in vivo organ properties is a highly challenging task, but is a requirement for realistic organ modelling. Indeed, as we have seen in sections 2.60 and 2.6.5, constitutive equations require the experimental determination of material constants. Organ models with incorrect material properties will affect training in surgical simulator systems (Sedef et al., 2006). Some devices exist for inert materials which allow the measure of Young's modulus, shear modulus and Poisson's ratio. However, the same measures are much more difficult to obtain for living tissues. In particular, experiments confirmed the important differences between in vitro and in vivo measures, as demonstrated by Kerdok et al. (2006) with the viscoelastic properties of the liver for instance. Absence of blood perfusion and difference in temperature are two of most important factors to explain these differences. Obviously, in vivo constraints are very heavy and developing experiments and devices for in vivo measures of soft tissue is a very complex task. The preparation of samples and their testings (tensile, compression, thermal effects, rate of loading etc.) are not straightforward procedures.

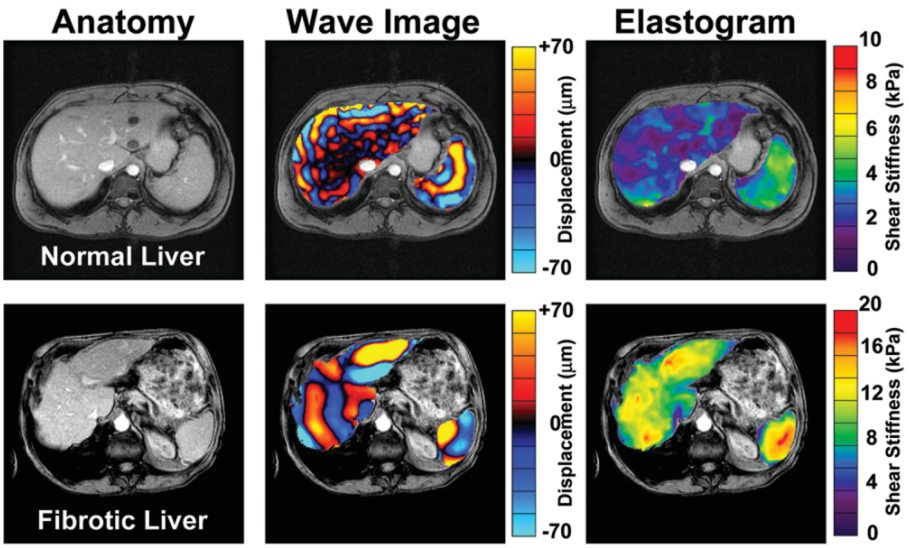


Figure 2.3: Magnetic resonance elastography of the liver in a healthy volunteer and a patient with cirrhosis. Image courtesy of Talwalkar et al. (2007).

A non-invasive technique called *elastography* was introduced recently to measure Young's modulus. The first device based on this technique was commercialised by Echosens in 2003 Bastard (2009). The techniques of elastography may be classified in two categories: static and dynamic elastography. In static elastography, a quasi-static constraint is applied onto the surface of the tissue and its deformation is measured. If the constraint is known, the Young's modulus may be obtained via Hooke's law. Dynamic elastography relies on shear waves propagation in the tissue. Shear waves are generated by an acoustic driver and the displacement caused by the waves is determined by ultrasounds, magnetic resonance imaging (MRI) or optical tomography. As an example, magnetic resonance elastography is a quantitative technique which allow the generation of maps describing the shear modulus within the tissues (see Fig. 2.3). However, elastography is still in development and only a few techniques of elastography are actually used clinically. In the future, elastography might be able to provide a map of the material properties of tissues which may be used as input data for any computational model. This technique would be a step towards patient-specific simulation. As diseased organs are often much stiffer than healthy structures, the integration of these data would greatly increase the fidelity of the simulation.

Yet, precised characterisation of soft tissue is not sufficient to achieve a good simulation. The development of a simplified or optimised version of the model suitable for real-time computation is also mandatory. As we will see in chapter 4 and 7, numerous models are available in the literature to describe physics of objects, from fairly simple and naive approaches to more complex and thorough representations. However, the most common technique to solve equations of continuum mechanics is the finite element method. Moreover, since the two main contributions of this thesis

deal with finite element methods formulations, the general principles of this method will be detailed in the next section.

Chapter 3

A practical approach of the finite element method

The previous chapter explained in details how physics of anatomical structures could be described by continuun mechanics. When an organ is submitted to a force, the first effect of this application is a change in the organ's shape. This deformation can be measured using different strain tensors. By modelling the mechanical aspects of the material with an appropriate constitutive law, we can then deduce the stress tensor (or internal forces) at each point within the organ. The next step is to apply Newton's second law (law of conservation of momentum), which yields a partial differential equation at every point of the volume. The finite element method is a numerical procedure that allows to turn an infinite number of partial differential equations into a finite number of algebraic equations. The method follows different steps. We will first begin by describing the subdivision of the domain into smaller sub-domains into which local simpler equations are derived. We will then discuss how all the local equations are summed over the whole domain which eventually leads to the global solution.

3.1 Introduction

3.1.1 A numerical method

One of the most important things engineers and scientists do is to model physical phenomena. Using assumptions concerning how the phenomena works and using the appropriate laws of physics governing the process, they can derive a mathematical model, often characterised by complex differential or integral equations relating various quantities of interest. However, because of the complexities in the geometry and complex boundary conditions found in real life problems, they cannot analytically solve these equations. A few decades ago, the only possible approach was to drastically simplify them, which was not always sufficient to find an approximate solution. Nowadays, in practice, most of the problems are solved using numerical methods. Indeed, with suitable mathematical models and numerical methods, computers can help solving many practical problems of engineering. Numerical methods typically transform differential equations governing a continuum to a set of algebraic equations of a discrete model of the continuum that are to be solved using computers (Reddy, 1993). The *finite element method* (FEM) is the most popular numerical procedure that is used to approximately solve differential equations, especially in continuum mechanics.

3.1.2 The basic ideas of FEM

The finite element method begins by dividing the structure into small pieces, manageable regions, called *elements*. The collection of all these elements makes up a *mesh* which approximates the problem geometry. Why is this a good idea? If we can expect the solution for an engineering problem to be very complex, it is mostly due to the complex geometry, on which a global solution is difficult to figure out. If the problem domain can be divided (*meshed*) into small elements, the solution within an element is easier to approximate (MacDonald, 2007). Over each finite element, the unknown variables are approximated using known functions. These functions can be linear or higher-order polynomial expressions that depend on the geometrical locations (*nodes*) used to define the finite element shape. The governing equations are integrated over each finite element using linear algebra techniques and the solution over the entire domain problem is obtained by summing (*assembling*) the solution of each element. Thus, the finite element method transforms an infinite number of differential equations (one can be defined at any point of the continuum) into a finite number of algebraic equations (depending on the chosen number of elements).

This approach may be compared to trying to find the area under a curve. We know that we can find the exact solution for the area under the curve by integration. However, sometimes the function describing the curve is not known, or is difficult to integrate. One method to obtain an approximate solution is to break up the area into a series of rectangles and add the areas of all rectangles (see Fig. 3.1). It is worth nothing that the solution accuracy can be increased by reducing the width of

the rectangles to better follow the curve.

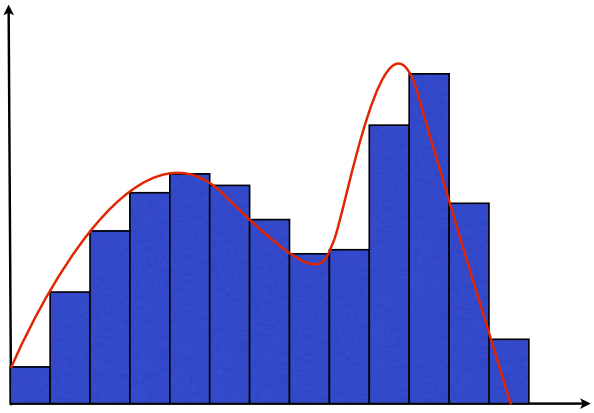


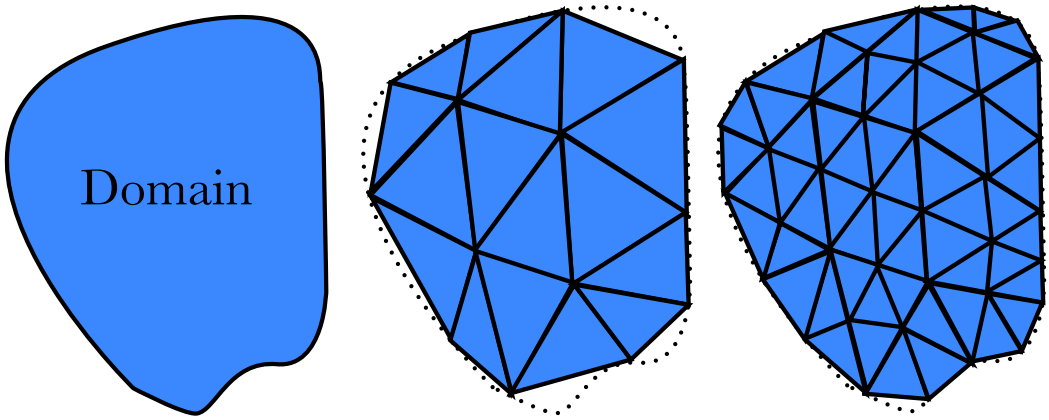
Figure 3.1: Approximation of the area under a curve by adding the areas of all rectangles (left Rieman sum for this example).

It is crucial to keep in mind that approximations occur at different stages during finite element analysis. The division of the whole domain into finite elements may not be exact (see Fig. 3.2), introducing error in the domain being modelled. The second stage is when element equations are derived. As mentionned earlier, the unknowns of the problem are approximated using the idea that any continuous function can be representated by a linear combination of known functions and undetermined coefficients. Algebraic relations between the undetermined coefficients are then obtained by satisfying the governing equations over each element. There are a few types of approach for establishing these equations but, without going into details, the mathematical foundations of all these approaches are energy principles or the *weighted residual methods* (see Appendix B), which both lead to integrals during the process. Therefore, the second stage creates two sources of error: the representation of the solution by a linear combination of functions and the evaluation of the integrals. Finally, errors are introduced when solving the assembled system of algebraic equations.

3.2 Discretisation

3.2.1 Meshing process

The first step in the finite element method is to create a mesh of the domain to study. Mesh generation is a very important task and can be very time consuming. The domain has to be meshed properly into elements of specific shapes. All the elements together form the entire domain of the problem without any gap or overlapping. For example, triangles or quadrilaterals can be used in two dimensions, and tetrahedra and hexahedra in three dimensions. Information, such as *element connectivity*, must be created during the meshing process for later use in the formulation of the FEM



(a) (b) (c)

Figure 3.2: The division of the domain (a) into elements is not always perfect (b). The discretisation error may be reduced by using smaller elements (c). However, this improvement is obtained to the detriment of computational efficiency as the number of algebraic equations to solve will increase.

equations. The number of elements into which the domain is divided in a problem depends mainly on the geometry of the domain and on the desired accuracy of the solution. Usually, the number of elements increases with the complexity of the geometry. For instance, if one part of the domain is thiner (see Fig. 3.2 (c)), the size of the elements must be reduced in order to tile this particular part, which increases the total number of elements, and hence the number of algebraic equations to solve. However, adding elements is sometimes desirable. Indeed, increasing the number of elements tends to get an approximate solution closer to the exact analytical solution (as reducing the width of the rectangles allows for a better approximation of the area under the curve). Elements are classically added in the particular regions of interest for a given problem, if this information is known a priori. A trade-off between accuracy and computational time must be found. This is why the meshing process on the problem domain must be carried out carefully. One needs to create a mesh which gives an accurate enough solution for the desired application while restraining the computational time according to the time constraints of the problem. As an example, finding the maximum load that can sustain a bridge in structural mechanics demands a high degree of accuracy, no matter how much time it is needed to compute the solution. Conversely, organ deformation in medical training simulators must be computed at an interactive rate so that no apparent delay can be observed between the manipulation of a given organ and its deformation. It does not mean that precision is not required, simply that the time constraints will necessarily limit the accuracy of the solution.

3.2.2 Solution interpolation

As we have seen, the finite element method is based on finding an approximate solution over each simple element rather than the whole domain. Any continuous function f may be approximated by a linear combination of known functions ϕ_i and undetermined coefficients c_i:

$$f \approx \tilde{f} = \sum_{i=1}^{n} c_i \phi_i. \tag{3.1}$$

Moreover, the approximation solution u^ewithin the element Ω_e must fullfill certain conditions of continuity in order to be convergent to the actual solution u as the number of elements is increased. The finite element method approximates the solution by the following polynomial expression:

$$u^e(x) = \sum_{i=1}^{n} u_i^e \psi_i^e(x) \tag{3.2}$$

where Ω_e is a one-dimensional element[1], x a position within this element, u_i^e are the values of the solution u at the nodes and $\psi_i^e(x)$ the approximation functions over the element. This particular form will be assumed for brevity but the interested reader may refer to Reddy (1993) for demonstration. Note that u_i^e plays the role of the undetermined coefficients and ψ_i^e the role of approximation functions. Writing the approximation in terms of the nodal values of the solution is necessitated by the fact that we require the solution to be the same at points common to the elements in order to connect the approximate solution from each element and form a continuous solution over the whole domain. The approximation ψ_i^e can be linear or higher-order polynomial expressions and are called interpolation functions. Depending on the degree of polynomial approximation used to represent the solution, additional nodes may be identified inside the element. It is worth noting that the type of interpolation directly affects the accuracy of the solution. In other words, finding the solution to the problem consists of merely figuring out the values of the sought variable at every node of the mesh. Its value at any other point of the domain may then be deducted by interpolation within elements.

The next logical question is how to derive the interpolation functions ψ_i^e for a given element. Their derivation depends only on the geometry of the element and the number and location of the nodes. The number of nodes must be equal to the number of terms in the polynomial expansion. Therefore each element contains a single interpolation function for each of its nodes. As stated by interpolation theory, each interpolation function is required to be equal to 1 at its corresponding node, and 0 at all other nodes:

$$\psi_i^e(x_j) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{3.3}$$

[1]For the sake of simplicity, all derivations are carried out in 1D but remain valid for each component in higher dimensions.

where x_j is the position of node j. From there, we can state that the interpolation function at node i may be written as:

$$\psi_i^e(x) = C_i \prod_{i \neq j}^{n-1} (x - x_j) \tag{3.4}$$

where j are the indices of the other nodes and C_i is a constant to be determined such as:

$$\psi_i^e(x_i) = 1. \tag{3.5}$$

This function is zero at all nodes except the i^{th} node. It is worth noting that such a definition for the interpolation functions ψ_i^e used in (3.2) yields:

$$u^e(x_j) = \sum_{i=1}^{n} u_i^e \psi_i^e(x_j) = u_j^e \tag{3.6}$$

as expected.

Since elements over the whole domain are generally not identical (different shapes), this process of determining all interpolation functions can become fastidious. We will now introduce a general method for defining interpolation functions which allows for arbitrary element shapes (subject to a given topology). To this end, we discuss the concept of element *natural coordinates*.

3.2.3 Natural coordinates

The natural coordinate system allows us to map every element into a typical and simpler element. As an example, an 8-node hexahedron element is shown in Fig. 3.3 as it appears in the global (x_1, x_2, x_3) and natural (ξ_1, ξ_2, ξ_3) coordinate systems. While in its natural coordinate system the element is a regular aligned cube, in the global coordinate system the element may assume any admissible arbitrary form. Essentially, admissible means that the element must not be too distorted, and must certainly not fold back on itself. Another example is given Fig. 3.4 with a tetrahedral element. The components of each node has a simple expression in natural coordinates, usually $0, 1$ or -1 and the centre of the coordinate system is taken at one of the nodes or at the centre of the element.

The use of the natural coordinate system has several advantages (Biswas et al., 1976). Not only the interpolation functions can be derivated only once per type of element in the mesh, but their expression is also much simpler, regardless of the actual element shape. Consequently, the element equations and the derived element matrices get to be simplified too. However, we now need to switch between the natural and global coordinate systems before solving the equations on the whole domain. Indeed, it is necessary to account for the diversity of element shapes within the mesh. Solving the element equations expressed into the natural coordinate system would not yield the expected result for the whole domain.

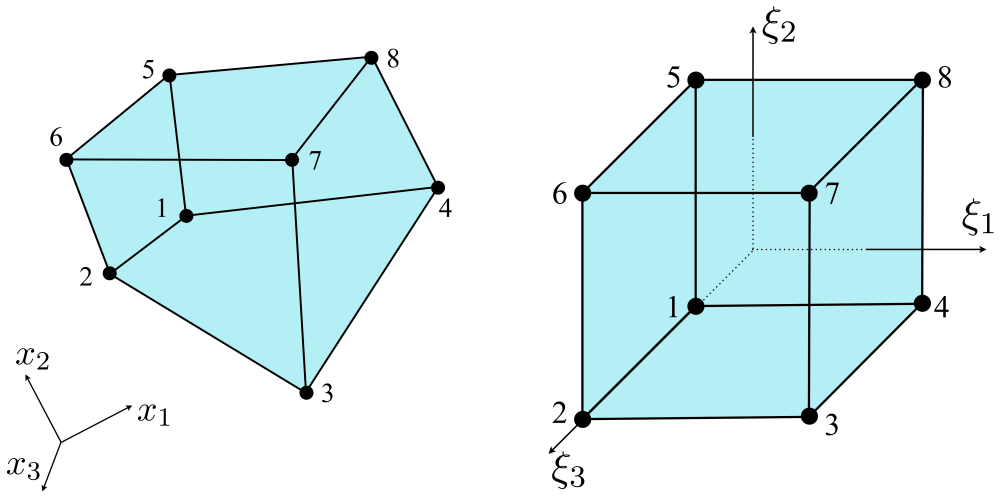


Figure 3.3: Hexahedral element as it appears in the global coordinate system (a) and its natural coordinate system (b).

3.2.4 Geometry interpolation

Let us assume a relation between the global coordinate x and the natural coordinate ξ in the following form:

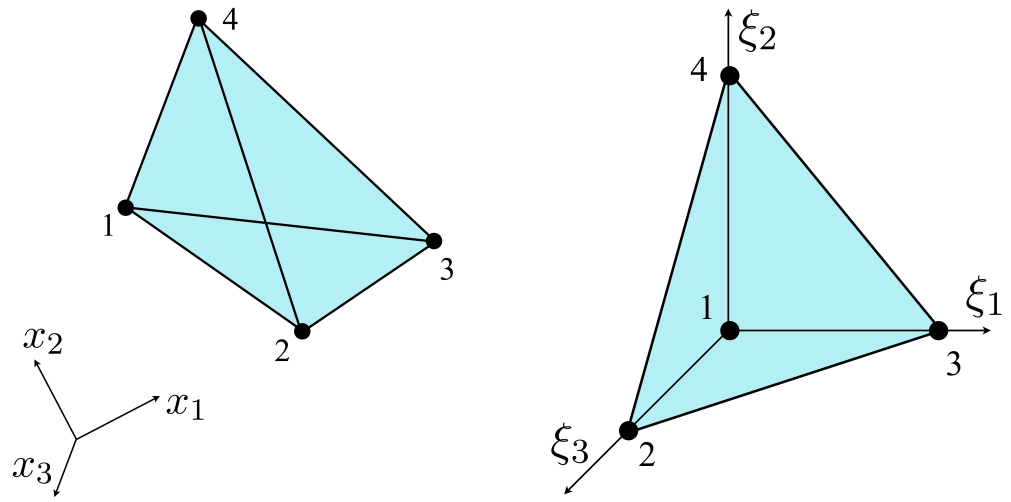
$$x = f(\xi) \tag{3.7}$$

where f is assumed to be a one-to-one correspondence (that is, a bijective function). This function may be seen as a transformation between the natural shape of the element and its arbitrary shape within the actual mesh, it describes a change in geometry. It is natural to think of approximating the geometry in a similar way that we approximated the solution in section 3.2.2. Hence, the transformation defined by (3.7) may also be written as

$$x = \sum_{i=1}^{m} x_i^e \hat{\psi}_i^e(\xi) \tag{3.8}$$

where x_i^e is the global coordinate of the i^{th} node of the element Ω_e, m the number of nodes for the element and $\hat{\psi}_i^e(\xi)$ are the interpolation functions of degree $m-1$. Thus, we have a linear transformation when $m = 2$ and the relation betweeen x and ξ is quadratic when $m = 3$. The interpolation functions $\hat{\psi}_i^e(\xi)$ are called *shape functions* because they are used to express the geometry or shape of the element.

Remark. It is worth noting that (3.8) can be easily extented to three-dimensional problems. Let us consider a Cartesian coordinate system and we denote the position of node i $\mathbf{x}_i = [x_{i1}, x_{i2}, x_{i3}]^T$. The position of a point $\mathbf{x} = [x_1, x_2, x_3]^T$ within the element may then be interpolated by applying (3.8) to each component of the



(a) (b)

Figure 3.4: Tetrahedral element as it appears in the global coordinate system (a) and its natural coordinate system (b).

position:

$$\begin{aligned} x_1 &= \sum_{i=1}^{m} x_{i1}\hat{\psi}_i^e(\xi) \\ x_2 &= \sum_{i=1}^{m} x_{i2}\hat{\psi}_i^e(\xi) \\ x_3 &= \sum_{i=1}^{m} x_{i3}\hat{\psi}_i^e(\xi) \end{aligned} \tag{3.9}$$

If we construct the vector $\hat{\mathbf{x}}^e$ of all nodal positions $\mathbf{x}_i$ of the element, we can define a shape function matrix $\mathbf{H}$ such as:

$$\mathbf{x} = \mathbf{H}\hat{\mathbf{x}}^e \tag{3.10}$$

where $\mathbf{H}$ is a $3 \times m$ matrix built from submatrices $\mathbf{H}_i$

$$\mathbf{H}_i = \begin{bmatrix} \hat{\psi}_i^e & 0 & 0 \\ 0 & \hat{\psi}_i^e & 0 \\ 0 & 0 & \hat{\psi}_i^e \end{bmatrix} \quad i = 1, 2, \ldots, m. \tag{3.11}$$

3.2.5 A particular case: isoparametric elements

Most of the time, we choose to interpolate the solution and the element geometry with the same interpolation functions. In this case, the elements are said to be *isoparametric*. Under this configuration, natural coordinates appear as parameters

that define the shape functions. Note that the shape functions describe the geometry and the solution with the same degree of interpolation. Because isoparametric elements are very common in practice, the use of the phrase *shape functions* is often extented to denote the interpolation functions employed to approximate the solution.

3.3 Derivation of element equations

Let us sum up where we are into the finite element analysis process so far. The whole domain has been divided into sub-domains, smaller, that we call elements, for which the solution to be sought will be simpler to find. We know how to approximate the solution within each of these elements using a linear combination of shape functions. And we even have a method to simplify the expression of these shape functions using natural coordinates.

3.3.1 Strong and weak forms

The next step is therefore to derive the element equations themselves. Obviously these equations depend on the type of problem that we seek to solve. From now on, we will only consider the context of continuum mechanics since our overall goal is the modelling of organ deformation. However, a similar approach can be used in the other fields of physics. In continuum mechanics, applying Newton's second law yields a partial differential system of equations. Such equations are called *strong forms*. The strong form, in contrast to a *weak form*, requires strong continuity on the dependent field variables, such as displacements (Liu and Quek, 2003). Indeed, the functions defining these field variables have to be differentiable up to the order of the partial differential equations that exist in the strong from. While the finite element method can be used to find an approximated solution for a strong form, the method usually works well only with regular geometry and boundary conditions.

Consequently, a weak form is often created using energy principles or weighted residual methods. The weak form is frequently an integral form and requires a weaker continuity on the field variables. Hence, a formulation based on a weak form usually produces a set of discretised system equations that give much more accurate results, especially for problems of complex geometry. For this reason, the weak form is usually preferred over the strong form for obtaining an approximate solution of a practical engineering problem .

3.3.2 Time dependence

In all the equations of the finite element method that we derived so far, we have not taken time into account. For some problem, this is not an issue, the motion is sufficiently slow to assume that the dynamic effects (such as damping effect) are neglectable. However, this assumption is not always valid, particulary in our case where we want to model organ deformation. The deformation obviously depends on

time. Finite element models of time-dependent problems can be developed in two alternatives ways (Reddy, 1993):

(a) coupled formulation in which the time t is an additional coordinate along with the spatial coordinate x

(b) decoupled formulation where time and spatial variations are assumed to be separable.

Thus, the approximation of the solution u takes one of these two forms:

$$u(x,t) \approx u^e(x,t) = \sum_{i=1}^{n} \hat{u}_i^e \hat{\psi}_i^e(x,t) \quad \text{(coupled formulation)} \tag{3.12}$$

$$u(x,t) \approx u^e(x,t) = \sum_{i=1}^{n} u_i^e(t) \psi_i^e(x) \quad \text{(decoupled formulation)} \tag{3.13}$$

where $\hat{\psi}_i^e(x,t)$ are time-space interpolation functions, $\hat{u}_i^e$ are the nodal values that are independent of x and t, $\psi_i^e(x)$ are the usual interpolation functions in spatial coordinate x only and the nodal values $u_i^e(t)$ are functions of time t only. Coupled finite element formulations are not common and they have not been adequately studied. Hence, we consider the decoupled formulation only. Of course, the assumption that the time and spatial variations are separable is not valid in general. However, with sufficiently small time steps, it is possible to obtain accurate solutions nevertheless.

3.3.3 Dynamic system of equations

The space-time decoupled finite element formulation of time-dependent problems involves two steps:

1. **Spatial approximation.** The solution u is first approximated by expressions of the form (3.13) while keeping the time-dependent term during the finite element model derivation. This step leads to a set of ordinary differential equations in time.

2. **Temporal approximation.** The system of ordinary differential equations is then approximated in time where different schemes may be used to assess the time derivatives. This step yields a set of algebraic equations for u_i^e at discrete time $t_{k+1} = (k+1)\Delta t$ where Δt is the time increment and $k > 0$ an integer.

At the end of this two-step approximation, we have a continuous spatial solution at discrete intervals in time:

$$u(x,t_k) \approx u^e(x,t_k) = \sum_{i=1}^{n} u_i^e(t_k) \psi_i^e(x) \quad \text{with } k = 0,1,\ldots \tag{3.14}$$

The construction of the weak form using either energy principles or weighted residual methods will not be detailed here. The reader may easily find it in dozens

of textbooks. By substituting (3.14) in the weak form, we obtain the finite element equations of equilibrium in matrix form:

$$\mathbf{M}^e\ddot{\mathbf{U}}^e + \mathbf{D}^e\dot{\mathbf{U}}^e + \mathbf{K}^e(\mathbf{U}^e)\cdot\mathbf{U}^e = \mathbf{R}^e, \tag{3.15}$$

where $\mathbf{M}^e$ is a constant mass matrix, $\mathbf{D}^e$ is a constant damping matrix, $\mathbf{K}^e(\mathbf{U}^e)$ is the stiffness matrix, which is a function of nodal displacements $\mathbf{U}^e$, and $\mathbf{R}^e$ are externally applied loads. All matrices are expressed at the level of individual elements (which is the reason for the superscript ^{e}).

3.3.4 Static system of equations

Under some assumption, (3.15) may be simplified. For instance, let us consider the problem of finding the maximum load that a desk can sustain. It is fair to say that strain and stress within the structure does not fluctuate over time: the system is in an equilibrium state. At all times, the sum of all forces applied on the system is equal to zero. Therefore, we can assume that the equations describing the structure do not depend on time: the problem is said to be *static*. Static analyses assume that loading of a body occurs slowly enough that inertial and damping terms may be neglected. In this case, the static system of equations can be easily obtained by merely dropping out the dynamics terms in (3.15). Indeed, the static case may just be seen as a special case of the dynamic equations and the static equations are:

$$\mathbf{K}^e(\mathbf{U}^e)\cdot\mathbf{U}^e = \mathbf{R}^e. \tag{3.16}$$

3.3.5 A few words on the matrices involved

Mass matrix

The mass matrix $\mathbf{M}$ for the whole domain may be computed by summing the mass contributions from every elements:

$$\mathbf{M} = \sum_e \mathbf{M}^e, \tag{3.17}$$

where $\mathbf{M}^e$ is the mass of element Ω_e given by:

$$\mathbf{M}^e = \int_{V^e} \rho^e \mathbf{H}^T\mathbf{H}\, dV \tag{3.18}$$

where ρ the mass density, V^e the volume of the element and $\mathbf{H}$ is the shape function matrix defined by (3.10). This integral may be evaluated using a procedure of numerical integration and generally results in a band mass matrix (non-zero entries are confined to a diagonal band, comprising the main diagonal and more diagonals on either side). This method for computing the mass matrix is called *variational mass lumping*.

However, it is very common in finite element analysis to use a diagonal mass matrix. The most common way of achieving this is to define the diagonal entries

with the sum of the corresponding rows and then set all other values to 0. The physical interpretation of this is lumping all of the mass in the volume surrounding a node at the node itself. A diagonal mass matrix may offer significant computational and storage advantages. First, it is easily inverted, since the inverse of a diagonal matrix is also diagonal. And a diagonal mass matrix can also be stored simply as a vector. Therefore, such a matrix may be built by directly computing the mass of each element and assigning an equal proportion of this to each of the element's nodes. This method is named *direct mass lumping*. The mass m^e of element Ω_e is given by:

$$m^e = \rho^e \int_{V^e} dV, \tag{3.19}$$

which is integrated over the element using a numerical method. Each diagonal component of the mass matrix then receives a contribution of m^e/m.

Damping matrix

In a similar way to the mass matrix, a damping matrix may be defined by summing the damping contributions from every elements:

$$\mathbf{D} = \sum_e \mathbf{D}^e, \tag{3.20}$$

where $\mathbf{D}^e$ is the damping of element Ω_e given by:

$$\mathbf{D}^e = \int_{V^e} \kappa^e \mathbf{H}^T \mathbf{H}\, dV \tag{3.21}$$

where κ^e is a damping parameter. In practice, the components of damping matrices are difficult to determine experimentally. Therefore, it is common to employ the *Rayleigh damping*. The Rayleigh damping assumes that the damping may be written in the form of:

$$\mathbf{D}^e = \alpha \mathbf{M}^e + \beta \mathbf{K}^e, \tag{3.22}$$

where α and β are coefficients defining mass and stiffness proportional damping components. However, for systems with large degrees of freedom, it is difficult to guess meaningful values of α and β. Chowdhury and Dasgupta (2003) have proposed a method to calculate these coefficients. In practice, further assumptions may only lead to include the mass proportional component ($\beta = 0$), which yields a diagonal damping matrix if a diagonal mass matrix is used.

Stiffness matrix

The expression of the stiffness matrix depends on the chosen strain tensor as well as the appropriate constitutive model for the material. Its general form cannot be explicited. However, it can be demonstrated that $\mathbf{K}^e$ may be written as

$$\mathbf{K}^e = \int_{V^e} \mathbf{B}^T \mathbf{E}\, \mathbf{B}\, dV, \tag{3.23}$$

Elements	1	2
1	1	2
2	2	3
3	3	4

Table 3.1: Element connectivity information

where $\mathbf{E}$ is a material matrix, which only depends on the constitutive model chosen, and $\mathbf{B}$ is the *strain-displacement matrix* defined as the derivatives of the shape functions (see Liu and Quek (2003); Belytschko et al. (2000) for details).

3.4 Assembly of element equations

We have now derived the equations for each element. However, since the element is physically connected to its neighbours, the resulting algebraic equations will contain more unknowns than the number of algebraic equations. In other words, elements share vertices. The deformation of an element induces the deformation of its neighbours. Thus, it becomes necessary to put the elements together to eliminate the extra unknowns, to model this interaction between elements. The assembly of the global stiffness matrix should be carried out as soon as the element matrices are computed, rather than waiting until the element coefficients of all elements are computed. This would require storage of the coefficients for each element. Instead, we can perform the assembly while we calculate the element matrices.

The assembly operation is actually quite simple. Let us consider an example with stiffness matrices for an one-dimensional problem. We assume that we have three two-node elements. And we suppose the element connectivity information given by Table 3.1, which provides the global node number for each local node number within an element. We also assume the following stiffness matrices for the three elements:

$$\mathbf{K}^1 = \begin{bmatrix} -1 & 2 \\ 3 & 1 \end{bmatrix}, \quad \mathbf{K}^2 = \begin{bmatrix} 5 & 2 \\ -2 & 1 \end{bmatrix} \text{ and } \quad \mathbf{K}^3 = \begin{bmatrix} 3 & 1 \\ 2 & -1 \end{bmatrix} \tag{3.24}$$

The assembled matrix is a square matrix of dimension number of nodes $\times$ number of degrees of freedom. In our case, the global matrix is therefore a 4×4 and is assembled in the following manner:

$$\mathbf{K} = \begin{bmatrix} -1 & 2 & 0 & 0 \\ 3 & 1+5 & 2 & 0 \\ 0 & -2 & 1+3 & 1 \\ 0 & 0 & 2 & -1 \end{bmatrix} \tag{3.25}$$

In other words, the global matrix is merely a rewritting on the whole domain of the element stiffness matrices, which are expressed with respect to local node numbers, in terms of global node numbers. In addition to constructing the global mass and

damping matrices (as shown in section 3.3.5), this process leads to the following global system of equations:

$$\mathbf{M}\ddot{\mathbf{U}} + \mathbf{D}\dot{\mathbf{U}} + \mathbf{K}(\mathbf{U}) \cdot \mathbf{U} = \mathbf{R}. \tag{3.26}$$

The same set of equations was established at the element level in section 3.3.3 but we may note that the same system is also valid for the whole domain.

3.5 Solution of global problem

The complete and global system of equations has now been established. In this section, we will examine the techniques commonly used for solving these equations. In the general case of a dynamic system of equations, the equations are not yet algebraic and still depends on derivatives of the unknown displacements with respect to time. Therefore, an additional step needs to be performed before using a solver by integrating the equations over time. As we will see shortly, the different methods of time integration are classified into two main schemes: *explicit* or *implicit* integration. We begin by introducing the simplest method, the Euler method to explain the concept of an explicit method. Then the central difference method will be detailled. We then describe a few implicit methods, allowing us to highlight the relative advantages of the two types of methods. Finally, the static case will be solved by emphasizing the similarities with implicit systems through the critical step of linearisation of the governing equations (Belytschko et al., 2000).

3.5.1 Explicit time integration

The Euler method

We want to approximate the solution of the following differential equation:

$$\dot{\mathbf{U}}^t = f(t, \mathbf{U}^t). \tag{3.27}$$

Note that we restrict ourselves to first-order differential equations (meaning that only the first derivative of $\mathbf{U}^t$ appears in the equation, and no higher derivatives). However, a higher-order equation can easily be converted to a system of first-order equations by introducing extra variables. For example, the second-order equation $\ddot{\mathbf{U}}^t = -\mathbf{U}^t$ can be rewritten as two first-order equations: $\dot{\mathbf{U}}^t = \mathbf{V}^t$ and $\dot{\mathbf{V}}^t = -\mathbf{U}^t$. By solving equations like (3.27), we will therefore be able to integrate the global system of equations over time.

We start by replacing the derivative $\dot{\mathbf{U}}^t$ by the finite difference approximation:

$$\dot{\mathbf{U}}^t \approx \frac{\mathbf{U}^{t+\Delta t} - \mathbf{U}^t}{\Delta t}, \tag{3.28}$$

which gives:

$$\mathbf{U}^{t+\Delta t} \approx \mathbf{U}^t + \Delta t\, \dot{\mathbf{U}}^t \tag{3.29}$$

And substituting the last relation in (3.27) yields:

$$\mathbf{U}^{t+\Delta t} \approx \mathbf{U}^t + \Delta t\, f(t, \mathbf{U}^t) \tag{3.30}$$

This formula is usually applied in a recursive scheme. We choose a time step Δt and we construct the sequence of time $t^n = n\Delta t$. Using (3.30), we can approximate the exact solution $\mathbf{U}^{t^n}$ at each time step by $\mathbf{U}^n$:

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t\, f(t^n, \mathbf{U}^n). \tag{3.31}$$

This method is called *Euler method* and is said to be explicit because the new value $\mathbf{U}^{n+1}$ is defined in terms of variables from the previous time step $\mathbf{U}^n$.

The central difference method

The central difference method is one of the most popular explicit methods in computational mechanics. This method is based on central difference formulas for the velocity and acceleration. Let us the time t of the simulation be subdivided into time steps Δt. Let us denote by $\mathbf{U}^t$, $\dot{\mathbf{U}}^t$ and $\ddot{\mathbf{U}}^t$ the matrices of nodal displacements, velocities and accelerations, respectively, at time t. The central difference method makes use of the following finite differences approximations for acceleration and velocity:

$$\ddot{\mathbf{U}}^t = \frac{1}{(\Delta t)^2}\left(\mathbf{U}^{t-\Delta t} - 2\mathbf{U}^t + \mathbf{U}^{t+\Delta t}\right) \tag{3.32}$$

$$\dot{\mathbf{U}}^t = \frac{1}{2\Delta t}\left(\mathbf{U}^{t+\Delta t} - \mathbf{U}^{t-\Delta t}\right). \tag{3.33}$$

One of the key advantage of the method is that the stifness term in (3.26) may be computed from

$$\mathbf{K}(\mathbf{U}) \cdot \mathbf{U} = \mathbf{F}(\mathbf{U}) = \sum_e \tilde{\mathbf{F}}^e, \tag{3.34}$$

where $\tilde{\mathbf{F}}^e$ are the global nodal force contributions due to stresses in element Ω_e. $\tilde{\mathbf{F}}$ may be computed using strain-displacement equations, the constitutive equation and the element connectivity. Therefore, let us assume that all nodal forces $\mathbf{F}^t$ at time t are known.

By subtituting (3.32) and (3.33) into (3.26) we obtain:

$$\left(\frac{\mathbf{M}}{(\Delta t)^2} + \frac{\mathbf{D}}{2\Delta t}\right)\mathbf{U}^{t+\Delta t} = \mathbf{R}^t - \mathbf{F}^t + \frac{2\mathbf{M}}{(\Delta t)^2}\mathbf{U}^t + \left(\frac{\mathbf{D}}{2\Delta t} - \frac{\mathbf{M}}{(\Delta t)^2}\right)\mathbf{U}^{t-\Delta t}. \tag{3.35}$$

Both the right-hand side of the equation and the factor in front $\mathbf{U}^{t+\Delta t}$ may be evaluated, which gives the nodal displacements for the next time step. The entire update can be accomplished without solving any system of equations provided that the mass matrix $\mathbf{M}$ and the damping matrix $\mathbf{D}$ are diagonal. As we have seen in section 3.3.5, these approximations are often used in practice. And this is the characteristic of an explicit method: the time integration of the general system of

equations for finite element model does not require the solution of any equations. This is a key point because it gives explicit methods a tremendous computational advantage. However, explicit methods are only conditionally stable. If the time step exceeds a critical value, the solution may be erroneous. This critical time step depends on the mesh and the material properties. It will decrease with mesh refinement and increasing stiffness of the material.

3.5.2 Implicit time integration

The backward Euler method

The problem that we wish to solve is the same than the one we introduced with the Euler method, which we reproduce here for convenience:

$$\dot{\mathbf{U}}^t = f(t, \mathbf{U}^t). \tag{3.36}$$

This time, instead of using (3.28), we approximate the derivative $\dot{\mathbf{U}}^t$ by:

$$\dot{\mathbf{U}}^t \approx \frac{\mathbf{U}^t - \mathbf{U}^{t-\Delta t}}{\Delta t}. \tag{3.37}$$

The update expression for y^n then becomes:

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t\, f(t^{n+1}, \mathbf{U}^{n+1}). \tag{3.38}$$

The *backward Euler method* is an implicit method, meaning that we have to solve an equation to find $\mathbf{U}^{n+1}$. Of course, it costs time to solve this equation and this cost must be taken into consideration when we select the time integration method to use. The advantage of implicit methods is that they are usually more stable for solving a stiff equation, meaning that larger steps can be used.

The Runge-Kutta method

The accuracy of backward Euler method can be improved by making the solution depends on more values. The 4^{th} order Runge-Kutta method gives the following update equation:

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \frac{1}{6}\Delta t\,(k_1 + 2k_2 + 2k_3 + k_4)\,, \tag{3.39}$$

where :

$$\begin{aligned}
k_1 &= f(t^n, \mathbf{U}^n) \\
k_2 &= f(t^n + \frac{1}{2}\Delta t, \mathbf{U}^n + \frac{1}{2}\Delta t\, k_1) \\
k_3 &= f(t^n + \frac{1}{2}\Delta t, \mathbf{U}^n + \frac{1}{2}\Delta t\, k_2) \\
k_4 &= f(t^n + \Delta t, \mathbf{U}^n + \Delta t\, k_3)
\end{aligned}$$

This method is a fourth-order method because the total accumulated error has order Δt^4. It may be seen as a generalisation of the backward Euler method (Press et al., 1992). Again, we have to solve an equation to find $\mathbf{U}^{n+1}$.

3.5.3 Static solutions

First, we recall the general system of equations in static introduced in section 3.3.4, which is also valid for the whole system:

$$\mathbf{K}(\mathbf{U}) \cdot \mathbf{U} = \mathbf{R}. \tag{3.40}$$

The situation is similar to implicit time integration methods for dynamic systems by the fact that we also have to solve a set of algebraic equations to find the nodal displacements. In the general case, the equations are non-linear and therefore quite difficult to solve. A suitable solver for this kind of system is, for instance, the *Newton-Raphson* method. In essence, this method consists of obtaining linear equations instead. This process is called *linearisation*. One may refer to (Belytschko et al., 2000) for more details.

However, if the problem is linear, the stiffness matrix does not depend on the displacement and the resolution is facilitated. The unknown displacements may be computed by:

$$\mathbf{U} = \mathbf{K}^{-1}\,\mathbf{R}. \tag{3.41}$$

Many techniques are available in the literature to solve this type of system, and a few of them are introduced in the next section.

3.5.4 Solvers

We will now examine the techniques commonly used for solving linear systems of algebraic equations. Note that this is only the most classical approaches and the list is obviously not exhaustive. Two categories of techniques are in opposition: *direct* or *iterative* solvers (Press et al., 2002). In any case, we seek the solution for the unknown solution $\mathbf{x}$ of the following system of linear equations:

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{3.42}$$

The system may already be linear by nature, or been linearised using techniques such as Newton-Raphson methods. At this point in the discussion, we are therefore only interested in solving a linear system of equations. If the matrix $\mathbf{A}$ is squared, which is the case in finite element methods, the problem may be written as follows:

$$\mathbf{x} = \mathbf{A}^{-1}\,\mathbf{b}. \tag{3.43}$$

However, only solvers relevant to the context of finite element methods will be discussed. Indeed the matrix $\mathbf{A}$ takes a specific form in problems solved by finite element methods. The matrix $\mathbf{A}$ is *sparse*. A sparse matrix is merely a matrix populated mainly with zeros. An example is given by Fig. 3.5 for a finite element problem in two dimensions. Because of the peculiar structure of the matrix involved in finite element computations, methods have been specifically designed to take advantage of this structure. Indeed, it is wasteful to use general methods of linear algebra on such problems, because most of the arithmetic operations devoted to solving the set of equations or inverting the matrix involve zero operands. Furthermore, it is wasteful to reserve storage for zero elements.

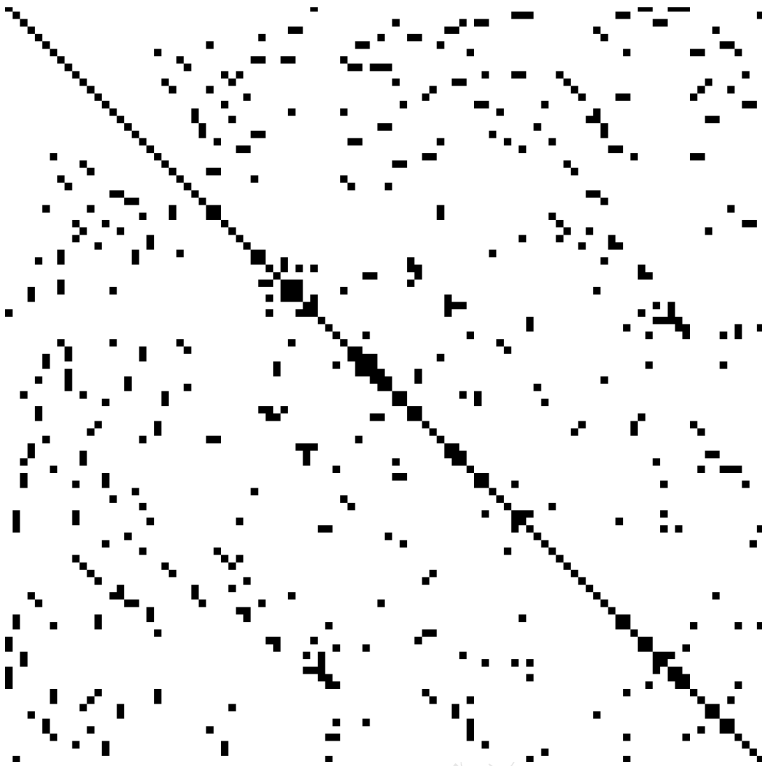


Figure 3.5: Illustration of a sparse matrix for a two-dimensional finite element problem. Each non-zero elements of the matrix is represented in black. Image courtesy of Oleg Alexandrov.

Direct solvers

As their name is suggesting, direct solvers make an attempt into finding a solution directly by inverting the matrix $\mathbf{A}$. They execute in a predictable number of operations. We will now introduce a few direct methods in the context of finite element methods.

LU decomposition. The LU decomposition is a matrix decomposition which writes a matrix as the product of a lower triangular matrix and an upper triangular matrix. In other words, $\mathbf{A}$ may be written as:

$$\mathbf{A} = \mathbf{LU}, \tag{3.44}$$

where $\mathbf{L}$ is lower triangular (has elements only on the diagonal and below) and $\mathbf{U}$ is upper triangular (has elements only on the diagonal and above). As an example,

for the case of a 4×4 matrix $\mathbf{A}$, we have:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix} \tag{3.45}$$

By subtituting (3.44) into (3.42) we obtain:

$$\mathbf{Ax} = (\mathbf{LU})\mathbf{x} = \mathbf{L}(\mathbf{Ux}) = \mathbf{b}. \tag{3.46}$$

Therefore, we can solve (3.42) by first solving the vector $\mathbf{y}$ such that:

$$\mathbf{Ly} = \mathbf{b} \tag{3.47}$$

and then solving

$$\mathbf{Ux} = \mathbf{y}. \tag{3.48}$$

The advantage of breaking up one linear set of equations into two successive ones is that the solution of a triangular set of equations is quite trivial. Indeed, (3.47) can be solved by:

$$\begin{cases} y_1 = \dfrac{b_1}{\alpha_{11}} \\ y_i = \dfrac{1}{\alpha_{ii}} \left[b_i - \displaystyle\sum_{j=1}^{i-1} \alpha_{ij} y_j \right] \quad i = 2, 3, \ldots, N. \end{cases} \tag{3.49}$$

for a $N \times N$ matrix $\mathbf{A}$. We then solve (3.48) in a similar way:

$$\begin{cases} x_N = \dfrac{y_N}{\beta_{NN}} \\ x_i = \dfrac{1}{\beta_{ii}} \left[y_i - \displaystyle\sum_{j=i+1}^{N} \beta_{ij} x_j \right] \quad i = N-1, N-2, \ldots, 1. \end{cases} \tag{3.50}$$

Cholesky decomposition. Cholesky decomposition allows to write a symmetric and definite positive matrix as the product of a lower triangular matrix with strictly positive diagonal entries by the transpose:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T, \tag{3.51}$$

where $\mathbf{L}$ is lower triangular with strictly positive diagonal entries. In general, Cholesky decompositions are not unique. Similarly to LU decomposition, a Cholesky decomposition may be used to break one set of equations into two successive but simpler systems.

Iterative solvers

An iterative method attempts to solve the system of equations by finding successive approximations to the solution starting from an initial guess. In the case of a system of linear equations, the two main classes of iterative methods are the stationary iterative methods, and the more general Krylov subspace methods. Stationary iterative methods solve a linear system with an operator approximating the original one and based on a measurement of the error in the result (called the residual), form an equation of correction for which this process is repeated. While these methods are simple to derive, convergence is only guaranteed for a limited class of matrices. Examples of stationary iterative methods are the Jacobi method and Gauss-Seidel method. Krylov subspace methods work by forming an orthogonal basis of the sequence of successive matrix powers times the initial residual (the Krylov sequence). The approximations to the solution are then formed by minimising the residual over the subspace formed. The typical method in this class is the conjugate gradient method (CG). In practice, for large system of equations (as often encountered in finite element analysis), direct methods would take too much time and an iterative solvers is commonly used.

The Gauss-Seidel method. For this method, convergence is only guaranteed if the matrix is either diagonally dominant, or symmetric and positive-definite. In a similar way than the LU decomposition, we start by decomposing the matrix $\mathbf{A}$ into a lower triangular component $\mathbf{L}_*$ and a strictly upper triangular component $\mathbf{U}$:

$$\mathbf{A} = \mathbf{L}_* + \mathbf{U}. \tag{3.52}$$

The system of linear equations (3.42) may be rewritten as:

$$\mathbf{L}_*\mathbf{x} = \mathbf{b} - \mathbf{U}\mathbf{x} \tag{3.53}$$

The Gauss-Seidel method is an iterative technique that solves the left-hand side of the expression for $\mathbf{x}$ using previous values for $\mathbf{x}$ on the right-hand side. That is:

$$\mathbf{x}^{n+1} = \mathbf{L}_*^{-1}(\mathbf{b} - \mathbf{U}\mathbf{x}^n). \tag{3.54}$$

Similarly to the LU decomposition method, we take advantage of the triangular form of $\mathbf{L}_*$. The components of $\mathbf{x}^{n+1}$ can be computed sequentially in the following manner:

$$x_i^{n+1} = \frac{1}{a_{ii}} \left[b_i - \sum_{j>i} a_{ij} x_j^n - \sum_{j<i} a_{ij} x_j^{n+1} \right] \quad i = 1, 2, \ldots, N. \tag{3.55}$$

The procedure is usually continued until the changes made by an interation are below a given threshold.

The conjugate gradient method. The simplest class of conjugate gradient method is based on the idea of minimising the quadratic function:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A}\,\mathbf{x} - b^T \mathbf{x} + c. \tag{3.56}$$

Indeed, if the matrix $\mathbf{A}$ is symmetric and positive-definite, $f(\mathbf{x})$ is minimised by the solution to $\mathbf{Ax} = \mathbf{b}$ (Shewchuk, 1994). The minimisation is carried out by generating a succession of search directions $\mathbf{p}_k$ and improved minimisers $\mathbf{x}_k$. At each stage a quantity α_k is found that minimises $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$, and $\mathbf{x}_{k+1}$ is set equal to the new point $\mathbf{x}_k + \alpha_k \mathbf{p}_k$. The $\mathbf{p}_k$ and $\mathbf{x}_k$ are built up in such a way that $\mathbf{x}_{k+1}$ is also the minimiser of f over the whole vector space of directions already taken, that is the space formed by the basis of vector $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k$. After N iterations, you arrive at the minimiser over the entire vector space, that is, the solution to our linear system of equations.

Later, the *biconjugate gradient method* was introduced as a generalisation of the ordinary conjugate gradient to solve systems where the matrix $\mathbf{A}$ was not necessarily symmetric and positive-definite (see Belytschko et al. (2000) for details).

Preconditioners Unfortunately, it is common for the matrix $\mathbf{A}$ not to be well conditioned. Consequently, the convergence of the chosen iterative method is often slow. In order to achieve an accelerated convergence, we may use a *preconditioner*. Although the formation and the study of preconditioners are out of the scope of this PhD thesis, preconditioners are a key element for solving systems of equations. To be effective, a preconditioner $\mathbf{P}$ should be a good approximation of $\mathbf{A}$ and the following preconditioned system must be easier to solve than the original system (3.42):

$$\mathbf{P}^{-1}\mathbf{Ax} = \mathbf{P}^{-1}\mathbf{b}. \tag{3.57}$$

Two criteria should also be considered when forming a preconditioner: (1) the ease with which it may be evaluated and (2) its memory requirements. The easiest preconditioner would be the identity since $\mathbf{P}^{-1} = \mathbf{I}$ in this case. Of course, such a conditioner leads to the resolution of the original system and is therefore totally useless. In constrast, the choice of $\mathbf{P} = \mathbf{A}$ leads to the solution of the system (3.42) in one single iteration. In practice, an optimal preconditioner is to be found between these two extremes to minimise the number of iterations required by the solver while keeping the evaluation of the preconditioner simple. Finding a good preconditioner for all types of problems is difficult. A preconditioner is only useful if the overall time spent on both the formation of $\mathbf{P}$ and the resolution of (3.57) is less than the time spent at solving (3.42) alone.

One of the simplest preconditioner is the *Jacobi preconditioner* where the preconditioner is chosen to be the diagonal of the matrix $\mathbf{A}$. This choice is efficient in cases where $\mathbf{A}$ is diagonally dominant. Sparse approximations of LU or Cholesky decompositions may also be used as preconditioners (see sections 3.5.4 and 3.5.4 for more details).

Part II

Solid organs modelling

Chapter 4

Modelling the deformation of solid objects in real-time

Modelling the deformation of solid organs is a crucial problem in medical simulation. If the finite element method is classicaly used in mechanical engineering for its ability to solve the equations of continuum mechanics fairly accurately, its development in medical simulation was more progressive. The main reason for this is that the finite element method is computationally demanding and not easily compatible with time constraints required in interactive simulation. Therefore, other methods were developed to allow faster computations, at the cost of larger approximations of course. Over time, along with the increase in computational power, methods became more elaborate and less approximative. These different approaches may be grouped into three main categories: (1) geometrically based techniques, (2) approaches physically motivated and (3) methods actually based on the equations of continuum mechanics. In this chapter, we will review the most classic methods applied to modelling of solid objects with a particular emphasis on medical simulation.

4.1 Introduction: the problematic

The challenges faced by the field of medical simulation were reviewed in details in the introductory chapter (see chapter 1). Hence, we know that at the very least, the deformation of solid organs must appear physically realist in medical simulators. In addition, this objective has to be achieved within heavy time constraints. Indeed, although each step of a simulation alone (deformation of organs, collision detection, contacts processing or visualisation) is computationally demanding, the whole simulation process must be computed in real-time or close to real-time. Of particular interest to us is the accurate modelling of the deformation of anatomical structures. The best mathematical model for predicting the deformation of a continuous structure is provided by continuum mechanics. If the finite element method is classicaly used in mechanical engineering for its ability to solve the equations of continuum mechanics fairly accurately, its development in medical simulation was more progressive. The main reason for this is that the finite element method is computationally very demanding and not easily compatible with the time constraints required in interactive simulation. Very often simplifications have to be made. Since medical simulation may find applications in various areas (such as medical training, patient-specific planning or per-operative guidance), the requirements in terms of accuracy as well as time constraints may be different. Consequently, a range of models has been developed over time to suit all usages.

For instance, the correctness of deformation is crucial for patient-specific planning and per-operative guidance. Let us give an example where simulation can lead the surgeon towards its goal during surgery: the removal of a brain tumor. Prior surgery, images of the patient's brain were acquired and the tumor was localised. To access the brain during surgery, the surgeon must first drill a hole into the patient's skull. However, drilling a hole releases intra-cranial pressure and this change in internal pressure deforms the brain. This phenomenon is called *brain shift*. The overall deformation is quite complex and the displacements depend on local elasticity of the brain, the size and the location of the tumor. Removing the tumor relies heavily on knowing the spatial relation between the patient's brain and the images acquired prior to surgery. And because the brain deforms during surgery, the location and the shape of the tumor will no longer match those of the images acquired. Therefore, a large brain shift, if not corrected, will result in inaccuracies in the surgical procedure and has the potential to cause damage to normal tissue. A solution is to accurately model the mechanical response of the patient's brain to predict the actual deformation and hence keep the knowledge of the tumor's location. If the accuracy is obviously of prime importance in this situation, computation of the deformation does not necessarily need to be carried out in real-time, a reasonable delay is acceptable (a couple of minutes).

Conversely, when medical simulation is applied to training, such a precision in terms of deformation is not always required. A visually realistic deformation is often sufficient for an inexperienced physician to learn the appropriate gestures required for a given medical procedure. However, in some cases, the feel of touch can be very

important to master the procedure. For instance, during a colonoscopy procedure, force feedbacks give precious information about what type of loop is forming during the colonoscope's insertion and the physician can then react accordingly to prevent loop formations. Consequently, a good colonoscopy simulator is required to provide accurate force feedbacks. If the contact forces were not reproduced to the operator, he would never be able to learn how to detect loop forming. Obviously, the computation of a realistic force to be returned demands an appropriate mechanical modelling of all structures. In constrast of per-operative guidance, medical training demands real-time computations of the deformations.

If the reasons differ, the need for modelling the mechanical response of organs with precision remains. However, strong time constraints often limit the complexity of the modelling and affects the overall accuracy. Because of this trade-off, various approaches were proposed over the years to fit into real-time constraints and may be grouped into three main categories: (1) geometrically based techniques, (2) approaches physically motivated (usually relying on Newton's second law) and (3) methods actually based on the equations of continuum mechanics.

4.2 Techniques based on geometry

4.2.1 Free-form deformation

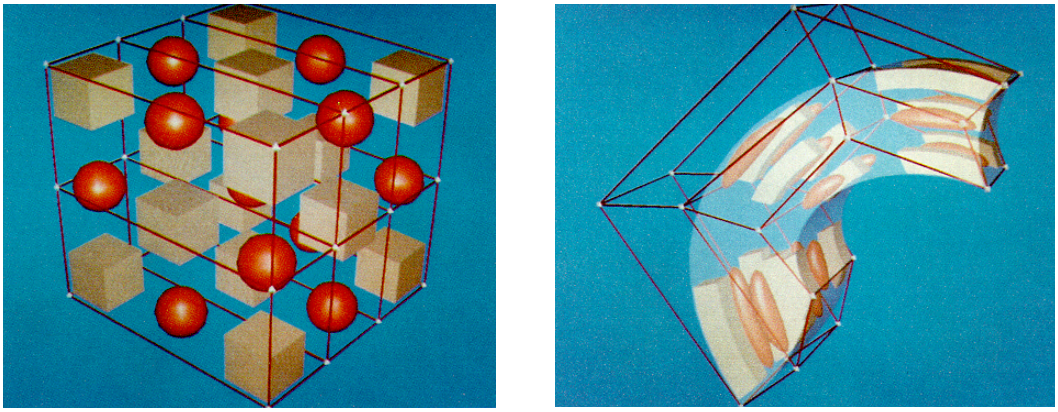
The first needs to deforming solid objects came from for the field of solid modelling. The goal was to allow the designer to shape an object into the form he wishes in a similar way that clay is manipulated by a sculptor's hands. Sederberg and Parry (1986) introduced a technique called *free-form deformation* based on the paper of Barr (1984). It involves a mapping from $\mathbb{R}^3$ to $\mathbb{R}^3$ through a tensor product trivariate Bernstein polynomial. The deformed shape of the object is interpolated using control points as in the case of Bézier curves and surface patches (see Fig. 4.1).

The capability of locally applying deformations makes the technique strongly analogous to sculpting with clay, which was the sought target. Although the authors do not give details on computational efficiency, their work constitutes the beginnings of deformable solid objects.

Coquillard later developed the idea of Extented free-form deformation (1990) and Animated free-form technique (1991). The idea of free-form deformation with lattices of arbitrary topology was introduced by MacCracken and Joy (1996). Finally, Schein and Elber (2004) presented a technique to facilitate the incorporation of discontinuities in a model while deforming it.

4.2.2 Shape matching

One field particulary interested in modelling deformable objects in a very efficient way is the game industry. Of course, for games we are more attracted by computational efficiency and extreme stability features than accuracy to physics laws. Most of the time, we can tolerate a degradation of realism as long as the result



(a) (b)

Figure 4.1: Let us imagine that the objects that we want to deform (spheres and cubes) are placed into a transparent parallelepiped (a). Control points are indicated by small white diamonds. The free-form deformation technique offers to compute the deformation of the objects from the positions of the control points (b). Image courtesy of Sederberg and Parry (1986)

looks realistic. Müller et al. (2005) developed a technique with this idea in mind called *shape matching*. We start with a set of particules with masses m_i in an initial configuration where we denote by $\mathbf{x}_i^0$ the initial positions of the particules. No connectivity information is required. The particles are simulated as a simple particle system without particle-particle interactions, but including response to collisions with the environment and including external forces such as gravity. The positions in the deformed configuration are noted $\mathbf{x}_i$. The method consists of finding a set of points $\mathbf{g}_i$ which minimises the difference between the two sets $\mathbf{x}_i^0$ and $\mathbf{x}_i$. The first step is to find the rotation matrix $\mathbf{R}$ and the translation vectors $\mathbf{t}$ and $\mathbf{t}_0$ such that

$$\sum_i m_i(\mathbf{R}(\mathbf{x}_i^0 - \mathbf{t}_0) + \mathbf{t} - \mathbf{x}_i)^2 \tag{4.1}$$

is minimal. The optimal translation vectors $\mathbf{t}$ and $\mathbf{t}_0$ turn out to be the centre of mass of the initial shape and actual shape, that we will note $\mathbf{x}_c^0$ and $\mathbf{x}_c$, respectively. The optimal rotation matrix is found by first finding the optimal linear transformation $\mathbf{A}$. The optimal rotation is eventually obtained through the rotational part of $\mathbf{A}$ after a polar decomposition. Finally, the goal positions $\mathbf{g}_i$ can be computed as follows:

$$\mathbf{g}_i = \mathbf{R}(\mathbf{x}_i^0 - \mathbf{x}_c^0) + \mathbf{x}_c. \tag{4.2}$$

Once the goal positions are known, they are used to integrate the positions of the particules:

$$\begin{cases} \mathbf{v}_i(t+h) = \mathbf{v}_i(t) + \alpha \dfrac{\mathbf{g}_i(t) - \mathbf{x}_i(t)}{h} + h\dfrac{f_{\text{ext}}(t)}{m_i} \\ \\ \mathbf{x}_i(t+h) = \mathbf{x}_i(t) + h\mathbf{v}_i(t+h) \end{cases} \tag{4.3}$$

where α is a parameter between 0 and 1 which simulates stiffness. Because the method in this basic form is not suitable for objects undergoing large deformations, it is then extended to linear and quadratic deformations by replacing $\mathbf{R}$ in (4.2) with combinations of the form $\beta\mathbf{A} + (1-\beta)\mathbf{R}$ where β is an additional parameter. Linear transformation can represent shear and stretch while quadratic transformation add twist and bending. Of course, this technique is not physically realistic. But this is beyond the point. This method allows the authors to simulate hundreds of deformable objects at an interactive rate with an unconditional stability (see Fig. 4.2). Another illustration of this stability is shown Fig. 4.3.

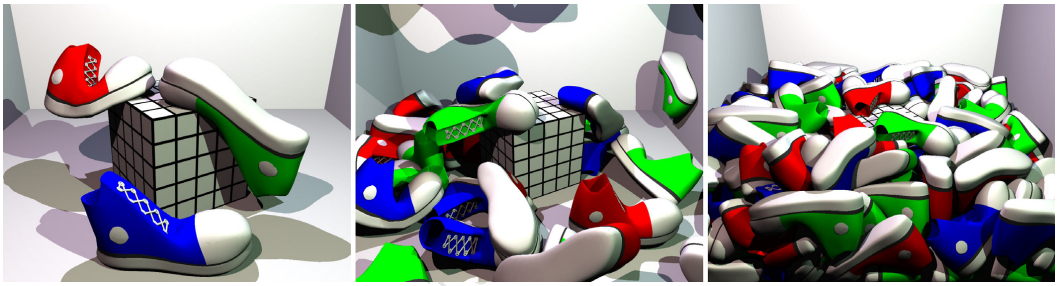


Figure 4.2: Shape matching allows to simulate hundreds of deformable objects in real-time. Image courtesy of Müller et al. (2005).

Later, Rivers and James (2007) extented the technique to regular lattices with embedded geometry. From a surface mesh to be deformed, they start by voxelising the model to construct a lattice of cubic cells containing the mesh. The embedded mesh is then deformed using trilinear interpolation of lattice vertex positions. A particule is placed at each vertex of the lattice and associated with a shape matching region. Their technique of *fast lattice shape matching* allows them to compute per-region transformations in an efficient manner (see Fig. 4.4).

One limitation of fast lattice shape matching is that small features of a geometry yield an explosion of the runtime cost. Indeed, a small surface feature may require fine sampling in order to be deformed independently from non-adjacent material, but this fine sampling must be applied to the whole object. Therefore, Steinemann et al. (2008) further improved the technique by allowing dynamic adaptive selection of levels of detail through the use of an octree-based hierarchical sampling. Their technique also handles efficient topological changes.

Although realism is not a crucial feature for game-like environments, other models based on laws of physics were created for other applications.

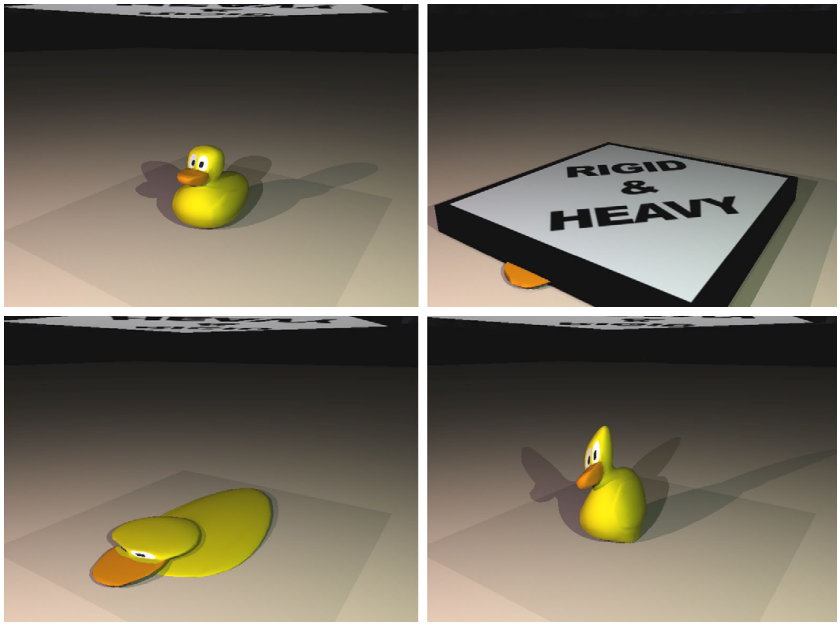


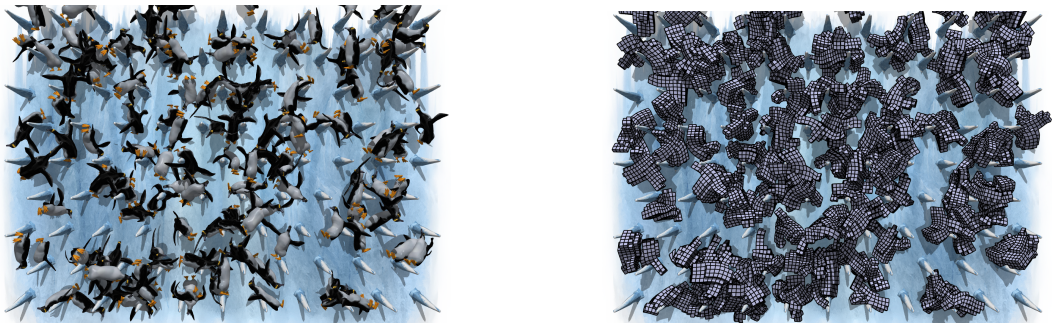
Figure 4.3: A duck model is squeezed to demonstrate the stability of the shape matching technique. Although the flat shape of the duck is not physically plausible, this example exposes the impressive ability of the approach to recover from highly deformed or inverted configurations. Image courtesy of Müller et al. (2005)

4.3 Techniques relying on physics

4.3.1 3D Chainmail algorithm

The *3D Chainmail* was introduced by Gibson (1997) as a fast algorithm for deforming solid objects. Rather than carrying out complex computations on a low resolution geometry (as opposed to FEM techniques), the main idea is to take advantage of the original data resolution (of a CT scan for instance) but perform simple deformation calculation for each element. In this algorithm, each volume element is linked to its six nearest neighbours. If a link between two nodes is stretched to its limit, displacements are transferred to its neighbouring links. It works in a very similar way to a chain and is illustrated by Fig. 4.5. The chain mail algorithm is then followed by an elastic relaxation, which relaxes the shape of the deformed object approximated by the chain mail algorithm. This step consists of minimising an elastic energy measure within the object to insure that the calculated deformation is a minimum energy configuration.

This method was extented by Schill et al. (1998) to integrate inhomogeneous and anisotropic behaviour. This extension of the chain mail algorithm allows the authors to simulate the vitrous humor in the eye during a vitrectomy procedure. However, the main drawback of this algorithm is that it does not model volume preservation, an important characteristic of many tissues in the human body.



(a) (b)

Figure 4.4: (a) 150 cartoon penguins deforming dynamically using fast lattice shape matching (b). Deformed lattices consisting of 150 particles per penguin (22 500 particles). The penguins can be deformed at 25 frames per second on a Pentium4 3.4 GHz. Image courtesy of Rivers and James (2007).

4.3.2 Modal analysis

Modal analysis was first introduced to the graphics community by Pentland and Williams (1989) as a fast method for approximating deformation. Modal analysis is the process of taking the non-linear description of a system, finding a good linear approximation, and then finding a coordinate system that diagonalises the linear approximation. This process transforms a complicated system of non-linear equations into a simple set of decoupled linear equations that may be individually solved analytically (Hauser et al., 2003). Moreover, because each decoupled equation may be solved analytically, stability concerns due to the choice of a time integration procedure are eliminated. The linearised system of equations may be written as:

$$\mathbf{M}\ddot{\mathbf{U}} + \mathbf{D}\dot{\mathbf{U}} + \mathbf{K}\mathbf{U} = \mathbf{R}. \tag{4.4}$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{D}$ is the damping matrix, $\mathbf{K}$ is the stiffness matrix and $\mathbf{R}$ are the externally applied loads. Modal decomposition refers to the process of diagonalising equation (4.4). By using Rayleigh damping (see discussion on damping in section 3.3.5), (4.4) may be rewritten as:

$$\mathbf{K}(\mathbf{U} + \alpha_1\dot{\mathbf{U}}) + \mathbf{M}(\alpha_2\dot{\mathbf{U}} + \ddot{\mathbf{U}}) = \mathbf{R}, \tag{4.5}$$

where α_1 and α_2 are the Rayleigh coefficients. We denote $\mathbf{W}$ the solution to the generalised eigenproblem $\mathbf{K}\mathbf{x} + \lambda\mathbf{M}\mathbf{x} = 0$ and $\mathbf{\Lambda}$ the diagonal matrix of eigenvalues, then (4.5) may be expressed as:

$$\mathbf{\Lambda}(\mathbf{Z} + \alpha_1\dot{\mathbf{Z}}) + (\alpha_2\dot{\mathbf{Z}} + \ddot{\mathbf{Z}}) = \mathbf{G}, \tag{4.6}$$

where $\mathbf{Z} = \mathbf{W}^{-1}\mathbf{U}$ is the vector of modal coordinates and $\mathbf{G} = \mathbf{W}^T\mathbf{U}$ is the external force vector in the modal coordinate system. Each row of this equation now corresponds to a single scalar differential equation:

$$\ddot{z}_i + (\alpha_1\lambda_i + \alpha_2)\dot{z}_i + \lambda_i z_i = g_i. \tag{4.7}$$

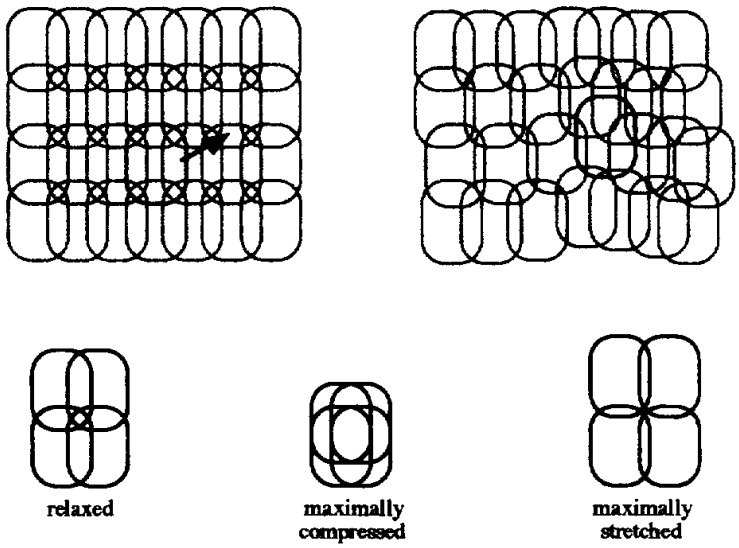


Figure 4.5: Deformation of a 2D chain mail when the link indicated by an arrow is moved. Image courtesy of Gibson (1997).

The analytical solutions to each equations are well known and are the following:

$$z_i = c_1 e^{t\omega_i^+} + c_2 e^{t\omega_i^-} \tag{4.8}$$

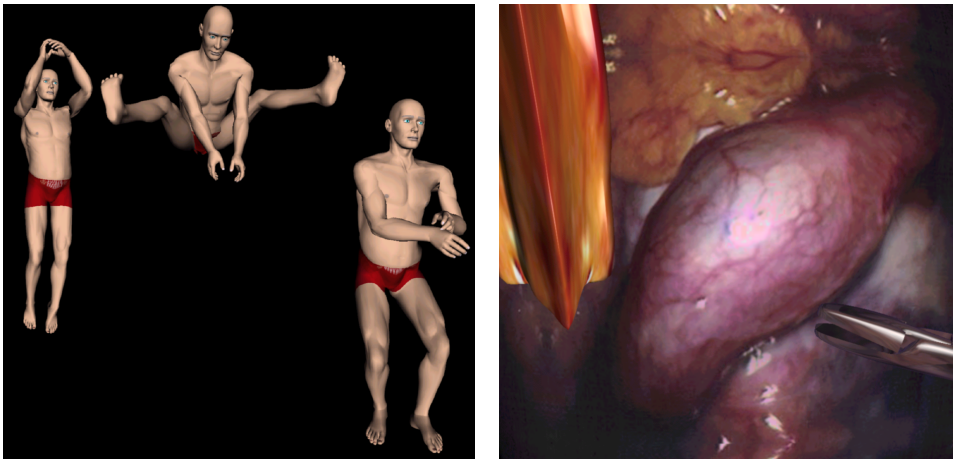
where c_1 and c_2 are complex constants and ω_i is the complex frequency given by:

$$\omega_i^{\pm} = \frac{-(\alpha_1\lambda_i + \alpha_2) \pm \sqrt{(\alpha_1\lambda_i + \alpha_2)^2 - 4\lambda_i}}{2}. \tag{4.9}$$

The main advantage of modal analysis is the possibility to model only the necessary modes. Indeed if the eigenvalue λ_i associated with a particular mode is large, then the force required to cause a discernible displacement of that mode will also be large. Similarly, some displacements may be too small to be detected. Removing those modes from the computation will not change the appearance of the resulting simulation while substantially reducing the computational cost.

James and Pai (2002) went even further by using graphics hardware to reduce CPU costs to a minimum. In particular, they use modal analysis to model human skin and secondary tissues in a laparascopic surgical simulation to allow the CPU to focus on more complex tissue models and user contact interactions (see Fig. 4.6). The technique is fairly efficient as it allows precomputations of the vibration modes.

Although modal analysis significantly accelerates the simulation, it generates noticeable artifacts when applied to large deformations due to the step of linearisation. It can produce unnatural results with deformations of large amplitude. **?** proposed to overcome this limitation by taking the rotational component of the deformation into account at each node. Barbič and James (2005) make use of modal analysis to



(a) (b)

Figure 4.6: (a) A jumping motion that leads to significant thigh and belly vibrations of human skin when rendered with modal analysis. (b) The same hardware accelerated technique is applied to secondary tissues in a laparoscopic simulation to ease the CPU of computations. Image courtesy of James and Pai (2002).

build a deformation basis from modal derivatives. It allows them to simulate large deformations of solid objects with non-linear material.

4.3.3 Mass-spring model

A classic technique for modelling a deformable object is the mass-spring system. In this method, the geometry is described by a network of masses connected together by springs. The force $\mathbf{F}$ exerted onto each mass by the spring may be computed with:

$$\mathbf{F}_{\text{spring}} = -k\mathbf{x}, \tag{4.10}$$

where k is the spring stiffness constant, $\mathbf{x}$ represents the stretch of the spring from its rest position. From our experience, we know that a mass attached to a single spring has a tendency to oscillate. If no appropriate measure is taken, a network of springs will a forciori oscillate as well. To prevent the system from oscillating, a damper is also added between each couple of masses. The force created by the damper acts to reduce the velocity and may be expressed as:

$$\mathbf{F}_{\text{damper}} = -d\dot{\mathbf{x}} \tag{4.11}$$

where d is the damping constant and $\dot{\mathbf{x}}$ the derivative of x with respect to time. The damper added between each couple of masses allows to model friction and reduce the amplitude of the oscillations. Applying Newton's second law $\sum \mathbf{F} = m\ddot{\mathbf{x}}$ yields the standard differential equation for a mass-spring system:

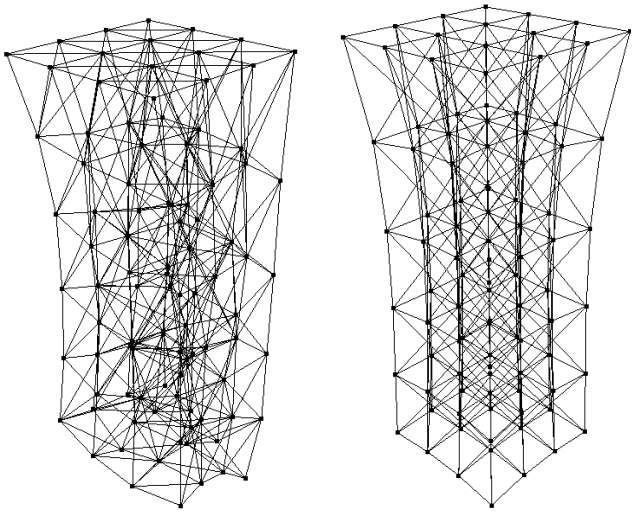
$$m\ddot{\mathbf{x}} = -d\dot{\mathbf{x}} - k\mathbf{x}. \tag{4.12}$$

For complex systems with more than one spring attached to each mass, the right-hand side of the equation (4.12) becomes more complex as it needs to be repeated and adjusted for each spring. The easiest way to solve a complex mass-spring system iteratively, is to first calculate the total force currently acting on each mass by summing all forces currently exerted by all springs attached to this mass (using (4.10) and (4.11)). Dividing the total force by the mass gives the current acceleration of each mass (by (4.12)). This equation must be numerically solved for each mass, either by explicit or implicit methods. A discussion of integration methods for mass-spring models can be found in Shinya (2005).

Mass-spring systems have been extensively used and are still very popular. And this is not surprising because they are easy to implement and computationally efficient. Of course, they have drawbacks. But as we will see shortly, various recent works attempt to circumvent them. One limitation often mentioned is the difficulty to derive spring stiffnesses from elastic properties (Young's modulus and Poisson's ratio). To overcome this deficiency, Lloyd et al. (2007) mention two ways of obtaining the parameters of a mass-spring model. The first one consists in varying the parameters until the behaviour of the system is similar to the one obtained through the experiments or the one obtained with the Finite Element Method (Bianchi et al., 2004). The second way consists in establishing an analytical reasoning for calculating the constants of the mass-spring model. For instance, starting from the definition of Young's modulus, Poisson's ration, shear and bulk modulus, Baudet et al. (2007) apply simple tests to their mass-spring model to find the most appropriate stiffnesses. Lloyd et al. (2007) proposes a linearisation of the mass-spring equations with the aim of equating the linearised stiffness matrix to the stiffness matrix of a linear finite element method. By following the same method, San Vicente et al. (2009) derived a mass-spring model equivalent to a linear finite element model for maxillofacial surgery simulation.

Because they do not derive from the equations of continuum mechanics, mass-spring models have a limited capacity to model the various aspects of a material like anisotropy, viscoelasticity etc. Even worse, according to Bourguignon and Cani (2000), if all springs are set to the same stiffness, the mesh geometry may generate undesired anisotropy as shown in Fig. 4.7. However, some works tried to take advantage of this feature by giving directions of interest to their model by specifically designing the mesh to align springs on specific directions. For instance, it was used by Miller (1988) to animate the motion of snakes and worms. It was also used by Ng-Thow-Hing and Fiume (1997) in their muscle model where some of the springs were aligned with the muscle fibers and more recently by Magjarevic et al. (2009) for modelling the mechanical behavior of the myocardial tissue. More details on controlling anisotropy in mass-spring systems may be found in Bourguignon and Cani (2000).

A mass-spring model may also be enhanced to feature viscoelasticity. Indeed, Stiles and Alexander (1972) proposed a viscoelastic mass-spring system to test hypotheses on the mechanical response of muscles. Much more recently, Tamura et al.



(a) (b)

Figure 4.7: The two meshes are undergoing a downward pull at their bottom. While the tetrahedral mass-spring system (a) shows signs of anisotropy, the hexahedral mesh (b) with springs aligned along the gravity does not. Image courtesy of Bourguignon and Cani (2000).

(2005) introduced a method to simulate viscoelastic material using a mass-spring system. They use a large number of particules to create a randomly connected mesh to mimic the structure of polymeric materials and hence their viscoelascity characteristics. Basafa and Farahmand (2010) went even further and extended a mass-spring system to simulate non-linear viscoelastic deformations of soft tissue for laparoscopic surgery. They tune their parameters by a simple optimisation procedure to fit the mechanical response obtained on a set of experimental data.

Terzopoulos et al. (1991) even designed a mass-spring system that experience transitions from solid to liquid. Nodes are connected by springs in a hexahedral lattice and each of them has an associated temperature. Spring stiffnesses decreased with the increase in temperature. The diffusion of heat through the material is computed by using a discretised form of the heat equation. When the melting point is reached, the stiffness is set to zero and the node is detached from this particular spring. Once a node is freed from all his neighbouring springs, it becomes an independent glop of fluid and its behaviour is then modelled using a discrete fluid model.

Early 2000s, the rapid increase in the performance of graphics hardware, coupled with recent improvements in its programmability, have made graphics hardware a compelling platform for computationally demanding tasks in a wide variety of application domains. Researchers and developers have become interested in harnessing this power for general-purpose computing, an effort known collectively as

GPGPU (for General-Purpose computing on Graphics Processing Units). A survey of GPGPU algorithms and techniques was written by Owens et al. (2007).

Mosegaard et al. (2005); Mosegaard and Sørensen (2005) were probably the first to implement a mass-spring model on GPU in medical simulation. For the first time, the whole simulation of was performed on the GPU (physics, interaction and visualisation) and they used their framework in a congenital cardiac surgery simulation. They compared two implementations: the first one explicitly defined the connectivity between springs using a connectivity texture while the second one determines implicitly determines at runtime that springs are connected when they are neighbours. They were able to compute 188 iterations per second on a 42 745 particule model of a pig heart. Depending on the size of the model and the type of implementation, the authors achieved speed-ups between 10 and 30 × over their CPU implementation.

Taking advantage of the GPU to reach a substantial speed-up is usually not straightforward. One must understand the limitations inherent in its design and devise algorithms accordingly. With this idea in mind, Sørensen and Mosegaard (2006) reviewed the most important concepts and data structures required to realise two popular deformable models on the GPU: the spring-mass model and the finite element model.

TODO: Other GPU-based mass-spring implementations?

If mass-spring systems are quite versatile and easy to implement, the most complete mathematical formalism available to describe the mechanical behaviour of a solid is provided by continuum mechanics. Consequently, it feels natural to derive computational models from the equations of continuum mechanics and a few techniques based on these equations will now be introduced.

4.4 Techniques based on continuum mechanics

4.4.1 The finite element method

The finite element method brings accuracy a step further. The method will not be explained here since it was already discussed with great details in chapter 3. This is the method of first choice for modelling deformable objects with precision as it is directly derived from the most complete theory available for describing the mechanical response of a continuum. Different kind of analysis may be carried out with the finite element method. As seen in section 3.3, an analysis may be either static (that is, the transient response is ignored) or dynamic (more general). In all analyses, the deformation of the body at a point is described by some measure of strain. If this measure is chosen to be linear (see section 2.4.5 for details), the analysis is said to be *geometrically* linear. In general, we consider this assumption to be valid for strain less than 10 %. In contrast, geometrically non-linear analyses make use of a non-linear strain tensor to measure the deformation and thus may handle large deformations. Another characteristic of an analysis is whether the relationship between strain and stress (the constitutive model) is assumed to be

linear or non-linear. The analysis is qualified as *materially* linear or non-linear, respectively.

Because of its complexity, the finite element method is substantially more computationally expensive than the ones introduced so far. And this has been a problem in computer graphics and interactive simulation for many years. So researchers came up with simplifying assumptions and ideas to make the computations more efficient. Over time, with the increase in computational power, the simplifications were progressively relaxed and the observed tendency for real-time simulation is now towards more and more complete finite element methods. In this section, we will describe this evolution in the use of finite element methods for interactive simulation.

The use of the finite element method in computer graphics was pioneered by Terzopoulos et al. (1987) for simulations of elastic deformations. They derived a geometrically non-linear strain tensor from differential geometry theory. Obviously, the computations were not carried out in real-time but this technique allows Terzopoulos et al. to calculate the interaction between solid and deformable models in a physically realistic manner.

In order to speed up the computations, linear strain measures started to be used (see section 2.4.5). For instance, Gourret et al. (1989) make use of a linear finite element method to model the deformation of grasped objects. The equations are however solved in static for simplicity. An approach to solve the system of equations in dynamic in a reasonable amount of time is to decrease the number of elements. This reduction must be compensated by the use of high-order elements, that is the degree of the shape functions used in the interpolation of the solution is higher to maintain the overall accuracy. This approach was applied to an elaborate model of a muscle by Chen and Zeltzer (1992) and to modelling skin in a simulator of plastic surgery by Pieper et al. (1992) where they both use 20-node brick elements. Yet, finite element modelling remained very computationally expensive and the interactivity required for medical simulators was still to be reached.

In 1996, Bro-Nielsen and Cotin developed a real-time finite element method formulation by introducing three improvements. The first one is to compress the stiffness matrix of the volumetric system into a matrix with the same complexity as a surface model of the same object. This technique is called *condensation*. The deformation is solved only at surface points while still taking the volumetric nature of the object into account. The second improvement concerns the fact that they explicitly precompute the inverse of the stiffness matrix and use matrix vector multiplication with this matrix to achieve a low calculation time. Finally, they exploit the sparse structure of the force vector. They showed the efficiency of their method by modelling a leg with 700 nodes at 20 frames per second.

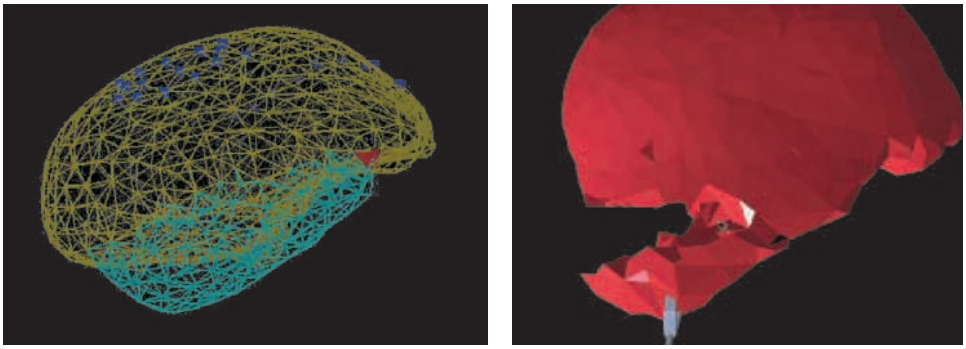
Later, James and Pai (1999) had another idea to improve the speed of the computation. They based their model on boundary integrals and the *boundary element method*, used for the first time in computer graphics. The method consists of calculating only boundary displacements and forces. Unfortunately, the authors do not provide the performance of their framework.

Cotin et al. (1999) took advantage of the superposition principle which applies to linear problems. Their idea was to approximate the deformation of a solid object with a linear combination of pre-computed states. In addition, they enhanced the formulation by adding a non-linear radial component. This method was applied to compute the deformation of liver in a hepatic surgery simulator. Their approach substantially improved the computational time for a finite element model as it allowed them to compute the positions and the forces of a 1 500 node liver at 300 Hz. An important drawback emphasised by the authors themselves is the impossibility to simulate tissue cutting. Indeed, the large precomputations associated with this method are dependent of the geometry of the mesh and therefore cannot be updated in real-time.

Besides real-time computation of soft tissue, another major concern for medical simulators is the ability to cutting through soft tissue models. Mor and Kanade (2000) proposed a technique to generate a minimal set of new elements to replace intersected tetrahedra along the path formed by the cut. The method do not wait for the cut to be completed for splitting an element, instead, a minimal subdivision of a partially cute tetrahedron is generated. The authors used a very small time step to insure the stability of the soft tissue model and their simulations were not real-time. In addition, their method can create very small elements and the model can become unstable. This occurs because the stiffness matrices for the small elements are quite large, and even small displacements can generate very large forces.

Cotin et al. (2000) proposed a hybrid model allowing real-time cutting. They combined a quasi-static precomputed model, which is very efficient but does not allow topology changes, with a tensor-mass model which requires more computation but authorises cutting. The key idea is to separate a same structure into two parts: one susceptible to be cut during the surgery procedure and one that will not be submitted to any topological change. While the former will be simulated with the tensor-mass model, the deformation of the latter will be computed using the precomputed model. Since both linearly elastic models follow the same physical law, the combination of these two models should behave exactly as a global, linearly elastic model. To achieve this goal, Cotin et al. describe how to impose the additional boundary conditions at the connection nodes for the global model to be consistent in terms of forces and displacements. The efficiency of this hybrid model was demonstrated by simulating an hepatectomy (removing of one of the eight anatomical segments of the liver). They reported the simulation (allowing deformation and cutting) of a mesh with more than 8 000 tetrahedra in real-time (see Fig. 4.8).

Serby et al. (2001) described a new approach to cutting with finite element models. The central idea is not to introduce new nodes/elements but to displace the existing ones to account for the topological changes introduced by a cut. They also added a step of homogenisation of the mesh to avoid tiny elements. Thus, the problem of decreasing element size is minimised and consequently the stability of the solution of the equations of motion is increased.



(a) (b)

Figure 4.8: Simulation of a hepatectomy using a hybrid model. 18% of the mesh was modelled with the tensor-mass model (green) and therefore could be cut out. The rest of the mesh was simulated with the precomputed finite element model. Image courtesy of Cotin et al. (2000). TODO: Stéphane> tu dis que l'image est moche, mais c'est pourtant bien celle présente dans le papier... Et même dans ton HDR tu utilises cette image... regarde page 46 et 50... Mine de rien ça fait plus de 10 ans ;-)

Cutting is a problem for all techniques precomputing the inverse of the stiffness matrix. Indeed, topological changes require updates of this matrix, which is too computationally heavy to be done in real-time. Nienhuys and van der Stappen (2001) avoid the precomputations by approximating the inverse at runtime with an iterative algorithm (Conjugate Gradient). Moreover, instead of subdividing each tetrahedron to make the cut appear where the user performed it, they adapt the mesh locally so that there are always triangles on the scalpel path, and perform the cut along those triangles (see Fig. 4.9). If the approach is interesting, the authors reported a lag between the scalpel and the realised cut. In addition, repositioning the nodes within the mesh can easily generate degeneracies, which have to be eliminated. TODO: add PhD of Brian Lee... Mais je l'ai pas sa thèse... et tu m'as pas dit que tu étais en train de la reviewer ? Du coup c'est pas encore publique, c'est difficile de la citer, non ?

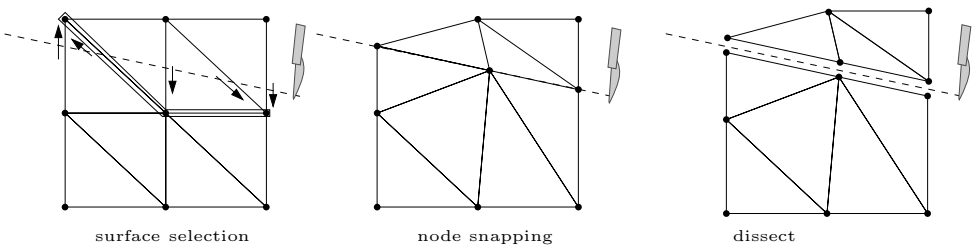


Figure 4.9: The three steps in performing a cut (represented here in 2D). Image courtesy of Nienhuys and van der Stappen (2001).

As we know, the assumption of linearity is usually only valid for very small deformations and strains. If this is often an acceptable trade-off between the visual deformation result and the speed of the system, research was carried out to handle large deformations. However, since geometrically non-linear analyses are highly computationally demanding, another approach was found which allows large deformation while still relying on a linear strain measure: the *co-rotational* methods . A very complete description of the co-rotational approach is given by Felippa (2000). As explained in section 2.4, an appropriate strain measure is a tensor which only measures the deformation applied to the object. This is the case of the Cauchy-Green or Green tensors for instance. Those strain metrics only measures deformation, they are invariant with respect to rigid-body transformations applied. However, geometrically linear analyses make use of the infinitesimal strain tensor, which is only an approximation of the Green tensor. Consequently, this linearised strain measure is not invariant with respect to rigid-body transformations. In particular, the error brought by the approximated (linear) strain measure increases with the rotation of the element. This error in the measure creates ghost forces which overdeforms the simulated object. The idea behind co-rotational methods is the decomposition of the motion into a rigid-body and deformational components. In other words, the deformation of each element may be seen as measured in a coordinate system which follows the rigid-body motion of the element, hence cancelling the part of the error due to the rotation in the strain measure. Consequently, co-rotational formulations allow large displacements and rotations (since the rigid-body motion is not considered) as long as the strain remains small (the strain measure is still linear).

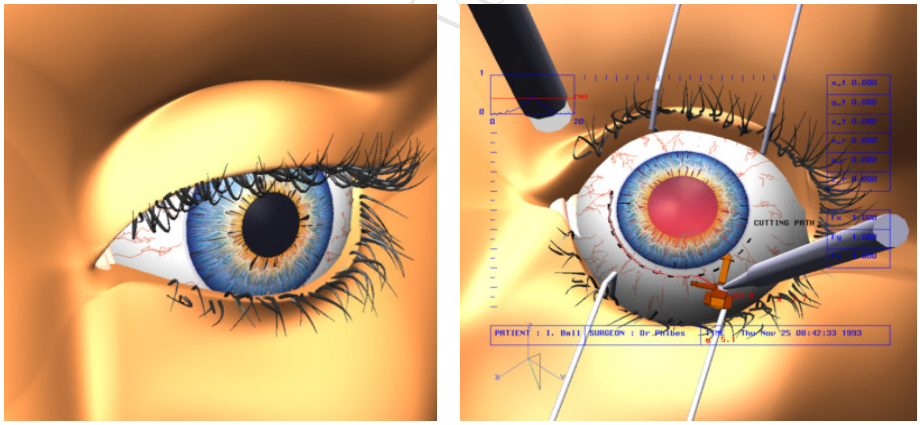
Among the firsts to apply this technique to static analyses are Argyris et al. (1964) and Wempner (1969). Although the birth of co-rotational methods is not very clear (Felippa and Haugen, 2005), Belytschko and Hsieh (1973) had a similar thinking and introduced the so-called *convected coordinates* for solving dynamic equations. However, while the spirit is the same than co-rotational methods, convected coordinates forms a curvilinear system that fits the change of metric as the body deforms, therefore the convected metric necessarily emcompasses deformations. In other words, the decomposition in rigid-body and deformational components is not exact. During the last decade, the co-rotational approach has regained interest in the computer graphics community.

In constrast of Müller et al. (2002) who discretised the rotation field at each vertex, Hauth and Strasser (2004) determined the rotation at the level of the element. If Müller et al.'s technique of stiffness warping yields ghost forces due to inaccuracies in the determination of the rotation field, Hauth and Strasser made use of the polar decomposition of the deformation gradient to extract the exact rotation. They also applied this formulation in a hierarchical finite element setting where the coarse level is computed with a non-linear strain tensor and the finer levels are computed with a co-rotational method. In addition, they examined the error that is arising from using the co-rotational approximation instead of a non-linear strain tensor. They measured that the co-rotational approximation induces an error of about $5 - 20\,\%$

but can give a speedup of about a factor seven or more.

Pouliquen et al. (2005) addressed the simulation of human fingers with haptic feedback. During the grasping task, the deformable finger pads undergo large rotations while their deformation remains small. Using a co-rotational approach for modelling the finger pads allows them to efficiently compute the contact forces between the deformable pads and the rigid object to be grasped.

If co-rotational approaches increase accuracy in the strain measure, all the examples we have seen so far make use of a linear constitutive law. Yet, the mechanical behaviour of soft tissue is known to be non-linear, viscoelastic and often anisotropic (Fung, 1993). Moreover, Misra et al. (2007) demonstrated a noticeable difference between the force feedbacks provided by linear and non-linear tissue models. One of the first materially non-linear finite element formulation used in medical simulation was probably the work of Sagar et al. (1994). While developing a teleoperated micro-surgical robot for eye surgery, they wanted to provide a virtual environment has part of the system for training. Since the cornea is anisotropic and non-linearly elastic and undergoes large deformations, they have used large deformation elasticity theory with orthotropic and non-linear material properties in the finite element model. Coupled with a detailed visual model of the eye, the simulation produces a fair level of realism with a performance greater than 10 Hz (see Fig. 4.10).



(a) (b)

Figure 4.10: (a) Exterior view of the model showing the eye and the eyelashes. (b) Surgical virtual environment including micro-tools and guidance information. Image courtesy of Sagar et al. (1994).

Keeve et al. (1998) compared a fully (geometrically and materially) non-linear finite element model with a mass-spring model to compute the deformation of skin after bone realignment in a craniofacial surgery simulator. The efficiency of the mass-spring model gives the ability to the surgeon to realise interactive simulation of the resulting tissue changes and to improve his planning process. The best precision was given by the finite element model which could be used off-line to verify the

chosen surgical procedure.

The first geometrically non-linear finite element model running in real-time was reported by Debunne et al. (2001). They provided an adaptative method for animating viscoelastic deformable objects in a guaranteed frame rate. As the object moves and deforms, the sampling is refined to concentrate the computational load into the regions that deform the most (see Fig. 4.11). They reported animating a few hundred points in real-time at a frequency higher than 300 Hz.

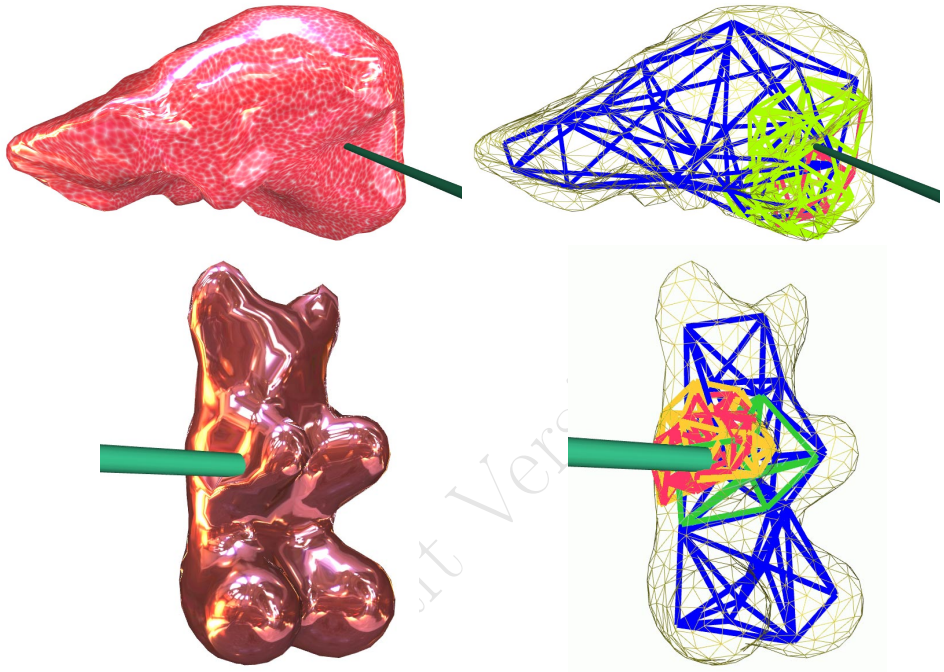


Figure 4.11: The approach of Debunne et al. (2001) makes use of a local refinement technique to ensure high physical fidelity while bounding the global computation load to guarantee real-time simulation with a geometrically non-linear and viscoelastic finite element model.

In the same spirit, Wu et al. (2001) applied an adaptative meshing technique to provide sufficient detail where required while minimising unnecessary computation. They included both Mooney-Rivlin and Neo-Hookean material models and used the non-linear Cauchy-Green tensor. In addition to the local mesh refinement technique, they used mass lumping and an explicit time integration to deform a liver model of 1 210 tetrahedra in real-time.

Still in 2001, Picinbono et al. proposed to use a non-linear model only where the displacements are larger than a given threshold, the remaining part of the object still uses linear elasticity.

Recently, research on non-linear finite element and more elaborate constitutive models increased significantly. For instance, Pathmanathan et al. (2004) used a

fully non-linear model to predict deformation of breast under compression during mammography. However, they were not interested in real-time. Zhong et al. (2005) claimed achieving real-time with their non-linear finite element method using an interpolation approach but do not give any figure. Sedef et al. (2006) proposed a viscoelastic model to simulate the deformation of a liver. They calculate the deformation in real-time by carrying out the computations in static and using the superposition principle. However, their approach used a linear strain measure, which limits their system to small deformations. Yan et al. (2007) claimed to compute their non-linear finite element model in real-time using a technique of *graded mesh* but do not give any information on their performance.

In 2007, Miller et al. proposed the total Lagrangian explicit dynamics (TLED) algorithm, a very efficient fully non-linear formulation. The algorithm is based on the finite element method using the total Lagrangian formulation, where stresses and strains are measured with respect to the original configuration. This choice allows for precomputing of most spatial derivatives before the commencement of the time-stepping procedure. The authors reported that the average number of floating-point operations per element per time step is 35 % lower than for the similar implementation of the algorithm based on updated Lagrangian formulation. They were able to compute the deformation of a hexahedral mesh of 6 000 elements in about 16 ms for a time step of 1 ms. If they could not achieve real-time computations, their work constituted a step towards the simulation of entire organs in real-time.

Finally, Taylor and Hawkes (2007) presented an efficient constitutive update procedure for viscoelastic models suitable for simulation of soft tissues at large strains and varying strain rates. The procedure is formulated for use in explicit dynamic finite element algorithms like the TLED algorithm. The procedure is based on a class of visco-hyperelastic constitutive models developed from purely hyperelastic strain energy functions by introducing relaxation terms and expressing in convolution integral form. The resulting constitutive equations are separated into rate-dependent and -independent terms, with the former being designated as state variables to be stored and updated at each time step also. These are converted to a differential form, which upon integration in time yields the required incremental update formula. They showed the validity of the procedure with a series of numerical examples.

The finite element method is very computationally demanding. However, by nature, very similar computations are carried out for each element. Consequently, the finite element method is a good candidate for being implemented on parallel architectures. Thus, Székely et al. (2000) implemented a fully non-linear formulation for an 8-processor machine. However, the dedicated hardware was not yet assembled and functional at the time so they did not provide any measure of performance.

Early 2000s, the rapid increase in the performance of graphics hardware, coupled with recent improvements in its programmability, have made graphics hardware a compelling platform for computationally demanding tasks in a wide variety of application domains. Researchers and developers have become interested in harnessing this power for general-purpose computing, an effort known collectively as

GPGPU (for General-Purpose computing on Graphics Processing Units). A survey of GPGPU algorithms and techniques was written by Owens et al. (2007).

To the best of our knowledge, Rumpf and Strzodka (2001) were the firsts to leverage the power of GPUs to solve partial differential equations. They established a correspondence between common mathematical operations and OpenGL operations and structures. Thus, as an example, a vector is represented as a RGBA color vector in a color image on the graphics card. They decided to implement an iterative solver for a linear system of equations, which we find at the core of most finite element codes. Using this hardware accelerated solver, they solved the linear heat equations and presented an anisotropic diffusion model for image processing. They obtained promising results but noted the hardware restrictions that forced them to approximate all non-linear functions by linear ones in the implementation of the anisotropic diffusion. Although a complete hardware accelerated finite element method implementation was yet to be done, Rumpf and Strzodka definitely stepped towards the right direction.

Wu and Heng (2004) proposed the first application of GPU in medical simulation. They were able to achieve an interactive frame rate with topology changes in surgical simulation. Not only they used the condensation technique for merely calculating the surface nodes in the non-operation parts, but they implemented the conjugate gradient solver onto GPU. Indeed, entering 2003, the full IEEE floating point precision data type supported in shaders and texture stages allows the GPU to be applied into more general applications. The conjugate gradient solver includes three primary operations: multiplication of the sparse matrix and the vector, addition of two vectors and the sum-reduction operation. The core component is the multiplication of the sparse matrix and the vector. Therefore, they migrated this part of calculation from the CPU into the fragment processor of the GPU, to take advantage of the fragment processor in its efficient manipulation of the local texture memory on the mathematical calculation. The performance of their method on the GPU is up to 2.15 times faster than the CPU implementation.

In 2007, Taylor et al. introduced the first GPU implementation of a fully non-linear finite element method (more details in Taylor et al. (2008b)). They reformulated the TLED algorithm presented by Miller et al. (2007) to accomodate with the limited hardware possibilities of GPUs. In particular, the main restriction when using graphics-based GPU execution is the inability to scatter, which necessitated reformulation of force summation as a gather in their implementation. It is worth noting that the entire finite element method implementation was ported to GPU. They reported significant solution speed gains up to 16.8× over the CPU implementation. Thus, they were able to compute a simple cube model with up to 16 000 tetrahedral elements in real-time.

4.4.2 Meshless methods

If the finite element method is a robust method and widely used in engineering, the method has a few shortcomings (Liu and Gu, 2005). First, the creation of a

mesh is compulsory and producing a good quality mesh is both difficult and time consuming. Second, the stresses obtained in FEM are often discontinuous at the interfaces of the elements. Special techniques are required in a post-processing stage to recover accurate stresses. In addition, under large deformations, considerable loss in accuracy in FEM results can arise from the element distortions. The origin of these problems is the use of elements. Consequently, the idea of getting rid of the elements and the mesh has emerged and the concept of *meshless* methods was born. Instead, meshless methods use a set of nodes scattered within the problem domain. They do not form a mesh, meaning it does not require any a priori information on the relationship between the nodes for the interpolation of the unknown variables. In theory, this should facilitate topological changes, a feature fairly desirable in surgery simulation. It is worth noting though that meshless methods are still in their developing stage and are not as mature as finite element formulations. We will now briefly review the research carried out in soft tissue modelling for medical simulation that makes use of meshless method. A complete review and details on meshless methods are beyond the scope of this thesis.

A meshless method was introduced for the first time for modelling soft tissue in medical simulation by De et al. (2001). They use the so-called method of finite spheres (MFS). Nodal points are only sprinkled locally around the surgical tool tip and not over the whole domain. The interpolation is performed by functions that are nonzero only on spheres surrounding the nodes. A point collocation technique is used to generate the discrete equations. The authors qualitatively compared the method of finite spheres with the finite element technique from a commercial package and the displacement profil was similar in both cases. They were able to achieve a computational rate of about 100 Hz when 34 spheres where used. Real-time haptic rates of about 1 kHz was then obtained using a force extrapolation technique.

Lim and De (2004) used the same technique of finite spheres introduced by De et al. to model soft tissue but they added progressive cutting, without the generation of new primitives. As the cut progresses, the nearest vertex to the intersection point of the tool is snapped to the tool path. Then, they observed that the prescribed boundary condition changes on only a very small portion as the tool interacts with the organ. Rather than solving the entire problem over and over again, they proposed instead to make incremental corrections, which results in an accelerated solution procedure.

Meshless methods were then introduced to computer graphics by **?**. In each step, they compute the spatial derivatives of the displacement field using a moving least squares (MLS) procedure. From these derivatives, they obtain strains, stresses and elastic forces at each simulated point. In addition, they proposed techniques for modelling and animating a point-sampled surface that dynamically adapts to deformations of the underlying volumetric model. They used a non-linear measure of strain with a linear constitutive model. They reported real-time simulation of models exibiting elastic, plastic, melting and flowing effects.

Horton et al. (2006) tested meshless methods for surgical simulation. They

proposed a total Lagrangian explicit dynamics algorithm using a non-linear material formulation. They applied the method on an example of brain shift and compared their results with a commercial finite element code. The shape functions are created from a cloud of unconnected nodes using the element free Galerkin method. They were interested in the displacement of the centre of mass for the brain and found the absolute displacement differences between the two algorithm are less than 0.85 mm which is the resolution of the MRIs in their study. The same team kept investigating and testing the technique. However, they used moving least squares procedure for creating the shape functions instead of the element free Galerkin method. They also realised experimental indentations (Horton et al., 2007) and provided more extensive comparisons with a finite element method (Horton et al., 2010).

Another example of application is the used of the technique called moving particule semi-implicit method (MPS) by Chhatkuli et al. (2009). They applied this method to simulate the deformation of lung and a tumor inside the left lung during inspiration. The lung tissues were considered to be homogeneous, isotropic and viscoelastic. They compared their results with the experimental CT taken at the end of inspiration and found out that the deformation predicted by numerical simulation matches reasonably well with the experimental results.

Recently, Zhu et al. (2010) proposed a framework based on point-based simulation techniques to model organ deformations realistically in a laparoscopic simulator. They combined a new strain-stress tensor computation scheme with smoothed particule hydrodynamics (SPH) method. The parameters of the tensor relating strain and stress were determined through mechanical experiments.

The main advantage of meshless methods is of course in the absence of a mesh. This advantage of not having to go through the complex process of meshing was often claimed by meshless methods partisans. It also allows much easier cutting as it saves the cost of a computationally demanding remeshing around the cut. Yet, in contrast of finite element methods, meshless techniques are young and still in development. Boundary conditions are an aspect of the numerical solution of partial equations where meshless methods have many difficulties for instance. According to Horton et al. (2010), if meshless methods are accurate in terms of overall reaction forces, they are not as good with individual displacements or forces. Therefore the authors suggest to use a meshless algorithm to fill the interior sections of an organ (where a local discrepancy has minor consequences) and use the finite element method for the boundary. As a conclusion, if meshless methods are promising, the technique does not seem as mature as the finite element method for modelling soft tissues.

Chapter 5

The total Lagrangian explicit dynamics (TLED) algorithm

In 2007, Miller et al. proposed the total Lagrangian explicit dynamics (TLED) algorithm, a very efficient fully non-linear formulation. The authors reported that the average number of floating-point operations per element per time step is 35 % lower than for the similar implementation of the algorithm based on updated Lagrangian formulation. Their work constituted a step towards the simulation of entire organs in real-time. Later in the year, Taylor et al. reformulated the TLED algorithm to propose the first GPU implementation of a fully non-linear finite element method. However, restrictions due to using a graphics-based API necessitated reformulation of the force summation in their implementation. Taylor and Hawkes (2007) also presented an efficient constitutive update procedure for viscoelastic models formulated for use in explicit dynamic finite element algorithms like the TLED algorithm. With the release by NVIDIA of a new graphics card architecture along with a new and more flexible non-graphics API specially designed for general-purpose GPU, we decided to investigate the re-implementation of the TLED for this new architecture with the idea of a more straightforward and efficient implementation in mind. We also wanted to extend the simple non-linear TLED formulation by adding the constitutive update procedure for viscoelastic models introduced by Taylor and Hawkes and an anisotropic constitutive formulation as well. This work eventually yields a very efficient GPU-based non-linear, viscoelastic and anisotropic formulation for modelling solid organs in medical simulation and was the object of three publications (Comas et al., 2008; Taylor et al., 2008a, 2009). This chapter aims at describing the TLED algorithm in details, as well as introducing the viscoelasticity and anisotropy extensions. Their GPU implementations will be the object of the next chapter.

5.1 Description of the TLED algorithm

5.1.1 Key ideas

The total Lagrangian explicit dynamics (TLED) algorithm was first introduced to the field of medical simulation by Miller et al. (2007). More details on this finite element method formulation may also be found in Bathe (1995). It is worth noting that the finite element analysis carried out by the TLED algorithm is dynamic and fully non-linear (geometrically and materially). Let us begin by exposing the key ideas of the approach.

The deformation of a body may be described by two different kind of description: Lagrangian or Eulerian (see section 2.2). Because the Lagrangian description focuses its attention on the particles of the continuous body, it is usually used in solid mechanics. In the Lagrangian description, the position and physical properties of the particles are referred to a reference configuration. For the great majority of commercial finite element programs, the reference configuration is the previous configuration, that is the one at the end of the previous time step. In this case, where all variables are referred to the previous configuration, the formulation is called *updated Lagrangian*. The advantage of this approach is the simplicity of the incremental strain description. The disadvantage is that all derivatives with respect to spatial coordinates must be recomputed in each time step, because the reference configuration is changing. The reason for this choice is historical, at the time of solver development the memory was expensive and caused more problems than actual speed of computations (Miller et al., 2007).

The first key idea of the TLED algorithm is to refer all variables to the undeformed configuration. This type of formulation is said to be *total Lagrangian*. Because of this, the choice of the second Piola-Kirchhoff stress tensor as a measure of stress is required. The strain measure that is work-conjugate with the second Piola-Kirchhoff stress is the Green-St.Venant strain tensor. That is, the work of stress increments on the strain increments gives an accurate expression for work (Ji et al., 2010). We should note that all derivatives in the definition of the Green tensor are with respect to the initial and undeformed configuration. The decisive advantage is that all derivatives are calculated with respect to the undeformed configuration and therefore can be precomputed. This is at the cost to a more complex strain-displacement matrix due an initial displacement effect in the incremental strain (Bathe, 1995). However, the TLED algorithm performs significantly fewer mathematical operations in each time step.

The second crucial idea is to use the central difference method for an explicit time integration of dynamic equilibrium equations. The advantage is that the stiffness term $\mathbf{K}(\mathbf{U}).\mathbf{U}$ of the system of equations may be computed from:

$$\mathbf{K}(\mathbf{U}) \cdot \mathbf{U} = \mathbf{F}(\mathbf{U}) = \sum_{e} \tilde{\mathbf{F}}^{e}, \tag{5.1}$$

where $\tilde{\mathbf{F}}^{e}$ are the global nodal force contributions due to stresses in element Ω_e. This implies that the stiffness matrix does not need to be assembled since element

nodal force contributions can be computed at the element level instead, which is a decisive computational advantage.

We will first explain how to compute element nodal forces based on the deformation gradient and a neo-Hookean constitutive model. We will then show how to use these element nodal forces to calculate the unknown displacements for each node of the model (Taylor, 2006).

5.1.2 Computation of element nodal forces

We adopt the notation of Bathe with respect to indication of the relevant configuration of the body: a left superscript indicates the configuration in which a quantity occurs and, when applicable, a left subscript indicates the configuration with respect to which the quantity is measured. We note the coordinates of a point ${}^t\mathbf{x}$ at time t.

The deformation gradient

As we know, a fundamental measure of deformation is the deformation gradient tensor ${}^t_0\mathbf{X}$ and we recall that it may be written as:

$$ {}^t_0\mathbf{X} = \frac{\partial {}^t\mathbf{x}}{\partial {}^0\mathbf{x}}. \tag{5.2} $$

This tensor describes the stretches and rotations that the material fibres have undergone from time 0 to time t. In order to compute the element deformation gradients, we first compute derivatives of displacements $u_{i,j}$ with respect to global coordinates. Since we use a total Lagrangian framework, the derivatives are referred to the undeformed configuration and we have:

$$ {}^t_0u_{i,j} = \frac{\partial {}^t u_i}{\partial_0 x_j} = \sum_{a=1}^{N} \frac{\partial h_a}{\partial_0 x_j} {}^t u_{ai}, \tag{5.3} $$

where h_a is the shape function associated with node a and N the number of nodes of the element. If we define a matrix ${}^t_0\partial\mathbf{u_x}$ of displacements derivatives

$$ {}^t_0\partial\mathbf{u_x} = \begin{bmatrix} {}^t_0u_{1,1} & {}^t_0u_{1,2} & {}^t_0u_{1,3} \\ {}^t_0u_{2,1} & {}^t_0u_{2,2} & {}^t_0u_{2,3} \\ {}^t_0u_{3,1} & {}^t_0u_{3,2} & {}^t_0u_{3,3} \end{bmatrix} \tag{5.4} $$

The deformation gradient may be obtained from the displacement derivatives with:

$$ {}^t_0\mathbf{X} = {}^t_0\partial\mathbf{u}\mathbf{x} + \mathbf{I}. \tag{5.5} $$

The Green-Lagrange strain tensor

From the deformation gradient we may obtain the right Cauchy-Green deformation tensor ${}^t_0\mathbf{C}$:

$$ {}^t_0\mathbf{C} = {}^t_0\mathbf{X}^T \, {}^t_0\mathbf{X}, \tag{5.6} $$

and then yields the Green-Lagrange strain tensor ${}^t_0\mathbf{E}$:

$$ {}^t_0\mathbf{E} = \frac{1}{2}({}^t_0\mathbf{C} - \mathbf{I}), \tag{5.7} $$

where $\mathbf{I}$ is the rank 2 identity tensor.

Constitutive equations and evaluation of stress

We know that the stresses result from the deformation of the material and they may be expressed in terms of some measure of this deformation such as the strain. The constitutive equations, which depends on the material under consideration, relate the stresses to the strain. As we have seen in section 2.6.1, hyperelastic materials are a general class of materials in which the constitutive relationship is expressed in the form of a strain energy density function ${}^t_0W$. In such materials, the stresses are obtained by differentiating this energy density function with respect to the appropriate strain measure (that is energically conjugate). In a total Lagrangian framework, the appropriate stress and strain measures are second Piola-Kirchhoff stress ${}^t_0\mathbf{S}$ and Green-Lagrange strain ${}^t_0\mathbf{E}$. Consequently, the stress may be computed from:

$$ {}^t_0\mathbf{S} = \frac{\partial^t_0\mathbf{W}}{\partial^t_0\mathbf{E}}. \tag{5.8} $$

The form of the strain energy function depends on the chosen material. For the TLED algorithm we choose a neo-Hookean material already described section 2.6.5. For a compressible neo-Hookean material, the strain energy density is given by:

$$ {}^t_0W = \frac{1}{2}\mu({}^t_0I_1 - 3 - 2\ln{}^tJ) + \frac{1}{2}\lambda({}^tJ - 1)^2, \tag{5.9} $$

where ${}^t_0I_1$ is the first invariant of the right Cauchy-Green deformation tensor ${}^t_0\mathbf{C}$ given by ${}^t_0I_1 = tr({}^t_0\mathbf{C})$ and ${}^tJ = det({}^t_0\mathbf{X}) = det({}^t_0\mathbf{C})^2$ is the Jacobian. Then, using the chain rule we have:

$$ \frac{\partial^t_0W}{\partial^t_0\mathbf{E}} = \frac{\partial^t_0W}{\partial^t_0\mathbf{C}} \frac{\partial^t_0\mathbf{C}}{\partial^t_0\mathbf{E}}. \tag{5.10} $$

Noting that

$$ \frac{\partial^t_0\mathbf{C}}{\partial^t_0\mathbf{E}} = 2 \quad \text{by (5.7)}, \tag{5.11} $$

Therefore, we may evaluate the stress from:

$$ {}^t_0\mathbf{S} = 2\frac{\partial^t_0W}{\partial^t_0\mathbf{C}}, \tag{5.12} $$

which we can then expressed as the following:

$$ {}^t_0\mathbf{S} = 2\left(\frac{\partial^t_0 W}{\partial^t_0 \mathbf{I_1}}\frac{\partial^t_0 \mathbf{I_1}}{\partial^t_0 \mathbf{E}} + \frac{\partial^t_0 W}{\partial^t J}\frac{\partial^t J}{\partial^t_0 \mathbf{E}}\right). \tag{5.13} $$

It may be shown that this yields:

$$ {}^t_0 S_{ij} = \mu(\delta_{ij} - {}^t_0 C^{-1}_{ij}) + \lambda {}^t J({}^t J - 1){}^t_0 C^{-1}_{ij}. \tag{5.14} $$

Element nodal forces

For a given element at time t, the global nodal force contributions $\tilde{\mathbf{F}}$ due to stresses in the element may be computed from:

$$ {}^t\tilde{\mathbf{F}} = \int_{{}^0V} {}^t_0\mathbf{B}^t_L \, {}^t_0\hat{\mathbf{S}} \, d^0V, \tag{5.15} $$

where ${}^t_0\hat{\mathbf{S}}$ is the vector form of the second Piola-Kirchhoff stress given by:

$$ {}^t_0\hat{\mathbf{S}} = \begin{bmatrix} {}^t_0S_{11} & {}^t_0S_{22} & {}^t_0S_{33} & {}^t_0S_{12} & {}^t_0S_{23} & {}^t_0S_{13} \end{bmatrix}^T, \tag{5.16} $$

${}^t_0\mathbf{B}^T_L$ is the strain-displacement matrix and d^0V is the initial (undeformed) volume of the element. The strain-displacement matrix $\mathbf{B}_L$ relates the strains in an element to the element's nodal displacements. If an implicit analysis is performed, the strain-displacement matrix is used in the assembly of the stiffness matrix. In explicit analyses it is used to compute element nodal force contributions according to (5.15). In geometrically linear analyses, we assume that the displacements are infinitesimally small so that the geometry of an element does not change over time and the strain-displacement matrix is constant. Conversely, because geometrically non-linear analyses (such as carried out by the TLED algorithm) take the change of geometry of the elements into account, the strain-displacement matrix $\mathbf{B}_L = {}^t_0\mathbf{B}_L$ varies. However, the strain-displacement matrix at time t may be computed by transforming a linear matrix ${}^t_0\mathbf{B}_{L0}$ using the deformation gradient. We begin by defining the linear matrix as:

$$ {}^t_0\mathbf{B}_{L0} = \begin{bmatrix} {}_0\mathbf{B}^{(1)}_{L0} & {}_0\mathbf{B}^{(2)}_{L0} & \dots & {}_0\mathbf{B}^{(N)}_{L0} \end{bmatrix}, \tag{5.17} $$

where N is the number of nodes per element and the submatrix ${}_0\mathbf{B}^{(a)}_{L0}$ are given by:

$$ {}_0\mathbf{B}^{(a)}_{L0} = \begin{bmatrix} {}_0h_{a,1} & 0 & 0 \\ 0 & {}_0h_{a,2} & 0 \\ 0 & 0 & {}_0h_{a,3} \\ {}_0h_{a,2} & {}_0h_{a,1} & 0 \\ 0 & {}_0h_{a,3} & {}_0h_{a,2} \\ {}_0h_{a,3} & 0 & {}_0h_{a,1} \end{bmatrix} \quad a = 1, 2, \dots, N. \tag{5.18} $$

The subscripted comma denotes partial differentiation such as:

$$ {}_0h_{a,i} = \frac{\partial h_a}{\partial^0 x_i}. \tag{5.19} $$

The full strain-displacement matrix (accounting for initial displacement effect) is then computed via

$$ {}^{t}_{0}\mathbf{B}_{L}^{(a)} = {}^{t}_{0}\mathbf{B}_{L0}^{(a)}\, {}^{t}_{0}\mathbf{X}^{T}. \tag{5.20} $$

We note that the linear strain-displacement matrix is composed of derivatives of shape functions with respect to the original undeformed coordinates of the body. These derivatives are therefore constant and may be precomputed, affording the total Lagrangian formulation a significant computational advantage.

5.1.3 Computation of node displacements

We recall that the dynamic system of equations describing a deformable solid is the following (see section 3.26):

$$ \mathbf{M}\ddot{\mathbf{U}} + \mathbf{D}\dot{\mathbf{U}} + \mathbf{K}(\mathbf{U}) \cdot \mathbf{U} = \mathbf{R}. \tag{5.21} $$

where $\mathbf{M}$ is a constant mass matrix, $\mathbf{D}$ is a constant damping matrix, $\mathbf{K}^e(\mathbf{U}^e)$ is the stiffness matrix, which is a function of nodal displacements $\mathbf{U}$, and $\mathbf{R}$ are externally applied loads. And we seek the unknown displacements $\mathbf{U}$.

Mass and damping matrices

The mass matrix for the system may be computed by summing contributions from individual elements:

$$ \mathbf{M} = \sum_{e} \mathbf{M}^{e}, \tag{5.22} $$

where $\mathbf{M}^e$ is the mass contribution from element Ω_e given by:

$$ \mathbf{M}^{e} = \int_{{}^{0}V_e} \rho_e \mathbf{H}^{T}\mathbf{H}\, d^{0}V. \tag{5.23} $$

Following the total Lagrangian framework, we observe that the expression is integrated over the undeformed volume and the mass density ρ_e of the undeformed configuration is also used. We then diagonalise the mass matrix by applying the technique of mass lumping as explained in section 3.3.5. Therefore, the mass matrix may be built by directly computing the mass of each element and assigning an equal proportion of this to each of the element's nodes. In other words, if the mass of element Ω_e is m_e then for every node a which is attached to this element, the a^{th} diagonal component M_{aa} of the global mass matrix will receive a contribution of m_e/N where N is the number of nodes per element. The complete matrix is compiled by summing contributions from all elements.

We also employ a Rayleigh damping from which we only consider the mass-proportional component to obtain a diagonal damping matrix. Consequently, $\mathbf{D}$ is computed from:

$$ \mathbf{D} = \alpha\mathbf{M}. \tag{5.24} $$

Gathering of nodal forces

As stated early in this chapter, the stiffness matrix of (5.21) may be obtained from

$$\mathbf{K}(\mathbf{U}).\mathbf{U} = \mathbf{F}(\mathbf{U}) = \sum_e \tilde{\mathbf{F}}^e. \tag{5.25}$$

The previous section detailed how to calculate element nodal forces $\tilde{\mathbf{F}}^e$. For each node we can now add the element force contributions up from all elements attached to this node. Gathering those nodal force contributions allows us to compute the stiffness term of the equilibrium equations ${}^t\mathbf{F}$. Therefore, (5.21) may be re-written as the following:

$$\mathbf{M}\,{}^t\ddot{\mathbf{U}} + \mathbf{D}\,{}^t\dot{\mathbf{U}} + {}^t\mathbf{F} = {}^t\mathbf{R}. \tag{5.26}$$

Explicit time integration

The remaining step is the time integration of (5.26) to find out the expression of ${}^t\mathbf{U}$. For this, we employ the explicit central difference method described page 39. We assume that all displacements, velocities and accelerations for the current time step is known and we seek a formula for computing displacements at time $t + \Delta t$. Let us remind (3.35) for convenience:

$$\left(\frac{\mathbf{M}}{(\Delta t)^2} + \frac{\mathbf{D}}{2\Delta t}\right)\mathbf{U}^{t+\Delta t} = \mathbf{R}^t - \mathbf{F}^t + \frac{2\mathbf{M}}{(\Delta t)^2}\mathbf{U}^t + \left(\frac{\mathbf{D}}{2\Delta t} - \frac{\mathbf{M}}{(\Delta t)^2}\right)\mathbf{U}^{t-\Delta t}. \tag{5.27}$$

By using diagonalised mass and damping matrices, this expression may be rearranged to give a formula for updating displacements component-wise:

$${}^{t+\Delta t}\mathbf{U}_i = \frac{{}^tR_i - {}^tF_i + \frac{2M_{ii}}{\Delta t^2}{}^tU_i + \left(\frac{D_{ii}}{2\Delta t} - \frac{M_{ii}}{\Delta t^2}\right){}^{t-\Delta t}U_i}{\frac{D_{ii}}{2\Delta t} + \frac{M_{ii}}{\Delta t^2}} \tag{5.28}$$

Let us define the following vectors:

$$A_i = \frac{1}{\frac{D_{ii}}{2\Delta t} + \frac{M_{ii}}{\Delta t^2}} \tag{5.29}$$

$$B_i = \frac{\frac{2M_{ii}}{\Delta t^2}}{\frac{D_{ii}}{2\Delta t} + \frac{M_{ii}}{\Delta t^2}} = \frac{2M_{ii}}{\Delta t^2}A_i \tag{5.30}$$

$$C_i = \frac{\frac{D_{ii}}{2\Delta t} - \frac{M_{ii}}{\Delta t^2}}{\frac{D_{ii}}{2\Delta t} + \frac{M_{ii}}{\Delta t^2}} = \frac{D_{ii}}{2\Delta t}A_i - \frac{B_i}{2}, \tag{5.31}$$

Equation (5.28) may be rewritten as:

$${}^{t+\Delta t}\mathbf{U}_i = A_i\left({}^tR_i - {}^tF_i\right) + B_i\,{}^tU_i + C_i\,{}^{t-\Delta t}U_i. \tag{5.32}$$

It is worth noting that the coefficient vectors $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ can be precomputed.

Of crucial importance in explicit analyses is the restriction on time step sizes imposed by stability constraints. As mentioned, the central difference method is only

stable if $\Delta t < \Delta t_{cr}$. Essentially, stability requires that the time step be not larger than the time required for a dilatational wave to propagate through the smallest element. In linear analyses this translates into the following formula for Δt_{cr}:

$$\Delta t_{cr} = \frac{L_e}{c}, \tag{5.33}$$

where L_e is the smallest characteristic element length (roughly interpreted as the smallest edge length) in the assembly, and c is the dilatational wave speed of the material, given by

$$c = \sqrt{\frac{E(1-\nu)}{\rho(1+\nu)(1-2\nu)}}, \tag{5.34}$$

where E is Young's modulus and ν is Poisson's ratio. If nonlinear analyses (either geometric or material) are conducted, a somewhat smaller time step must be employed since c will change with deformation, and in particular will generally increase. Additionally, the presence of damping necessitates a further decrease in the limit. Nonetheless, (5.34) may be used to estimate Δt_{cr}.

Some important observations arise from (5.33) and (5.34). Firstly, since the stiffness of soft tissues is very much smaller than that of common engineering materials ($E \approx 3 \times 10^3$ Pa for brain tissue versus $\approx 2 \times 10^{11}$ Pa for steel), the allowable time step is much larger for analysis of the former. This is a key reason for the expediency of explicit time integration for analysis of soft tissues. Secondly, soft tissues are generally considered to be incompressible, leading to $\nu \approx 0.49$ commonly being employed. However, if this value can be relaxed even further, significant increases in Δt_{cr} are possible. As an example, for a given L_e a material with $E = 3\,000$ Pa and $\nu = 0.45$ allows a time step of more than double that of a material with $\nu = 0.49$. Of course, lowering the Poisson ratio introduces inaccuracies, but for some applications (interactive medical simulation for instance) it may be that this is acceptable. Finally, it must be highlighted that Δt_{cr} is proportional to L_e. If the element size is decreased (by using a finer mesh, or by analysing very small objects), inexorably Δt_{cr} is also reduced.

5.2 Anisotropic and viscoelastic constitutive equations

5.2.1 An extension to the TLED algorithm

An attractive feature of explicit analyses is the relative ease with which arbitrarily complicated constitutive models may be incorporated. This arises from the fact that element stresses are computed directly from strains in the course of the procedure. Additionally, there is no requirement for computation of tangent matrices as in implicit or quasi-static procedures, since there is no involvement of Newtonian iterations. While the developments above incorporated a hyperelastic constitutive formulation (neo-Hookean), thus accommodating non-linearity of the strain-related tissue stress response, the formulation was intentionally simple nonetheless since

the focus was on validation of the computational framework. Two further key features of the response of most biological tissues are time- (and rate-)dependence and anisotropy (Fung, 1993).

Time-dependence manifests itself in many aspects of the mechanical response. Soft tissues under constant load generally exhibit creep, while those under constant deformation exhibit stress relaxation. Additionally, most tissues appear stiffer at higher loading velocities. In particular, so-called visco-hyperelastic models based on strain energy functions with time-dependent parameters have been shown to reproduce both the time-dependent and large strain aspects of the response (Miller and Chinzei, 1997; Miller, 2000; Nava et al., 2008).

Anisotropic mechanical response may arise, for example, from the presence of a highly organised microstructure such as those of connective tissues. These are predominantly composed of collagen or elastin fibres embedded in an amorphous matrix (Fung, 1993), and may be considered as fibre reinforced composites in some cases. Alternatively, the presence of vasculature and other functional components means even non-load bearing organs may exhibit directional dependence (Picinbono et al., 2003; Prange and Margulies, 2002). Whereas isotropic constitutive models may be formulated in terms of the usual principal strain invariants (as used in (5.9) for a neo-Hookean material), directional dependence requires inclusion of so-called pseudo-invariants of strain and material direction.

In order to model such soft-tissue features with the explicit TLED formulation without significant performance penalties, an efficient constitutive update procedure involving time integration of the relevant hereditary equations is required. We address this problem by presenting a procedure similar to that developed by Poon and Ahmad (1998) for analysis of anisotropic linear viscoelastic models. This was also adapted by Taylor et al. (2007b) for solution of their fibre composite-based microstructural model. In these cases the rate independent responses were based on linear elasticity, rather than a hyperelastic formulation suitable for large deformations. In the present work we begin with a class of anisotropic visco-hyperelastic models, and develop a constitutive update procedure for explicit analyses based on these.

5.2.2 Visco-hyperelasticity

Large recoverable deformations and the time- and rate-dependence of the mechanical response of soft-tissues have led to the formulation of visco-hyperelastic constitutive models in which an underlying hyperelastic formulation is augmented by time dependent (viscoelastic) material parameters. Models of this type are well known in the continuum mechanics community, and provide a kinematically consistent basis for modelling non-linear materials at large deformations.

For such materials the constitutive response is defined in terms of a time-dependent Helmholtz free energy (strain energy) function $\hat{\Psi}$, expressed in the form

of a convolution integral:

$$\hat{\Psi}(\Psi, t) = \int_0^t \alpha(t-s)\frac{\partial \Psi}{\partial s} ds, \tag{5.35}$$

where t is time and Ψ is the underlying hyperelastic strain energy function. The relaxation functions $\alpha(t)$ commonly assume the form of a Prony series:

$$\alpha(t) = \alpha_\infty + \sum_{i=1}^{N} \alpha_i e^{-t/\tau_i}, \tag{5.36}$$

where α_∞, α_i and τ_i are positive real constants. Such forms for the relaxation functions have a physical interpretation, namely that of a generalised Maxwell model (Holzapfel, 1996). If we impose the condition

$$\left(\alpha_\infty + \sum_{i=1}^{N} \alpha_i\right) = 1 \tag{5.37}$$

we may rewrite(5.36) as

$$\alpha(t) = 1 - \sum_{i=1}^{N} \alpha_i (1 - e^{-t/\tau_i}), \tag{5.38}$$

which will be of use in subsequent sections.

The required stress $\mathbf{S}$ may be obtained via differentiation with respect to strain:

$$\mathbf{S} = 2\frac{\partial \hat{\Psi}(\Psi, t)}{\partial \mathbf{C}} = \int_0^t \alpha(t-s)\left(2\frac{\partial}{\partial s}\frac{\partial \Psi}{\partial \mathbf{C}}\right) ds = \int_0^t \alpha(t-s)\frac{\partial \mathbf{\Phi}}{\partial s} ds, \tag{5.39}$$

where $\mathbf{C}$ is the right Cauchy-Green deformation tensor, and we have introduced $\mathbf{\Phi} \stackrel{\text{def}}{=} 2\partial\Psi/\partial\mathbf{C}$ as the instantaneous hyperelastic stress response.

Models of this form have been presented by Miller and co-workers for analysis of brain tissue (Miller and Chinzei, 1997, 2002) and of liver and kidney (Miller, 2000). They were shown to model the tissue responses to compressive (and in the case of brain, tensile) loading at strain rates varying over two orders of magnitude very well. We next consider evaluation of the hyperelastic stress response $\mathbf{\Phi}$.

5.2.3 Hyperelastic response

We firstly consider the case of isotropic materials, from which anisotropic formulations follow. We then consider the standard cases of transverse isotropy and orthotropy, which may be viewed as arising from, for example, the presence of mutually orthogonal reinforcing fibre phases. In considering these cases we encompass constitutive equations which have been proposed for a wide variety of biological tissues.

Isotropic materials

For isotropic materials the strain energy is a function of strain only, hence $\Psi = \Psi(\mathbf{C})$. In this work we consider strain energy functions with separated isochoric (volume preserving) and volumetric components (Holzapfel et al., 2000):

$$\Psi(\mathbf{C}) = \Psi^{iso}(\bar{\mathbf{C}}) + \Psi^{vol}(J) = \Psi^{iso}(\bar{I}_1, \bar{I}_2) + \Psi^{vol}(J), \tag{5.40}$$

where J is the Jacobian determinant, $\bar{\mathbf{C}} = J^{2/3}\mathbf{C}$ is the modified right Cauchy-Green deformation, and $\bar{I}_1 = tr\bar{\mathbf{C}}$ and $\bar{I}_2 = \left[(tr\bar{\mathbf{C}})^2 - tr(\bar{\mathbf{C}}^2)\right]/2$ are invariants of $\bar{\mathbf{C}}$. The hyperelastic stress $\mathbf{\Phi}$ then also consists of isochoric and volumetric components:

$$\mathbf{\Phi} = \mathbf{\Phi}^{iso} + \mathbf{\Phi}^{vol}, \tag{5.41}$$

where

$$\mathbf{\Phi}^{vol} = 2\frac{\partial \Psi^{vol}(J)}{\partial \mathbf{C}} = J\frac{d\Psi^{vol}(J)}{dJ}\mathbf{C}^{-1}, \tag{5.42}$$

and

$$\mathbf{\Phi}^{iso} = 2\frac{\partial \Psi^{iso}(\bar{\mathbf{C}})}{\partial \mathbf{C}} = J^{-2/3}\mathrm{Dev}\bar{\mathbf{\Phi}}, \tag{5.43}$$

where $\mathrm{Dev}(\bullet) = (\bullet) - (1/3)[(\bullet) : \mathbf{C}]\mathbf{C}^{-1}$ is the referential configuration deviatoric operator for a second order tensor (Holzapfel et al., 2000), and

$$\bar{\mathbf{\Phi}} = 2\frac{\partial \Psi^{iso}(\bar{\mathbf{C}})}{\partial \bar{\mathbf{C}}} = \bar{\gamma}_1\mathbf{I} + \bar{\gamma}_2\bar{\mathbf{C}}, \tag{5.44}$$

with

$$\bar{\gamma}_1 = 2\left(\frac{\partial \Psi^{iso}}{\partial \bar{I}_1} + \bar{I}_1\frac{\partial \Psi^{iso}}{\partial \bar{I}_2}\right) \quad \text{and} \quad \bar{\gamma}_2 = -2\frac{\partial \Psi^{iso}}{\partial \bar{I}_2}. \tag{5.45}$$

Transversely isotropic materials

Transversely isotropic materials are characterised by a single preferred direction $\mathbf{a}_0$ in the reference configuration; the mechanical response is isotropic in the plane orthogonal to this direction. The strain energy is then a function of both $\mathbf{C}$ and a structure tensor $\mathbf{A}_0 \overset{\text{def}}{=} \mathbf{a}_0 \otimes \mathbf{a}_0$, where $\otimes$ denotes a tensor product. Analogous to the isotropic case, we consider strain energy functions of the form

$$\Psi(\mathbf{C}, \mathbf{A}_0) = \Psi^{iso}(\bar{I}_1, \bar{I}_2, \bar{I}_4, \bar{I}_5) + \Psi^{vol}(J), \tag{5.46}$$

where $\bar{I}_4 = \mathbf{a}_0 \cdot \bar{\mathbf{C}}\mathbf{a}_0$ and $\bar{I}_5 = \mathbf{a}_0 \cdot \bar{\mathbf{C}}^2\mathbf{a}_0$ are pseudo-invariants of $\bar{\mathbf{C}}$ and $\mathbf{A}_0$.

As can be seen transversely isotropic strain energy functions (5.46) differ from isotropic ones (5.40) only in the form of the isochoric term. Therefore the volumetric and isochoric stresses $\mathbf{\Phi}^{vol}$ and $\mathbf{\Phi}^{iso}$ remain as in (5.42) and (5.43) but with $\bar{\mathbf{\Phi}}$ given by

$$\bar{\mathbf{\Phi}} = \bar{\gamma}_1\mathbf{I} + \bar{\gamma}_2\bar{\mathbf{C}} + \bar{\gamma}_4\mathbf{A}_0 + \bar{\gamma}_5(\mathbf{a}_0 \otimes \bar{\mathbf{C}}\mathbf{a}_0 + \bar{\mathbf{C}}\mathbf{a}_0 \otimes \mathbf{a}_0), \tag{5.47}$$

with

$$\bar{\gamma}_a = 2\frac{\partial \Psi^{iso}}{\partial \bar{I}_a}, \quad a = 4, 5. \tag{5.48}$$

Orthotropic materials

Orthotropic materials are characterised by three mutually orthogonal preferred directions, which we identify with unit vectors $\mathbf{a}_0$ and $\mathbf{b}_0$ (and corresponding structure tensors $\mathbf{A}_0$ and $\mathbf{B}_0$), in the reference configuration. We need specify only two vectors, since the direction orthogonal to these naturally emerges as a preferred direction also. Orthotropic strain energy functions are then of the form

$$\Psi(\mathbf{C}, \mathbf{A}_0, \mathbf{B}_0) = \Psi^{iso}(\bar{I}_1, \bar{I}_2, \bar{I}_4, \bar{I}_5, \bar{I}_6, \bar{I}_7) + \Psi^{vol}(J), \tag{5.49}$$

where $\bar{I}_6 = \mathbf{b}_0 \cdot \bar{\mathbf{C}}\mathbf{b}_0$ and $\bar{I}_7 = \mathbf{b}_0 \cdot \bar{\mathbf{C}}^2\mathbf{b}_0$ are pseudo-invariants of $\bar{\mathbf{C}}$ and $\mathbf{B}_0$. In a similar manner to the transversely isotropic case we obtain $\mathbf{\Phi}^{vol}$ and $\mathbf{\Phi}^{iso}$ from (5.42) and (5.43), with $\bar{\mathbf{\Phi}}$ now given by

$$\begin{aligned}\bar{\mathbf{\Phi}} = \bar{\gamma}_1\mathbf{I} + \bar{\gamma}_2\bar{\mathbf{C}} + \bar{\gamma}_4\mathbf{A}_0 + \bar{\gamma}_5(\mathbf{a}_0 \otimes \bar{\mathbf{C}}\mathbf{a}_0 + \bar{\mathbf{C}}\mathbf{a}_0 \otimes \mathbf{a}_0) \\ + \bar{\gamma}_6\mathbf{B}_0 + \bar{\gamma}_7(\mathbf{b}_0 \otimes \bar{\mathbf{C}}\mathbf{b}_0 + \bar{\mathbf{C}}\mathbf{b}_0 \otimes \mathbf{b}_0)\end{aligned} \tag{5.50}$$

where

$$\bar{\gamma}_a = 2\frac{\partial \Psi^{iso}}{\partial \bar{I}_a}, \quad a = 6, 7. \tag{5.51}$$

5.2.4 Recapitulation

For a visco-hyperelastic material stress may be obtained from (5.39). This form is general in the sense that any underlying hyperelastic response may be used, including the anisotropic formulations described (5.40), supplemented with (5.42), (5.43), and (5.44) is the general form of an isotropic hyperelastic stress response, defined in terms of invariants. Transversely isotropic or orthotropic models may be produced by substituting (5.44) for (5.47) or (5.50), respectively. The specification of particular forms of Ψ^{vol} and Ψ^{iso} would be motivated by the particular tissue/material under analysis, and may stem from phenomenological or microstructural considerations. Finally we note that for separated isochoric and volumetric hyperelastic functions as used here, viscoelastic terms may be applied to either or both independently.

5.3 Constitutive update procedure for explicit analyses

Use of the above visco-hyperelastic models within the TLED algorithm (or any other explicit dynamic finite element procedure) requires a constitutive update scheme involving time integration of (5.39).

5.3.1 Stress update equations

We proceed by restating (5.39) and using (5.38):

$$\mathbf{S} = \int_0^t \left[1 - \sum_{i=1}^{N} \alpha_i (1 - e^{(s-t)/\tau_i})\right] \frac{\partial \mathbf{\Phi}}{\partial s} ds. \tag{5.52}$$

This may be separated into rate-dependent and -independent terms as

$$\mathbf{S} = \mathbf{\Phi} - \sum_{i=1}^{N} \mathbf{\Upsilon}_i, \tag{5.53}$$

where

$$\mathbf{\Upsilon}_i = \int_0^t \alpha_i \left(1 - e^{(s-t)/\tau_i}\right) \frac{\partial \mathbf{\Phi}}{\partial s} ds, \quad i \in [1, N] \tag{5.54}$$

are rate-dependent terms associated with each term in the Prony series.

In an incremental analysis we require the stress at the current increment given the deformation state and history of the material. Adding superscripts to indicate time increments the stress may be updated using

$$\mathbf{S}^n = \mathbf{\Phi}^n - \sum_{i=1}^{N} \mathbf{\Upsilon}_i^n. \tag{5.55}$$

The instantaneous terms $\mathbf{\Phi}^n$ may be computed directly from the (known) current deformation $\mathbf{C}^n$. The main difficulty is then computation of the incremental rate-dependent terms $\mathbf{\Upsilon}_i^n$. Following Poon and Ahmad (1998), our strategy is to maintain each $\mathbf{\Upsilon}_i^n$ as a separate state variable to be updated at each increment also.

5.3.2 State variable update equations

Our approach is to convert the integral equation (5.54) into a rate form which may then be numerically integrated to produce an incremental update formula for $\mathbf{\Upsilon}_i^n$. We note that (5.54) is of the form

$$y(t) = \int_0^t f(t, s) ds. \tag{5.56}$$

Poon and Ahmad (1998) provide the following formula for differentiating such equations with respect to t:

$$\dot{y} = f(t, t) + \int_0^t \dot{f}(t, s) ds. \tag{5.57}$$

Applying this formula to (5.54) we obtain

$$\begin{aligned} f(t, t) &= \alpha_i \left(1 - e^{(t-t)/\tau_i}\right) \frac{\partial \mathbf{\Phi}}{\partial s} \\ &= 0 \end{aligned} \tag{5.58}$$

and

$$\begin{aligned} \dot{f}(t, s) &= \frac{d}{dt}\left[\alpha_i \left(1 - e^{(s-t)/\tau_i}\right) \frac{\partial \mathbf{\Phi}}{\partial s}\right] \\ &= \alpha_i e^{(s-t)/\tau_i} \frac{\partial \mathbf{\Phi}}{\partial s} \frac{1}{\tau_i}, \end{aligned} \tag{5.59}$$

leading to the required rate form for $\boldsymbol{\Upsilon}_i$:

$$\begin{aligned}\dot{\boldsymbol{\Upsilon}}_i &= \frac{1}{\tau_i}\int_0^t \alpha_i e^{(s-t)/\tau_i}\frac{\partial \boldsymbol{\Phi}}{\partial s}ds \\ &= \frac{1}{\tau_i}(\alpha_i \boldsymbol{\Phi} - \boldsymbol{\Upsilon}_i).\end{aligned} \tag{5.60}$$

Equation (5.60) may be integrated using a convenient numerical method. In particular the unconditionally stable backward Euler method suggested by Poon and Ahmad (1998) yields the following formula for $\boldsymbol{\Upsilon}_i^n$:

$$\begin{aligned}\boldsymbol{\Upsilon}_i^n &= \left(\frac{\alpha_i \boldsymbol{\Phi}^n}{\tau_i} + \frac{\boldsymbol{\Upsilon}_i^{n-1}}{\Delta t}\right) / \left(\frac{1}{\Delta t} + \frac{1}{\tau_i}\right) \\ &= A\boldsymbol{\Phi}^n + B\boldsymbol{\Upsilon}_i^{n-1},\end{aligned} \tag{5.61}$$

where Δt is the time step size, and $A = \Delta t \alpha_i/(\Delta t + \tau_i)$ and $B = \tau_i/(\Delta t + \tau_i)$ are constant coefficients.

5.3.3 Summary

The constitutive update procedure consists of

1. Updating state variables (one for each Prony term) via (5.61).
2. Updating stresses via (5.55).

Assuming a constant time step size Δt, the coefficients A and B may be precomputed.

Chapter 6

GPU implementation of TLED

The previous chapter introduced the TLED algorithm in details as well as enhancements with a viscoelastic and anisotropic formulation. If the TLED algorithm had already been implemented on GPU by Taylor et al. (2007a), restrictions due to using a graphics-based API and the limited hardware possibilities of GPUs necessitated reformulation of the force summation in their implementation. With the release by NVIDIA of a new graphics card architecture along with a new and more flexible non-graphics API specially designed for general computation on GPU, we decided to investigate the re-implementation of the TLED for this new architecture with the aim to achieve a more straightforward and efficient implementation. The aim was also to enhance the GPU-based implementation of the TLED with a viscoelastic and anisotropic formulation. This chapter will describe the implementation of this non-linear, viscoelastic and anisotropic finite element method algorithm in the open source framework SOFA. By providing an efficient and accurate non-linear FEM for soft tissue modelling to worldwide researchers, we thus hope to assist in enhancing the realism of medical simulators. This work has led to three publications (Comas et al., 2008; Taylor et al., 2008a, 2009).

6.1 Summary of the TLED formulation

A complete description of the TLED algorithm for soft tissue simulation was given in the previous chapter. However, let us remind the main steps of the algorithm before tackling the implementation. Briefly, the algorithm consists of a precomputation phase in which element shape function derivatives $\partial\mathbf{h}$ (and other quantities) and the system mass matrix $\mathbf{M}$ are calculated, followed by a time-loop in which incremental solutions for the node displacements $\mathbf{U}$ are found. During each step of the time-loop we:

1. Apply loads (displacements and/or forces) and boundary conditions to relevant nodal degrees of freedom
2. For each element compute
   - **(a)** deformation gradient $\mathbf{X}$ and right Cauchy-Green deformation tensor $\mathbf{C}$
   - **(b)** linear strain-displacement matrix $\mathbf{B}_L$
   - **(c)** second Piola-Kirchhoff stress $\mathbf{S}$
   - **(d)** element nodal forces $\tilde{\mathbf{F}}$, and add these forces to the total nodal forces $\mathbf{F}$
3. For each node compute new displacements $\mathbf{U}$ using the central difference method.

The nodal force contributions $\tilde{\mathbf{F}}$ from each element are obtained

$$\tilde{\mathbf{F}} = \int_{^0V} \mathbf{B}_L^t \, \hat{\mathbf{S}} \, d^0V, \tag{6.1}$$

where ^0V is the initial volume of the element and $\hat{\mathbf{S}}$ is the vector form of the stress tensor $\mathbf{S}$. This integral is generally evaluated numerically, for example using Gaussian quadrature. For reduced integration 8-node hexahedral elements we obtain

$$\tilde{\mathbf{F}} = 8 \, \mathbf{B}_L^t \, \hat{\mathbf{S}} \, J \tag{6.2}$$

where J is the Jacobian determinant. For 4-node tetrahedral elements we obtain

$$\tilde{\mathbf{F}} = V \, \mathbf{B}_L^t \, \hat{\mathbf{S}}. \tag{6.3}$$

The above equations make no assumption concerning the constitutive model employed. The deformation state $\mathbf{F}$ in each element is known, allowing stresses $\mathbf{S}$ to be computed from any valid constitutive equation.

6.2 General-purpose computation on GPU

6.2.1 Goal and motivation

Graphics processing units (GPU) functionality has, traditionally, been very limited. In fact, for many years the GPU was only used to accelerate certain parts of the

graphics pipeline. A GPU is essentially a special purpose hardware designed to accelerate each stage of the geometry pipeline, the process of matching image data or a computer model to the pixels on the computer's screen. Initially, GPU could only run two kinds of program: vertex and pixel shaders. Vertex shaders are run once for each vertex given to the graphics processor. The purpose is to transform each vertex's 3D position in virtual space to the 2D coordinate at which it appears on the screen. Vertex shaders can manipulate properties such as position, color, texture coordinate and normal vector. Pixel shaders are functions that compute color and other attributes of each pixel. They range from always outputting the same color, to applying a lighting value, to doing shadows, specular highlights or translucency for instance.

Eventually, vertex and pixel shaders became programmable to enable game programmers to generate even more realistic effects. Programmable pixel shaders allow the programmer to substitute, for example, a lighting model other than those provided by default by the graphics card. Shaders have enabled graphics programmers to create lens effects, displacement mapping and depth of field. This evolution of GPU's hardware and the increasing programmable capability naturally lead to use GPUs for non-graphics applications. The term GPGPU (General-purpose computation on graphics processing units) was coined by Mark Harris in 2002 when he recognised an early trend of using GPUs for non-graphics applications. However, capabilities of GPU's were still fairly limited for non-graphics applications at this time.

Things dramatically changed in 2007 when the two types of shaders were unified. While early shader models used very different instruction sets for vertex and pixel shaders, unified shader models have almost the same capabilities. An unified shader architecture allows more flexible use of the graphics rendering hardware. The computing units of the GPU can run vertex or pixel shaders according to work loads. Along with unified shader models, a new type of shader was created: geometry shaders. They are executed after vertex shaders and can generate new graphics primitives, such as points, lines and triangles.

GPUs may be seen as high-performance many-core processors that can be used to accelerate a wide range of applications. An example (given by Sanford Russell from NVIDIA) to think about to illustrate the difference between a traditional CPU and a GPU is this: if you were looking for a word in a book, and handed the task to a CPU, it would start at page 1 and read it all the way to the end, because it's a serial processor. It would be fast, but would take time because it has to go in order. A GPU, which is a parallel processor, would tear the book into a thousand pieces and read it all at the same time. Even if each individual word is read more slowly, the book may be read in its entirety quicker, because words are read simultaneously.

In addition to an increasing programmability, it is easy to understand why the development of GPGPU is soaring. GPU is now used in imaging, finance, signal processing, simulation, audio and video processing, astronomy, weather forecasting, molecular modelling, cryptography, quantum mechanics and many other fields. A

fairly recent review of GPGPU algorithms may be found in Owens et al. (2007). This is therefore not surprising that research has been carried out to leverage the power of GPUs to solve partial differential equations of continuum mechanics via the computationally expensive finite element method. Indeed, we have already insisted on the strong time constraints demanded by the field of medical simulation and the path towards GPGPU is natural. A non-exhaustive review of algorithms implemented on GPU in the context of medical simulation was given chapter 4 (page 67).

Before introducing our own GPU implementation of the TLED formulation, we will discuss the different GPU programming languages at our disposal.

6.2.2 Programming languages for GPUs

Interface with a GPU has traditionally been via a graphic application programming interface (API) such as OpenGL or DirectX. OpenGL is an open standard API that provides a number of functions for the rendering of 2D and 3D graphics and is available on most modern operating systems including but not limited to Windows, Mac OS X and Linux. It was initially designed by Silicon Graphics Inc. and now managed by the technology consortium Khronos group. DirectX is a proprietary API that provides functions to render three dimensional graphics, and uses hardware acceleration if it is available on the graphics card. It was designed by Microsoft for use on the Windows platform.

Because OpenGL and DirectX are both low-level APIs and include many irrelevant functionality from a GPGPU perspective, higher level languages were designed to give developers more direct control of the graphics pipeline without having to use assembly language or hardware-specific languages.

Graphics API

OpenGL (version 1.5 and newer) provides shader language based on the C programming language called OpenGL shading language (GLSL). In the DirectX API (DirectX 9 and newer), shaders are programmed with the high level shader language (HLSL). Cg is another high-level shading language developed by NVIDIA in close collaboration with Microsoft and it is very similar to Microsoft's HLSL (Mark et al., 2003). However, the Cg compiler is not specific to a graphics API and can output both OpenGL and DirectX shader programs.

These languages all share basic data types (float, integer, bool etc.) and also feature vector and matrix data types that are based on the basic data types (like float3 or float4x4 in Cg and vec3 or mat4 in GLSL) and standard library functions are provided (vector and matrix multiplication, cross product etc.). In addition, they support loops and branching like if/else, while, and for. However, because these languages are just an abstraction on the top of graphics API, there are significant restrictions on what they can do. Specifically, shader programs are unable to determine the destination of their outputs. They merely operate on fragment data (color, texture coordinates, normal etc.) and return the modified fragment at

the same location. Each shader is also executed independently of its neighbours since no communication is allowed between processors. Note that while this was made possible on recent hardware, these graphics API do not take advantage of this feature and therefore this statement remains valid.

Memory storage is allowed through the use of textures, which essentially are structured collections of floating point values arranged as arrays. Shaders can read textyres from any location and as many time as they like. As such it is a concept very similar to regular arrays. However, the individual components of textures consist of up to four floating point values in the form of a vector. This comes from the fact that graphics processing units were initially designed to manipulate color images, so each component of a texture is in fact a vector holding the red, green, blue and alpha (transparency) values for the given pixel. Textures are therefore optimised for retrieving up to four floating point values in a single read.

When it comes to writing to GPU memory though, things are a bit more limited. Indeed, in contrast to texture reads, shaders can only write once into textures (at the end of the set of instructions). Morevoer, as stated earlier, shaders can only write in predefined locations corresponding to their potential screen location. Therefore, graphics API does not allow writing to random memory locations (this feature is called *scattering*). Nethertheless, shaders can have multiple render targets. The write locations within each texture are still fixed though.

Non-graphics API

Eventually, non-graphics API were released to support the development of GPGPU. The two major graphics card manufacturers first released their own API. While the objective of hiding graphics aspects is the same, the approaches of NVIDIA and ATI (now owned by AMD) are quite different.

In November 2006, NVIDIA released CUDA which consists of C language extensions. CUDA is fairly high level. Programs (called kernels) may be coded as C functions and invoked directly without going through a render pass as with graphics APIs. When called, kernels are executed N times in parallel by N threads as opposed to only once like regular C functions. Memory management functions analogous to standard C are also provided (memcpy, free etc.).

At the same time, ATI released CTM released a software development layer aimed for low-level access (assembly-style language). Memory management functions similar to graphics API are also provided. A year later, in December 2007, ATI released Stream SDK and added a new high-level language derived from C, called ATI Brook+ which later evolved to ATI CAL (Compute Abstraction Layer). In both cases, these APIs work solely on each manufacturer's devices, CUDA with NVIDIA cards and Stream SDK on ATI cards.

OpenCL is a more general framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, and other processors. It is not specific to a single type of hardware. OpenCL was initially developed by Apple and refined into an initial proposal in collaboration AMD (ATI), IBM, Intel, and

NVIDIA. Apple submitted this initial proposal to the Khronos Group and the first release occurred in December 2008. AMD decided to support OpenCL instead of the now deprecated CTM in its Stream SDK.

The latest API to be released was DirectCompute from Microsoft. DirectCompute was released as part of DirectX11 in October 2009 and allows access to general-purpose computing with a fairly low level language.

The choice of API

The first GPU implementation of the TLED algorithm was carried out by Taylor et al. (2007a, 2008b). They used the Cg language and had to reformulate the TLED algorithm presented to accomodate with the limited possibilities of this graphics API. Indeed, the major difference between this parallel implementation and a serial one is the writing of element nodal forces to memory at the end of a first kernel and their subsequent retrieval and summation during a second kernel. In a serial implementation nodal force contributions would most likely be added to a global sum as they are computed, rather than stored and summed in a second loop. But as stated earlier, this graphics API does not allow scattering, that is writing to random memory locations. Because of this limitation, element nodal forces cannot be directly added to the total nodal forces. Instead, the authors had to reformulate force summation as a gather operation in their implementation. They reported significant solution speed gains up to 16.8× over the CPU implementation. Thus, they were able to compute a simple cube model with up to 16 000 tetrahedral elements in real-time.

A major improvement was introduced with the release of CUDA. In theory, CUDA allows scattered writes and this new feature could dramatically change the GPU implementation of the TLED as the reformulation of the scatter as a gather would no longer be useful. A single kernel would be sufficient and an increase in performance is expected. Along with a more readable and simple code, this is the main feature that made us choose the CUDA API to reimplement the TLED algorithm.

In addition, we decided to develop our CUDA-based re-implementation of the TLED algorithm within the international and open source framework SOFA. We will show that this integration has a very limited cost in terms of performance by comparing the SOFA version with a standalone implementation. By providing an efficient and accurate non-linear FEM for soft tissue modelling to worldwide researchers, we thus hope to assist in enhancing the realism of medical simulators.

6.3 Implementation into SOFA

6.3.1 SOFA, an open source simulation framework

Objectives

The multi-disciplinary aspect of medical simulation requires the integration within a single environment of solutions in areas as diverse as visualisation, biomechanical modelling, haptic feedback and contact modelling. Although their interaction is essential to design a realistic simulator, only few teams have the sufficient resources to build such frameworks. This diversity of problems creates challenges for researchers to advance specific areas, and leads rather often to duplication of effort. The open source SOFA framework (Allard et al., 2007) was created to overcome this issue by providing researchers with an advanced software architecture that facilitates the development of new algorithms and simulators. SOFA has been mostly developed by INRIA (the French national institute for research in computer science and control) and CIMIT (Center for Integration of Medicine and Innovative Technology) and is primarily targeted at real-time simulation with an emphasis on medical simulation. SOFA is highly modular and flexible: it allows independently developed algorithms to interact together within a common simulation while minimising the development time required for integration. The overall goal is to develop a flexible framework while minimising the impact of this flexibility on the computation overhead. To achieve these objectives, SOFA proposes a new architecture that implements a series of concepts described below.

SOFA architecture

High-level modularity. The SOFA architecture relies on the innovative notion of multi-model representation where an object is explicitly decomposed into various representations: Behaviour Model, Collision Model, Collision Response Model, Visual Model and Haptic Model. Each representation can then be optimised for a particular task (biomechanics, collision detection, visualisation, haptics) while at the same time improving interoperability by creating a clear separation between the functional aspects of the simulation components. These representations are then connected together via a mechanism called *mapping*. Various mapping functions can be defined, and each mapping associates a set of primitives of a representation to a set of primitives in the other representation (Fig. 6.1). For instance, a mapping can connect degrees of freedom in a Behaviour Model to vertices in a Visual Model.

Finer level modularity. In order to easily compare algorithms within SOFA, more flexibility was added to the Behaviour Model by introducing an even finer level of granularity. A series of generic primitives common to most physics-based simulations have been defined: DoF, Mass, Force Field and Solver. The DoF component describes the degrees of freedom, and their derivatives, of the object. The Mass component represents its mass. The Force Field describes both internal and

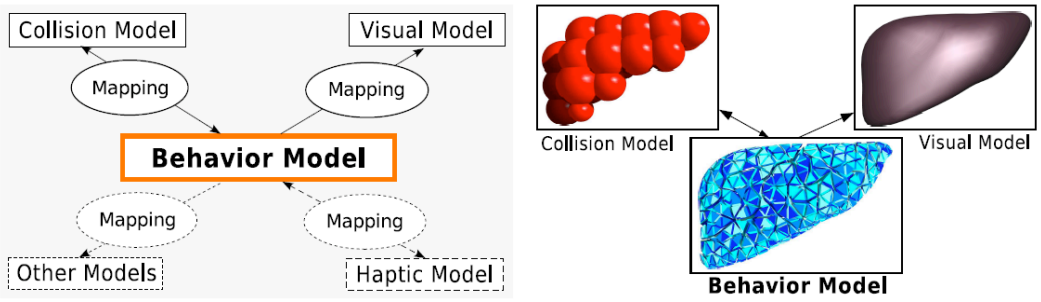


Figure 6.1: Multi-model representation in SOFA. **Left:** a Behaviour Model controls the other representations via a series of mappings. **Right:** examples of representations with a liver model.

external forces that can be applied to this object. The Solver component handles the time step integration, i.e. advancing the state of the system from time t to time $t + \Delta t$.

Scene-graph. Finally, another key aspect of SOFA is the use of a scene-graph to organise and process the elements of a simulation. Each component is attached to a node of a tree structure. This simple structure makes it easy to visit all or a subset of the components in a scene, and dependencies between components are handled by retrieving sibling components attached to the same node. During the simulation loop, most computations can be expressed as a traversal of the scene-graph. For instance, at each time step, the simulation state is updated by processing all Solver components, which will then forward requests to the appropriate components by recursively sending actions within its sub-tree.

These different functionalities and levels of abstraction allow the user to switch from one component to another by simply editing an XML file, without having to recompile. In particular this permits testing of different computational models of soft tissue deformation, and to assess the pros and cons of various algorithms within the same context.

6.3.2 CUDA description

GPUs achieve a high floating point capacity by distributing computation across a high number of parallel execution threads. They perform optimally as single instruction, multiple data devices. CUDA is a relatively new C API for compatible NVIDIA GPUs. CUDA organises threads in two hierarchical levels: blocks, which are groups of threads executed on one of the GPU's multiprocessors, and grids, which are groups of blocks launched concurrently on the device, and which all execute the same kernel. Figure 6.2 represents this thread organisation.

CUDA allows developers to specify the number of threads per block in each execution (the so-called execution configuration), effectively defining the distribution of computational load across all processors. For a given kernel the block dimensions

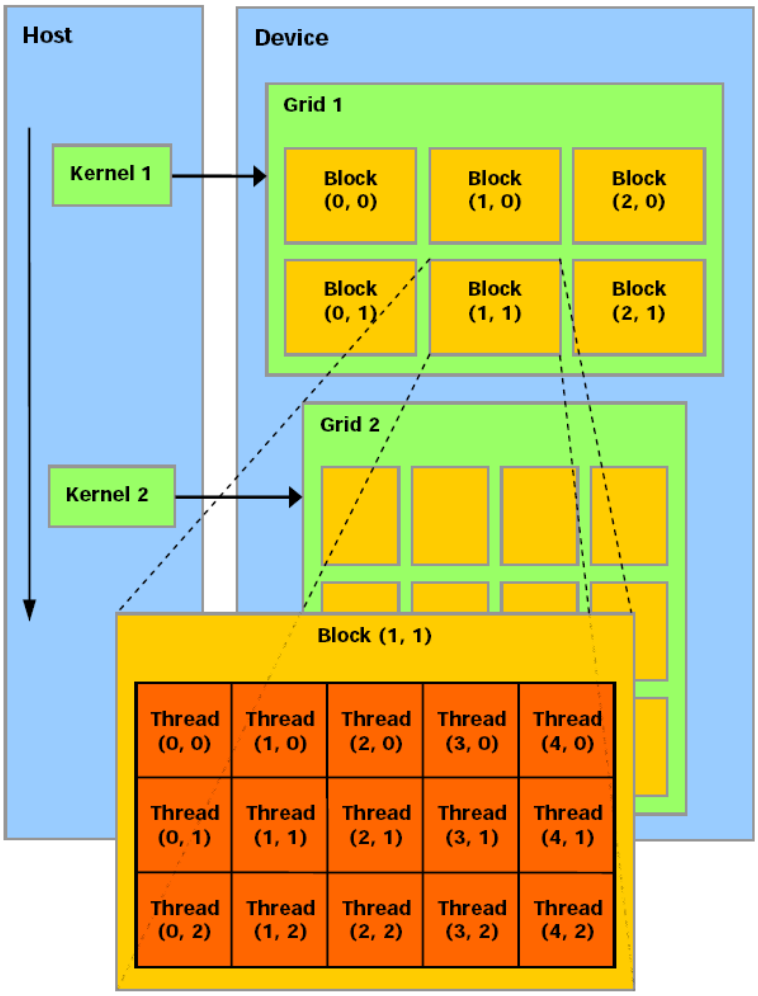


Figure 6.2: Each kernel is executed by CUDA as a group of threads within a grid. Image courtesy of NVIDIA.

are chosen to optimise the utilisation of the available computational resources. Care should be taken at the multiprocessor level in balancing the available memory required by the kernels with the ability to hide global memory latency. Since a finite amount of memory is available on a multiprocessor, the memory requirements of a kernel will determine how many threads can run concurrently on each. Importantly, CUDA's use of time slicing allows more than one block to be executed concurrently on a single multiprocessor, which has important implications for hiding memory latency. If more than one block is executing, the multiprocessor is able to switch processing between blocks while others are stalled on memory accesses, whereas it has no option but to wait for these if only one block is executing. Therefore for memory bandwidth bound kernels it may be preferable to launch several smaller blocks on each multiprocessor rather than a single larger one if both configurations make

the same use of multiprocessor memory resources. While tools are available from NVIDIA for estimating the optimal execution configuration, it has proved necessary to fine tune the configuration experimentally for each kernel.

6.3.3 First implementation: scatter as a gather

SOFA integration.

Implementing a biomechanical model in SOFA translates essentially into writing a new Force Field, that is describing the algorithm used to compute internal forces in the model. It merely comes down to creating a single C++ class and changing the position reads and force writes to integrate the algorithm into SOFA's design. The precomputation phase takes place in the initialisation method where relevant variables are computed and passed to an external C function that allocates memory on the GPU and binds textures to it. During the simulation loop, the Solver requests the computation of the forces by launching the appropriate kernels on the GPU.

Kernel organisation.

Although CUDA allows scattered writes in theory, it offers no write conflict management between threads. Why is this a problem? The major steps of the TLED algorithm are the following: (1) we compute the element nodal forces, that is for each node of a given element, we calculate the force contribution that this element has on this node; (2) we obtain the global force for each node by summing up the force contributions of all elements to which this node is attached; (3) we compute the displacement for each node by integration using the central difference method. In a serial implementation (CPU) we would do a loop on all elements and for each element we compute all element nodal force contributions. We would then directly add the element force contributions computed for each node into a global node array at the corresponding location for each node of the current processed element. In a parallel implementation, many elements are processed concurrently, in parallel. Inevitably, two elements sharing a node may be processed in the same time and may therefore try to both write their computed nodal force contribution into the exact same location in GPU's memory. At the time of this first implementation (2007), the result of concurrent writes was undefined. In fact, only one write was guaranteed to happen and there was no way to know which one succeeded. Potential measures to address this have proved extremely detrimental to performance. For this reason, the kernel arrangement in our first CUDA implementation eventually ends up to be essentially the same as the one used by Taylor et al. (2007a).

Consequently, the TLED CUDA implementation also relies on 2 kernels. The first kernel operates over elements in the model and computes the element stresses based on the current model configuration. It then converts these into nodal force contributions, which are written to global memory. The second kernel operates over nodes and reads the previously calculated element force contributions and sums them for each node. The SOFA central difference solver computes the displacements from

the nodal forces. Therefore, due to the impracticability of scattered writes, the sum operation is reformulated as a gather and the second kernel is needed to sum the nodal forces. The use of the developed viscoelastic constitutive update scheme necessitates storage of an additional array of state variables $\boldsymbol{\Upsilon}_i$.

Memory usage.

One efficient method for reading global memory data within kernels is texture fetching. Textures may be bound to either *cudaArrays* or regions of linear memory. CudaArrays have been designed to achieve optimal fetching when the access pattern has a high level of 2D locality. In the present application, the access pattern among threads is essentially random (since unstructured meshes are used) and our experiments have shown that texture fetching from linear memory is in fact fastest. Therefore all global memory precomputed variables were accessed using this method.

In SOFA, the forces are stored on the GPU in global memory. Since this memory space is not cached, it is important to follow the appropriate access pattern to obtain maximum memory bandwidth, especially given how costly accesses to device memory are. A multiprocessor takes 4 clock cycles to issue one memory instruction for a set of threads. When accessing global memory, there are, in addition, 400 to 600 clock cycles of memory latency. A suboptimal access pattern would yield incoherent writes. The memory bandwidth would then be an order of magnitude lower. In order to prevent this, a key feature of CUDA has been used: *shared memory*. This is a very fast memory shared by all the processors of a multiprocessor. Hence, the results of the second kernel are first copied to shared memory and then moved to global memory. If the copies are well organised, it is possible to re-order the access to fulfil all the memory requirements (for both shared and global) and thus reach the maximum bandwidth.

CPU-GPU interaction.

CPU-GPU interaction is generally a significant bottleneck in General Purpose GPU applications due to the relatively low interface bandwidth and it is desirable to minimise such interaction. However, interaction cannot be entirely removed from the present implementation since, for example, the solver requires inputs in the form of loaded nodes (which may change due to the interaction with the user) and their displacements, and may need to provide outputs in the form of reaction forces for haptic feedback. CUDA alleviates the problem somewhat by allowing allocation of areas of page-locked host memory which are directly accessible by the GPU and therefore offer much higher bandwidth. In SOFA, all transfers between CPU and GPU are made via this mechanism.

Element technology.

We used both reduced integration 8-node hexahedral and 4-node (linear) tetrahedral elements. Tetrahedral meshes are easily generated and therefore widely used

in simulations. However, hexahedra are preferable both in terms of solution accuracy and computational efficiency. Indeed, 4-node linear tetrahedra are known to be susceptible to volumetric locking (Hughes, 2000). Yet, a disadvantage of hexahedra is the difficulty in automatically generating hexahedral meshes over arbitrary domains. Construction of hexahedral meshes is time consuming and laborious, which fact is of even greater significance when patient-specific simulations are considered. For this reason tetrahedral meshes are widely used in simulations despite their short comings.

Another drawback of using hexahedra is the existence of so-called Hourglass modes that have to be addressed to avoid deterioration of the solution (Flanagan and Belytschko, 1981). Techniques for suppressing these modes exist, but naturally involve additional computations. However, for a given number of degrees of freedom (DOF), a hexahedral mesh can be built with far fewer elements than a tetrahedral one. Since the majority of the calculations in explicit dynamic analyses are performed per element, this results in reduced overall computation time.

From a GPU perspective, hexahedral element computations are substantially heavier and demand more memory resources than linear tetrahedral elements. Most of the matrices (such as shape function derivatives and nodal displacements) are twice as large for hexahedra, which necessitates twice as many texture fetches per element, and use of twice as many registers per thread. Similarly, twice as many nodal forces per element are written to global memory. Additional variables associated with hourglass control are required also. Therefore, on a per element basis hexahedra are significantly less efficient than tetrahedra, especially for GPU execution where memory efficiency is crucial. Thus, we observe that the occupancy (GPU percentage usage) drops from 25 % to only 8 % when using hexahedral elements. However, from the point of view of an entire model the lower number of hexahedra required for a given number of degrees of freedom still outweighs this element-wise inefficiency.

6.3.4 Second implementation: a better use of shared memory

Limitations of the previous implementation

The previous TLED algorithm has been extensively tested on regular meshes like cubes. The model was purposely simple to test our new CUDA implementation. However, the TLED algorithm was developed to compute the deformation of organs. Thus, we started to assess our TLED implementation on an irregular mesh: a liver. From a segmented liver obtained from IRCAD (Research institute against digestive system cancer, France), we meshed the surface using a marching cube technique (Lorensen and Cline, 1987). After smoothing it with Blender, a tetrahedral mesh has been generated with Tetgen, a free mesh generator. From there, we realised that the TLED was highly non efficient with such a mesh. After an extensive analysis with the profiler provided by NVIDIA, we became able to explain it. The liver mesh has a higher valency, meaning that each node has more elements attached than

within a regular mesh like a cube. And on GPU, nodes are processed in parallel by group of 32 threads (called a warp). But if the instructions are different within a warp, the warp is divergent and the computations are serialised. With its higher valency, the liver mesh involves much more divergent warps when the forces are added up. To give an idea of the difference, a cube (343 nodes and 1 296 elements) is computed in 0.16 ms when a liver mesh (399 nodes and 1 250 elements) is processed in 0.27 ms. Based on the limitations discovered when the TLED has been applied to this irregular mesh, we came up with a new design, that we implemented and tested.

A single kernel approach

The main bottleneck of the previous implementation was adding up the forces on each node in a second kernel after the computation of element force contributions in a first kernel. The latter were stored in 4 different textures (8 for the hexahedral formulation) and in order to add up the forces the second kernel had to switch all the time between the correct textures within the warps, causing them to diverge and slowing down substantially the overall process. Our idea was therefore to use shared memory to avoid writing in global memory at the end of the first kernel and prevent the warps to diverge. As explained before, shared memory is a very fast memory shared by all the processors of a multiprocessor. And each block of threads is sent to a multiprocessor to be executed. Obviously the amount of shared memory is finite, which limits the validity of this new approach and the results in performance will depend on the hardware (which may have different amount of shared memory).

In the precomputation phase, nodes and elements are sorted by block to leave enough space for storage in shared memory of the element force contributions initially calculated at the end of the first kernel. We add nodes to a block along with their unique elements until the shared memory is saturated. Of course, in this approach some elements will be computed several times (because being into different blocks). Yet, storing the force contributions in shared memory instead of global memory to read them later on via texture fetches may outperform the need of computing several elements several times. However, the efficiency of this single kernel approach depends on the nature of the mesh. Because the force contributions of all elements attached to a node must be eventually added up, this approach allows to avoid the numerous (slow) accesses to global memory when the valency of the nodes is high (that is, nodes are attached to many elements). This type of mesh is often encountered when dealing with the complex shapes of anatomical structures. Therefore, where the first implementation showed its limits with complex meshes, this single kernel approach has proven to be more computationally effective.

TODO: If time, try this implementation on Geforce GTX 480 as it features 3 times the amount of shared memory.

6.3.5 Third implementation: atomic writes

In April 2010, NVIDIA released a family of graphics cards based on a new GPU architecture called Fermi. Among many new features and improvements over previous generation, these cards added the capability of so-called *atomic writes* for floats. The write operation is said to be atomic in the sense that it is guaranteed to be performed without interference from other threads. In other words, no other thread can access this address until the operation is complete. The concurrent writes are simply serialised in hardware. This feature allows us to implement the TLED algorithm as we would on a CPU, using a single kernel for all computations, and without the need to re-order the nodes and elements organisation as with the second implementation.

TODO: If time, try simple implementation using atomic writes with floats on Geforce GTX 480

6.4 Results

We present a series of examples based on pure shear and compression of cube models to demonstrate the validity of the constitutive update procedure and the performance of the GPU-based implementation. We conclude with an example of simulation of liver deformation, since as mentioned there is experimental evidence that liver exhibits an anisotropic response (Chui et al., 2007). In each case we used a transversely isotropic visco-hyperelastic model with elastic strain energy components defined by

$$\begin{aligned}\Psi^{iso} &= \Psi^{I}(\bar{I}_1) + \Psi^{TI}(\bar{I}_4) \\ &= \frac{\mu}{2}(\bar{I}_1 - 3) + \frac{\eta}{2}(\bar{I}_4 - 1)^2, \qquad (6.4)\\ \Psi^{vol} &= \frac{\kappa}{2}(J-1)^2, \qquad (6.5)\end{aligned}$$

where Ψ^{I} and Ψ^{TI} are isotropic and transversely isotropic components, respectively, μ is the small strain shear modulus, κ is the bulk modulus, and η is a material parameter with units of Pa. The isotropic component Ψ^{I} represents the well known neo-Hookean model, while the anisotropic component Ψ^{TI} is the same form used by Picinbono et al. (2003). Here the dependence of Ψ^{TI} on $\bar{I}_5$ is omitted and we include only $\bar{I}_4$ terms. This was done foremost for simplicity, but is also a common practical feature of anisotropic models for soft tissues. The main reason is the clear physical interpretation of $\bar{I}_4$ as the square of the stretch λ [1] along the preferred direction $\mathbf{a}_0$ (Holzapfel et al., 2000), whereas $\bar{I}_5$ has a less clear physical meaning. It must be emphasised that this model is used as an example only, and we make no claim concerning its appropriateness for any particular tissue.

[1] Strictly, $\bar{I}_4 = J^{-2/3}\lambda^2$

The instantaneous isochoric and volumetric responses are then given by

$$\mathbf{\Phi}^{iso} = J^{-2/3}\left[\mu\left(\mathbf{I} + \frac{I_1}{3}\mathbf{C}^{-1}\right) + \eta(\bar{I}_4 - 1)\left(\mathbf{A}_0 + \frac{I_4}{3}\mathbf{C}^{-1}\right)\right], \tag{6.6}$$

$$\mathbf{\Phi}^{vol} = \kappa J(J-1)\mathbf{C}^{-1}. \tag{6.7}$$

We used viscoelastic isochoric terms only, with a single Prony series term for simplicity. The complete stress response is then

$$\begin{aligned}\mathbf{S} &= \mathbf{\Phi}^{vol} + \int_0^t \left[1 - \alpha_1\left(1 - e^{(s-t)/\tau_1}\right)\right]\frac{\partial \mathbf{\Phi}^{iso}}{\partial \tau}ds \\ &= \mathbf{\Phi}^{vol} + \mathbf{\Phi}^{iso}(1-\alpha_1) + \alpha_1\int_0^t e^{(s-t)/\tau_1}\frac{\partial \mathbf{\Phi}^{iso}}{\partial \tau}ds.\end{aligned} \tag{6.8}$$

Note that in this case the developed constitutive update procedure need only to be employed for isochoric terms.

6.4.1 Pure shear of a cube

The first example involved shearing of a unit cube model. We imposed a ramped loading such that the displacements u of the loaded nodes at time t were given by $u = rt$, where r is the loading speed, and we chose the x-direction to be the preferred material direction ($\mathbf{a}_0 = [1\,0\,0]$). Since the deformation is homogenous only a single element was required to reproduce the stresses exactly. Moreover an analytical solution is available with which numerical results may be compared (Taylor et al., 2009). Four independent nonzero stress terms are produced.

In Fig. 6.3 we demonstrate the effects of material anisotropy independent of viscoelastic effects by plotting the limiting instantaneous (hyperelastic) stress curves, produced by setting the relaxation parameter $\alpha_1 = 0$. Fig. 6.3(a) shows curves for the anisotropic model, while Fig. 6.3(b) shows curves for an equivalent isotropic model (i.e. with $\eta = 0$). Firstly we confirm that the FE solutions (data marks) match the analytical solutions (solid curves) very accurately. Secondly we observe that the anisotropic term significantly affects both the magnitude and shape of the stress curves. In particular the S_{11} curve which is coincident with the S_{33} curve in the isotropic model actually changes sign when anisotropy is introduced. The other curves exhibit a more pronounced stiffening for the anisotropic case.

In Fig. 6.4 we demonstrate the effect of loading rate on the complete viscoelastic model by plotting the S_{11} component for strain rates of $2.5\,s^{-1}, 0.25\,s^{-1}$, and $0.025\,s^{-1}$, plus the bounding instantaneous and equilibrium responses. The strain rate-dependence introduced by the model (and commonly observed in biological tissues) is clearly shown. Additionally the close match between the analytic and numerical solutions demonstrates the validity of the developed constitutive update scheme.

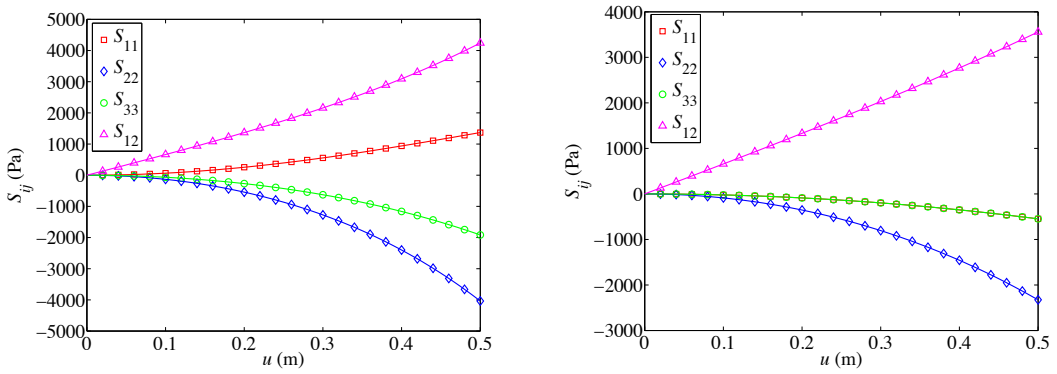


Figure 6.3: Instantaneous stress curves for pure shear deformation of (a) the anisotropic model compared with those of (b) an isotropic model. Solid lines correspond to the analytical solution, while markers indicate the FE solution.

6.4.2 Compression of a cube

The second example involved compression of a cube of edge length 0.1 m (comparable to some human organs). The cube was imagined to be fixed to opposing load platens and free on the remaining four faces. Compression was applied along the x-axis. A stress-relaxation type test protocol was simulated, in which the cube was compressed by 30 % in 0.5 seconds and the compression was held for a further 4.5 seconds. We demonstrate the effects of anisotropy on the deformation response by comparing the deformed shapes of anisotropic models with $\mathbf{a}_0 = [0\,1\,0] \stackrel{\text{def}}{=} \mathbf{a}_0^y$ and $\mathbf{a}_0 = [0\ \frac{1}{\sqrt{2}}\ \frac{1}{\sqrt{2}}] \stackrel{\text{def}}{=} \mathbf{a}_0^{yz}$ with that of an isotropic model (with $\eta = 0$).

Figure 6.5(a) shows the undeformed cube, while Fig. 6.5(b) shows the deformed isotropic model. With no preferred direction the lateral expansion was uniform. In Fig. 6.5(c) the increased y-direction stiffness of the first anisotropic model lead to a much reduced expansion in this direction (vertical in the image), and an accompanying increase in the orthogonal z-direction. For the second anisotropic model the direction defined by $y = z$ was stiffened, and Fig. 6.5(d) shows the resulting reduced expansion along this axis and the increased orthogonal expansion.

The end face reaction force history for each model is shown in Fig. 6.6. A distinct decay curve resulting from stress relaxation, and similar to that commonly noted in biological tissues (Fung, 1993), was observed. The increased stiffness afforded by the anisotropic models (albeit in an orthogonal direction to the loading) result in greater reaction forces than the isotropic model.

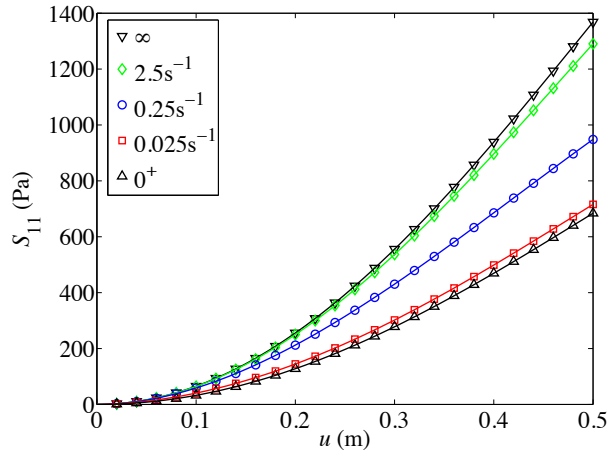


Figure 6.4: S_{11} curves for pure shear deformation of the anisotropic viscoelastic model at varying strain rates. Curves for strain rates of $2.5\,s^{-1}, 0.25\,s^{-1}$, and $0.025\,s^{-1}$ are given (as labelled), along with the bounding instantaneous and equilibrium responses. The latter two are labelled ∞ and 0^{+}, respectively, indicating that they correspond to strain rates approaching these values. Solid lines correspond to the analytical solution, while markers indicate the FE solution.

6.4.3 GPU performance

General GPU performance

Next, we assessed the computational performance of the GPU implementation and efficiency of the constitutive update scheme by measuring computation times for a range of mesh densities. Using the cube geometry as above we generated models with mesh sizes ranging from 3 993 DOF (1 331 nodes, 1 000 hexahedral elements) to 177 957 DOF (59 319 nodes, 54 872 hexahedral elements). We compared the GPU solution times for a single time step with those of a CPU-based implementation developed using C++. The test machine included an Intel Core2Duo 2.4GHz CPU, 2GB RAM, and an NVIDIA GeForce 8800GTX GPU. Three constitutive models were considered: the transversely isotropic viscoelastic model (TIV), TIV minus the viscoelastic terms (TIE), and TIE minus the anisotropy terms (NHE). Comparison of times for TIV and TIE indicates the computational load introduced by the developed viscoelastic constitutive update scheme. Comparison of times for TIE and NHE indicates the computational load introduced by material anisotropy. Figure 6.7 shows the results of the experiments.

In Fig. 6.7(a) and Fig. 6.7(b) it can be seen that GPU solution affords significant speed improvements over CPU solution for all constitutive models. Additionally it appears that solution times are little affected by the introduction of the more complex constitutive models, and importantly by the use of the developed constitutive update scheme. This is borne out in Fig. 6.7(d), where we observe that the maximum solution time ratio for model TIE to model NHE (anisotropy vs isotropy) was

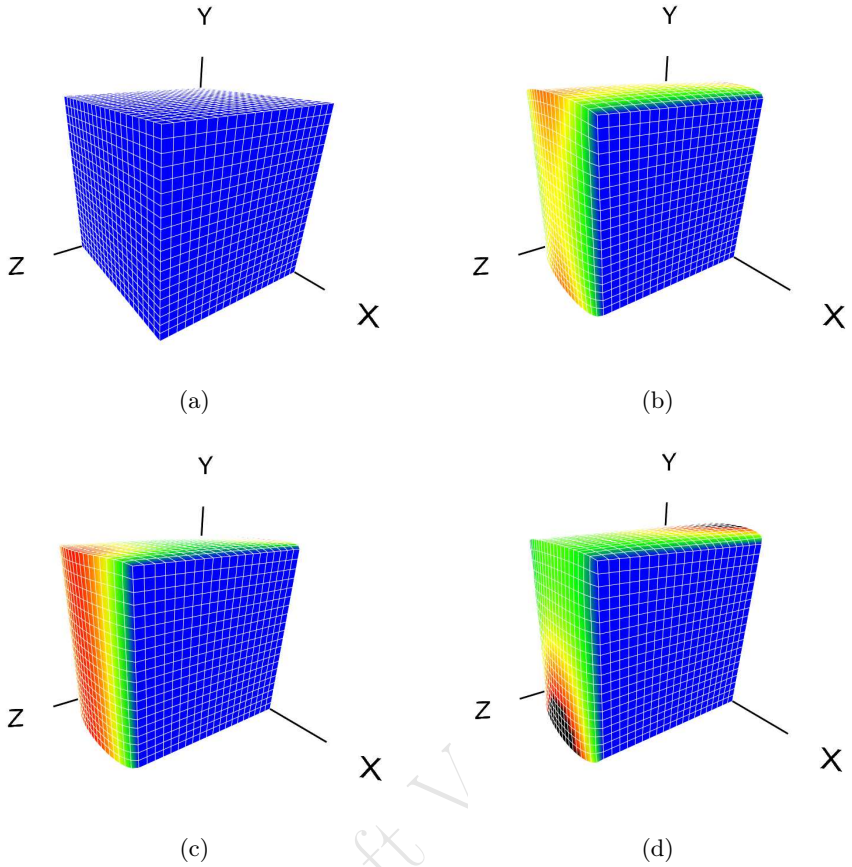


Figure 6.5: Deformation patterns of transversely isotropic models compared with that of an isotropic model: (a) the undeformed cube, (b) the deformed isotropic model, (c) the deformed anisotropic model with $\mathbf{a}_0 = \mathbf{a}_0^y$, (d) the deformed anisotropic model with $\mathbf{a}_0^{yz}$. Colour maps indicate relative magnitude of lateral displacement ($(u_y^2 + u_z^2)^{1/2}$).

1.013, and that of model TIV to model TIE (viscoelastic vs elastic) was 1.043. The largest total solution time increase for an anisotropic viscoelastic model compared with an isotropic hyperelastic one was 5.1 % We conclude that the key features of anisotropy and viscoelasticity may be included in simulations at very little additional computational cost. Referring to Fig. 6.7(c) we observe a maximum speed improvement for GPU solution over CPU solution of 56.3×.

Fig. 6.7(b) also shows the decreasing critical time steps Δt_{cr} with increasing model sizes (which results from the decreasing element sizes), using liver-like and brain-like materials. We define liver-like materials to be those described by model TIV with material parameters quoted at the beginning of the section, whereas brain-like materials are those described by model NHE with material parameters $\mu = 1006.71\,\text{Pa}$ and Poisson's ratio $\nu = 0.45$. The intersections of these curves with the solution time curves provide an estimate of the real-time capacity of the

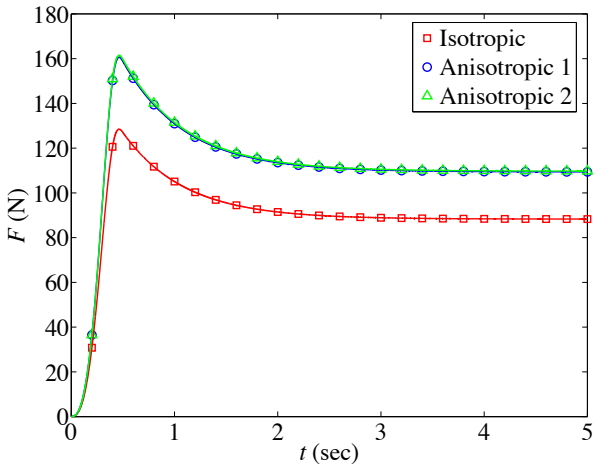


Figure 6.6: FE computed reaction forces on the end faces of the cube models over time. Curves are labelled according to the constitutive model used, where *Anisotropic 1* refers to the transversely isotropic viscoelastic model with $\mathbf{a}_0 = \mathbf{a}_0^y$ and *Anisotropic 2* refers to the case of $\mathbf{a}_0 = \mathbf{a}_0^{yz}$. Note that in this figure both markers and solid lines correspond to FE solutions.

implementation. This is discussed further in section 6.5.1.

This GPU implementation of the TLED algorithm was demonstrated publically at MICCAI 2007 during the talk given by Zeike Taylor (Taylor et al., 2007a). A non-linear finite element analysis of a cube featuring brain-like properties was solved in real-time at 1 430 Hz using 55 566 tetrahedral elements (see Fig. 6.8).

Performance within SOFA

The efficiency and the side-effects of porting the algorithm into the flexible SOFA framework need to be measured. Therefore the performance has been assessed by comparing the computational time of the algorithm running within and outside SOFA. NVIDIA provides a tool to check the GPU implementation by evaluating many variables during the execution like for instance timings, counts of inconsistent reads and writes or GPU occupancy. We used this tool to carry out two measures:

1. GPU time only estimates the GPU computational time.
2. CPU time allows the evaluation of the execution time with the additional overhead due to the framework.

The tests were performed on a simple test scene featuring a cube under gravity. Hexahedral meshes with different resolutions from 1 331 to 29 791 nodes were used and the results are presented in figure 6.9. In our standalone implementation, the second kernel not only accumulates nodal forces but also adds gravity and updates

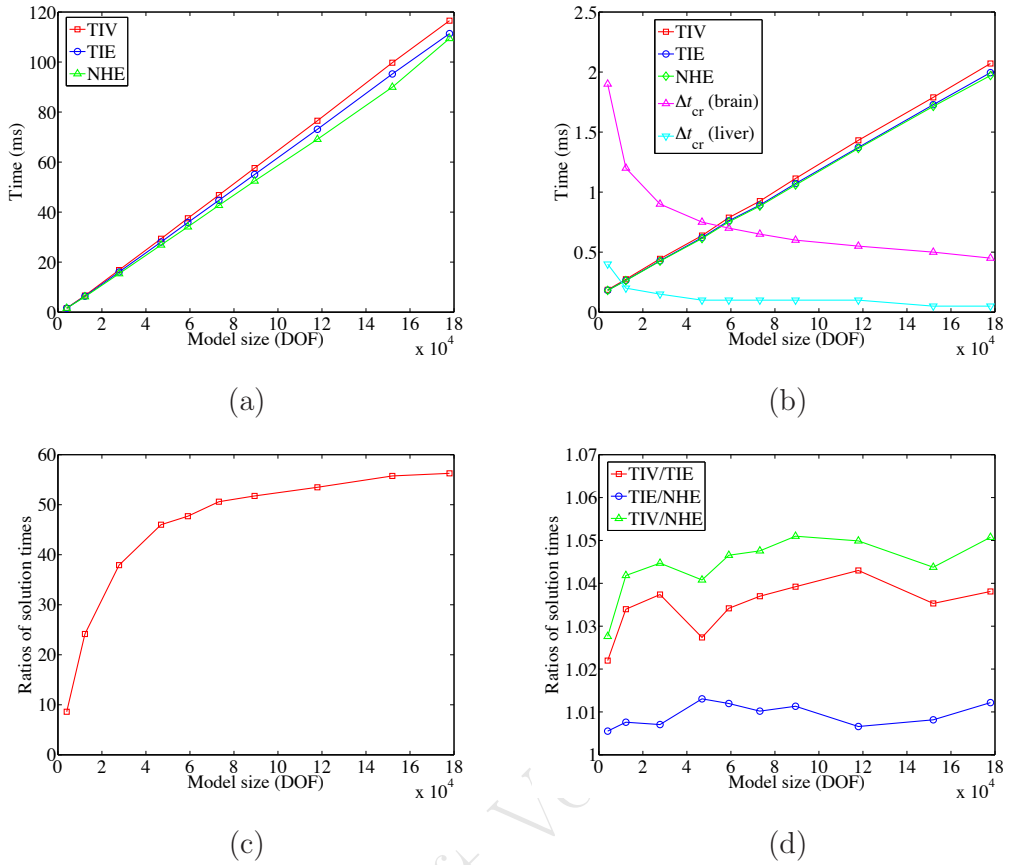


Figure 6.7: (a), (b) show CPU and GPU solution times, respectively, for a single time step for each constitutive model as a function of model size, (c) shows the ratio of CPU to GPU solution times for model TIV, and (d) shows ratios of GPU solution times for the constitutive models: TIV to TIE, TIE to NHE, and TIV to NHE. (b) also shows the critical time steps Δt_{cr} for each model size using liver-like and brain-like properties.

positions based on the central difference integration scheme. These operations are split into separated components in SOFA, in order to introduce more flexibility (such as applying additional forces or changing the integration algorithm). While this introduces no noticeable difference on a CPU-based simulation, when using the GPU it is more costly due to overheads in the CUDA API for the additional kernel launches. Although it reduces the performance by 8.4 % for large meshes, this could be optimised away by adding a kernel specific to a given combination of components.

6.4.4 Simulation of liver deformation

Finally, we performed simulations of manipulation of a liver model in order to demonstrate the feasibility of the procedure with more realistic organ geometries.

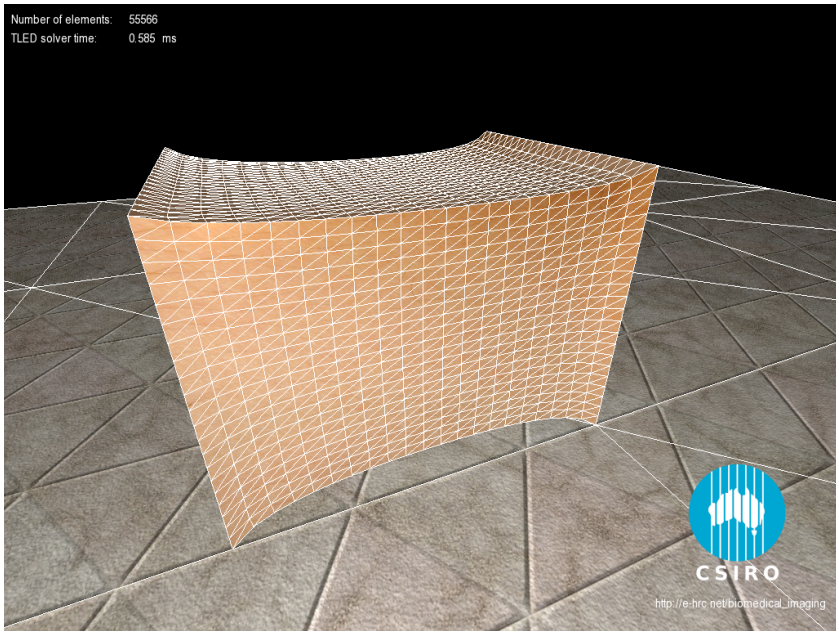
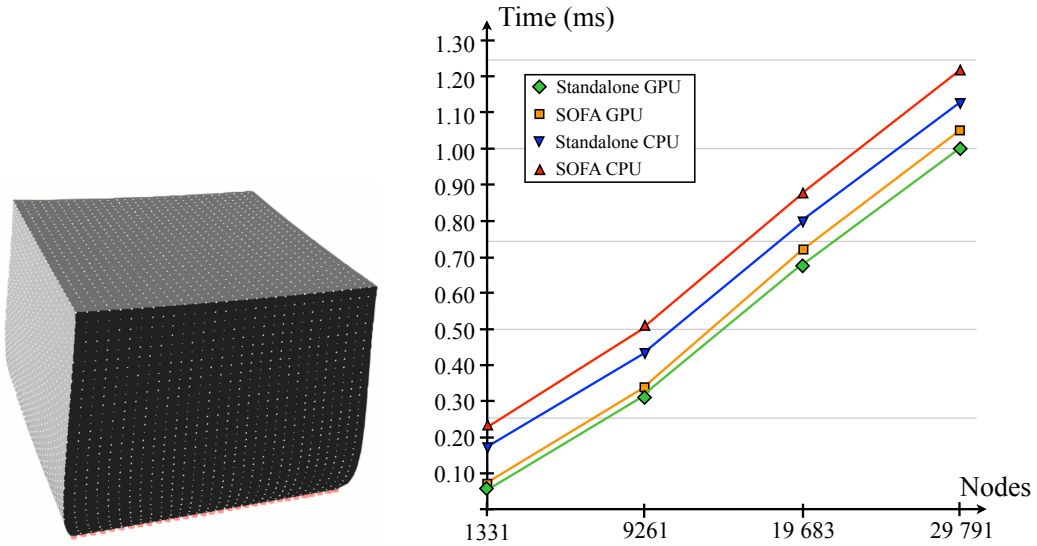


Figure 6.8: Non-linear finite element algorithm solved in real-time at 1 430 Hz with 55 566 elements. A strain between 0 and 40 % is periodically applied to one face whilst the opposite face is fixed. The material properties were the following: cube's edge = 10 cm, Young modulus = 3 000 Pa, Poisson's ratio = 0.45, density = 1 000 $kg.m^{-3}$ and the time step was 0.7 ms. The normal for each vertex was computed in real-time for a proper lightning and a skin texture was applied.

The model geometry was based on the liver mesh available in the SOFA framework, which we smoothed and remeshed to improve uniformity of element sizes and node valencies. A mesh of 748 nodes and 2 428 tetrahedral elements was solvable in real-time. For anisotropic models a preferred material direction $\mathbf{a}_0 = [0\,1\,0]$ was used, corresponding to the vertical direction in Fig. 6.10. We note that in a real organ the preferred direction likely varies depending on the local vasculature and orientation of lobules (Chui et al., 2007), but in the absence of experimental data for the present specimen the mentioned direction was assumed throughout. Moreover, on this point we note that different preferred directions may be specified for each element in the mesh, if such data are available.

The Von Mises equivalent stress distributions resulting from stretching the model with and without inclusion of viscoelasticity are compared in Fig. 6.10. For the viscoelastic case (Fig. 6.10, right image) the model is depicted several seconds after the completion of the stretch, so that stresses had reached an approximately steady state. The stress dissipation introduced by the viscoelastic model is clearly evident, and for example would result in significantly altered virtual tool reaction forces.

TODO: Add movies online? If possible...



(a) (b)

Figure 6.9: (a) FEM mesh of cube with 29 791 nodes deformed under a uniformed load. (b) Comparison of GPU computational timings and CPU overheads between the SOFA and standalone implementations for different mesh sizes. TODO: change the cube if I can get a decent visual model in SOFA

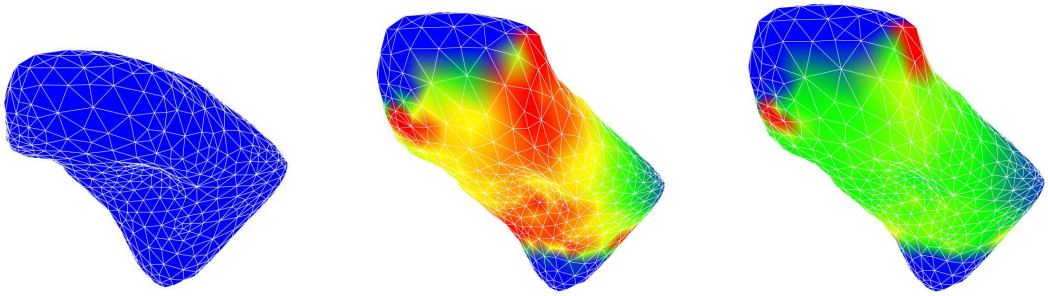
6.5 Discussion

6.5.1 Critical time step

As already emphasised in section 5.1.3 page 77, the use of an explicit integration scheme imposes that the solution time remains below the critical time step. This critical time step depends on two kind of criterion: the size of the elements (due to the choice of the mesh) and the material's properties (Young's modulus and Poisson's ratio).

The first factor leads to a strong constraint on the quality of the mesh. Indeed, meshing a complex shape with good and uniform elements is not a straightforward task and rather often bad and very small elements are created. In such a case, the critical time step for the whole simulation would be small even if there is only a single small element in the mesh. As an example, the liver mesh used in the section 6.4.4 had to be smoothed and remeshed properly with great care to allow a decent simulation.

Material's properties are also a very influent factor for the determination of a critical time step. If the critical time step may be fairly large for very soft tissue (like brain), it is already divided by 2 when modelling liver. As we have seen, the value of the Poisson's ratio may be relaxed from 0.49 to 0.45 for instance. If it allows a time step twice as large, this relaxation introduces inaccuracies. If this is certainly acceptable for medical training simulators, the demand in precision



(a) (b) (c)

Figure 6.10: Deformation of a liver model using the TI model (b) and the TIV model (c). The undeformed mesh is shown in (a). Colour maps indicate the relative Von Mises equivalent stress magnitude.

for surgical planning or per-operating assistance would probably prevent such an approximation.

In a nutshell, additional care must be taken to produce quality meshes and the range of materials that may be used while maintaining interactive computations is limited when using an explicit time integration scheme. However, it allows a huge gain in speed for very soft tissue when the compromises on material's properties are acceptable for the desired application, as with medical simulators used for training. For the interested reader, a more thorough discussion on the critical time step of explicit analyses may be found in Taylor et al. (2008b).

6.5.2 Handling contacts

The present work was entirely concerned with numerical solution of constitutive models in explicit FE analyses, however an equally important issue from the point of view of interactive simulation is the modelling of interaction between organs and virtual tools, and its possible impact on Δt_{cr} and the stability of the simulation. Explicit time integration schemes are known to be more sensitive to contacts than implicit schemes. While this is beyond the scope of the present work, we note that a range of contact formulations are available, with two main types in common use: penalty methods and kinematic constraint methods. Penalty methods involve use of fictitious contact springs at interfaces, effectively to penalise interpenetration of opposing surfaces. If excessively stiff springs are used, Δt_{cr} may indeed be reduced. According to LS-Dyna documentation (Hallquist, 2006), except in extreme conditions (that is from explosive or high speed impacts, or imposition of very large forces on highly constrained objects), spring stiffnesses of the same order of magnitude as the material stiffness are suitable and the effect on Δt_{cr} is therefore minimal. However, in practice, we observed that spring stiffnesses greater than the material stiffness are often required to prevent any penetration of the two objects, which results in an unstable system. Kinematic constraint methods involve (loosely) direct

relocation of penetrating nodes back to the outside surface of the penetrated body, and have no effect on Δt_{cr}. In theory, these may be especially useful for modelling interaction between soft tissues and stiff (effectively rigid) surgical tools. It is worth nothing that the computational cost of either variety has not been investigated here.

Even if a stable contact modelling could be achieved, one element of crucial importance has been omitted so far: the collision detection process. Indeed, before even handling the interaction between two objects, their interaction needs to be detected. Because the time step must be very small when using an explicit time integration scheme (often around one millisecond or less), the collision detection algorithm must also be run at the same frequency. The problem is that no collision detection algorithm is fast enough to execute at 1 000 Hz or more with objects of reasonable sizes. Of course, one could choose to detect the collisions at a lesser frequency but the risks of penetration would substantially increase. One notable exception is the interaction between a soft deformable object and a fixed rigid surface where the collision detection is rapid.

One possible extension to enhance the stability is to monitor the critical time step of the simulation. Indeed, we know an approximation of this critical time step and we recall its expression here for convenience:

$$\Delta t_{cr} = \frac{L_e}{c}, \tag{6.9}$$

where L_e is the smallest characteristic element length (roughly interpreted as the smallest edge length) in the assembly, and c is the dilatational wave speed of the material which depends on material properties (such as Young's modulus and Poisson's ratio). More details on the critical time step were given page 78. If c is constant for a given material, L_e varies over time during the simulation, in particular when contacts occur. Maintaining the time step of the simulation below the monitored critical time step would insure the stability of the simulation. However, by possibly reducing the time step of the simulation, real-time computations can no longer be guaranteed.

6.5.2.1 Conclusion

As we showed in the last two chapters, the TLED algorithm is a very efficient fully non-linear finite element formulation. We also added viscoelasticity and anisotropy features with minimal impact on its performance. In particular, its GPU implementation allows non-linear analyses of structures in real-time using a high number of elements as never before.

However, the use of an explicit integration scheme imposes substantial constraints which limits TLED's range of applications. More specifically, the usage of the TLED algorithm in very interactive environments (as in the manipulation of a liver with a tool) is not recommended. The collision detection would either become the bottleneck of the simulation (if run at every time step) or fail to provide accurate detection of contacts (otherwise). In addition, the TLED algorithm

is also not appropriate to simulate the deformation of rigid object as the critical time step would be too small to allow real-time computations. In constrast, TLED excels for computing the deformation of soft tissues when the boundary conditions are controlled. As an exemple, the TLED algorithm may efficiently provide brain shift predictions as the interaction of the brain with the fixed skull is determined beforehand and rapid collision detection can be achieved in this case.

Hence, provided that the application is appropriate, the TLED algorithm is an efficient and accurate algorithm to model the deformation of solid soft anatomical organs.

Part III

Hollow organs modelling

Chapter 7

Modelling the deformation of hollow objects in real-time

The human body is composed of various deformable anatomical structures and finding appropriate soft-tissue models for all these type of structures is challenging. In the previous two chapters, we described an efficient non-linear FEM formulation for simulating the deformation of solid organs (like brain, liver, prostate etc.). But there is also a need to simulate the deformation of hollow organs, whose volume is negligible compared to their surface area, such as colon, blood vessels, stomach etc. The solid models previously mentioned are not appropriate and specific approaches are required for these thin anatomical structures. In this chapter, we will review diverse techniques applied for the simulation of thin objects. These different approaches may be grouped into three main categories: (1) mass-spring models, (2) approaches which derive a bending energy from geometrical considerations and (3) methods based on the equations of continuum mechanics. Although very few models have been proposed for simulating the deformation of thin anatomical structures in the field of medical simulation, we will try to emphasize medical applications whenever possible.

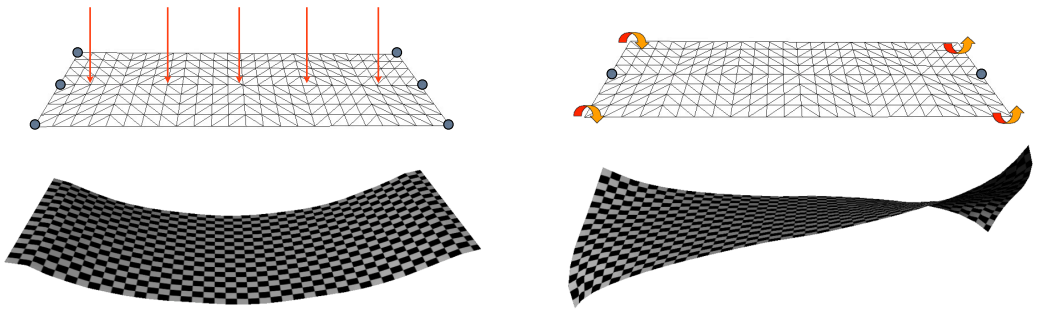
7.1 Introduction: the problematic

The human body is composed of various deformable anatomical structures. A key challenge of soft-tissue modelling is the variousness of the mechanical behaviours. The shape and the internal structure may be very different from one organ to another. If you take the liver for instance, it is a solid organ composed of four lobes, each one made up of units called lobules. Each lobule consists of a central vein surrounded by liver cells and constitutes the end of a dense network of blood vessels. This particular internal structure has a strong influence on the mechanical properties of the liver. In contrast, the colon is a long and hollow organ. It consists of four sections which all have different shapes and material properties. Consequently, it seems unrealistic to use a unique model for all tissues. Yet, most of previous works focus on volumetric models that are able to capture the behaviour of solid organs. In fact in the field of medical simulation, very few models have been proposed for simulating, in real-time, the deformation of thin anatomical structures whose volume is negligible compared to their surface area. Examples include hollow structures, such as the wall of blood vessels, or membranes, such as the Glisson's capsule surrounding the liver. It is also of particular interest to us for modelling the colon in our colonoscopy simulator and for simulating the deployment of the implant in cataract surgery.

The first idea that may come to mind for modelling thin structures is to apply the same methods than with solid objects. For instance, Holzapfel et al. (2002) proposed a very complex 3-layer model of an aterial wall for simulating an angioplasty procedure. Their model is non-linear and anisotropic along fibers placed according to histology. They also model diseased part of the vessel through a hardening parameter which adds plasticity. However, they discretised their geometric model with eight-node brick elements. Another example is given by Aloisio et al. (2004) who assume that the blood flow in the vessels may be considered as incompressible and the artery consequently becomes a solid body and can be modelled as a solid deformable object. TODO: why is this not appropriate? High-order elements could do the trick of following the curvature... Hence, specific models for thin objects were developed.

Indeed, numerous models are available in the literature to describe physics of thin objects, from fairly simple and naive approaches to more complex and thorough representations. Continuum mechanics provides many formulations able to accurately describe stresses occurring within thin objects. Most of them fall into one of the following two categories: plate theory or shell theory. Those theories have been a subject of interest in the mechanical community for decades. The difference between these two kinds of structures is very well explained by Liu and Quek (2003) and can be summarised by the fact that plate bending elements can only carry transversal loads while shells can undergo more complex deformations. For instance, if we consider the horizontal board of a bookshelf, that board can be approximated as a plate structure and the transversal loads are the weight of the books. A typical deformation of the board is illustrated in Fig. 7.1 (a). Conversely, a

shell structure can carry loads in all directions, and therefore can undergo bending, twisting and membrane (that is, in-plane) deformation (see Fig. 7.1 (b)).



(a) Plate theory (b) Shell theory

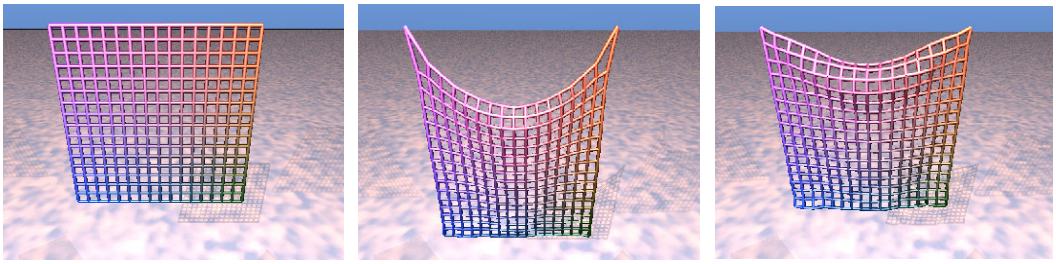
Figure 7.1: Illustration of a key difference between various models of thin objects. While thin plate theory allows to describe bending (a), it cannot represent more complex deformations such as twist (b) which is captured by shell theory.

Development of a satisfactory physical model that runs in real-time but produces visually convincing animation of thin objects has been a challenge in Computer Graphics, particularly in the area of cloth modelling. Rather than resorting to shell theory which involves the most complex formulations in continuum mechanics, most of works have relied on discrete formulations. We will first discuss early approaches based on mass-spring models. Then we will present techniques relying on the derivation of a bending energy from geometric considerations. At last, we will introduce approaches based on shell theory using the more computationaly demanding finite element method.

7.2 Mass-spring models

Early approaches to thin objects modelling only considered in-plane deformation, and often relied on mass-spring models. This technique was already presented in section 4.3.3 so we only present their use for modelling thin structures. Provot (1995) improved a mass-spring model to take into account the non-elastic properties of woven fabrics. Because of the high stiffness of textiles, mass-spring models are particulary unstable when used to simulate cloth. The author proposed a new method inspired from dynamic inverse procedure. Under certain conditions and because of the linear law employed in a mass-spring system, the fabrics may become super-elastic. Provot suggested to set a threshold on the deformation rate to correct this super-elasticity (see Fig. 7.7).

To improve the computational efficiency, Hutchinson et al. (1996) presented a mechanism for adaptively refining the network of mass-springs to concentrate efforts only where it is needed. When potential inaccuracies are detected, the object is locally refined in the affected region, and the simulation may carry on. Oshita and



(a) Initial position (b) Uncorrected (c) Corrected

Figure 7.2: Deformation of a sheet hanging by two adjacent corners. Images courtesy of Provot (1995).

Makinouchi (2001) used a sparse triangular mesh and an interpolation to generate a dense mesh.

A limitation of mass-spring models that we often hear is the difficulty to derive spring stiffness from elastic properties (Young's modulus and Poisson's ratio). With this drawback in mind, Volino and Magnenat-Thalmann (1997) improved a mass-spring system by allowing the modelisation of Young's modulus and Poisson's ratio (see Fig. 7.3). They also added a correction technique relying on the control of damping and velocities to ensure the stability of their model.

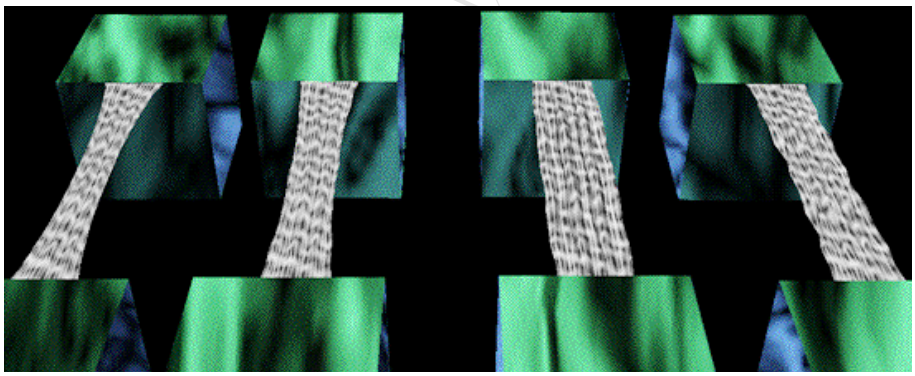


Figure 7.3: A 400 % stretched fabric square with various Poisson's ratio. Image courtesy of Volino and Magnenat-Thalmann (1997).

Other works have considered adding bending through angular springs. For instance, Taskiran and Güdükbay (2005) have chosen to use a linear representation of angular springs to supply bending and torsion effects in hair modelling, which allowed them to simulate curly hair with good realism. All the forces related to springs were implemented: gravity, repulsions from collision, absorption, friction, etc. They could simulate up to a few hundreds of hair stripes in real-time. Wang et al. (2007) successfully used a network of linear and angular springs to describe bending and twisting of catheters and guidewires in an interventional radiology simulator.

More recently and in the field of medical simulation, Hammer et al. (2008) employed a mass-spring model to simulate the mitral valve closure for surgical planning. In their model, triangle sides are treated as linear springs and sides shared by two triangles are treated as bending springs.

7.3 Techniques relying on the derivation of a bending energy

Another kind of approach is to geometrically derive a bending energy. Theses approaches are more computationally demanding than mass-spring models but allows for higher flexibility and accuracy. For instance, Baraff and Witkin (1998) derived stretch, bending and shear energies geometrically. They also used an adaptative time stepping based on detection of large stretches. Using an implicit integration allows the authors to use large time steps and substantially increase the stiffness with a limited impact on performance.

Among the geometry-based models, there is the noticeable work of Grinspun et al. (2003) who introduced a discrete shell model. They designed a simple shell model for triangle meshes by geometrically deriving membrane and bending energies. They applied geometric operators over piecewise-linear surfaces to define a discrete constitutive model, which results in simpler expressions than tensorial derivations from shell theory. They applied their method to simulate the deformation of common objects like beams, sheets of paper or hats (see Fig. 7.4).

Among the different models introduced recently we can also mention the work of Choi et al. (2007) and Bridson et al. (2003). Choi et al. proposed a real-time simulation technique for thin shells undergoing large deformations. The authors adopt the energy functions from the discrete shells proposed by Grinspun et al. (2003). For real-time integration of the governing equation, they adapted a modal analysis technique, called modal warping. The resulting simulations run in real-time even for large meshes, and the model can handle large bending and/or twisting deformations with acceptable realism. Bridson et al. followed the same energy approach to derive their bending model but improved the resolution of the equations by suggesting a novel mixed implicit/explicit integration scheme. In particular, the explicit update for positions allows the authors to modify velocities to enforce constraints and allows a strain limiting procedure which they used to create folds and wrinkles in their cloth simulation. They also presented a post-processing method for treating cloth-character collisions that preserves folds and wrinkles. Pabst et al. (2008) later improved the bending modelling used by Bridson et al. to allow the integration of measured material data.

Volino and Magnenat-Thalmann (2006) suggested an alternative somewhere between mass-spring and geometrically derived approaches. They compute a bending vector that represents the bending of the surface through a simple linear combination of particule positions and they distribute this vector as particule forces according to the bending stiffness. The authors claimed much better accuracy than mass-spring

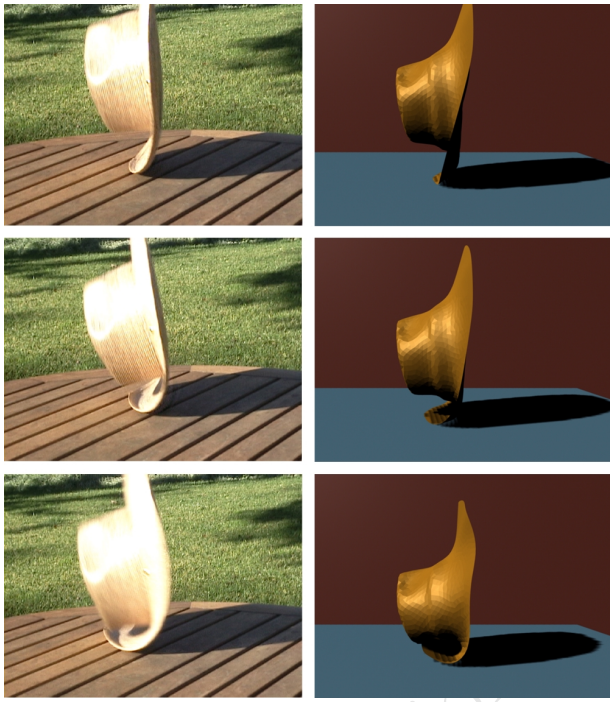


Figure 7.4: Comparison between real footage and simulation of a hat dropped on a table. Image courtesy of Grinspun et al. (2003).

models (particulary when dealing with low curvature and high bending stiffness) with similar computation time (see Fig. 7.5 for an illustration).

A method to use thin shell dynamics with point sampled surfaces for efficient animation was recently proposed by Wicke et al. (2005) where the curvature of the shell is measured through the use of fibers. The fibers are used to approximate the differential surface operators. Their method supports both elastic and plastic deformations as well as fracturing and tearing of the material.

7.4 Techniques based on continuum mechanics

Following a similar train of thoughts than with solid objects, methods based on continuum mechanics provide the most accurate solutions but are computationally very demanding. The mathematical framework of plate and shell theories is also fairly complex and deriving finite element methods from these theories is challenging. Their implementation is also far from being straightforward. Nevertheless, we will now review a few attempts of applying shell theory to simulation.

In 1991, Black et al. presented a materially non-linear shell finite element model to simulate the leaflets of a bicuspid bioprosthetic heart valve. The results indicate that modelling bending has a strong influence on the accuracy of the deformation. They concluded that models where only membrane stresses are taken into account

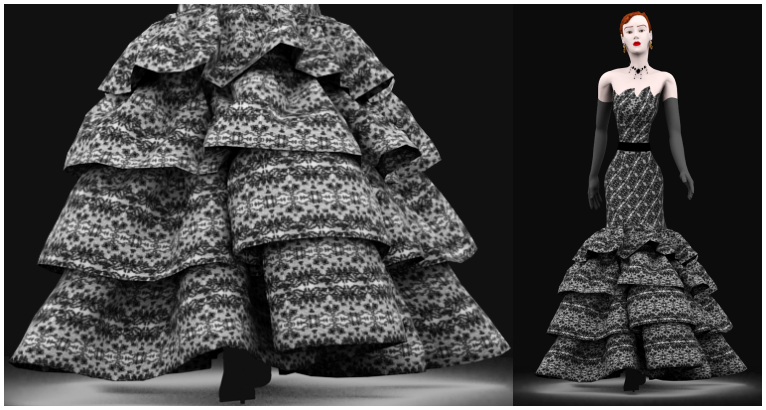


Figure 7.5: The model of Volino and Magnenat-Thalmann allows the simulation of complete garments with material that may have fairly large bending stiffness. Image courtesy of Volino and Magnenat-Thalmann (2006).

were likely to produce significant errors in the stress distribution.

Shell models are fairly common in the field of cloth modelling. For instance, Eischen et al. (1996) applied fully non-linear shell theory for modelling fabrics. The interested reader may also refer to their fairly complete review on the use of shell finite element methods in cloth modelling. However, their attention was towards the accuracy provided by shell theory and were not really interested in fast solution times.

Shell finite element modelling followed the same trend than solid objects and co-rotational formulations were introduced to simplify strain expressions while allowing large rotations (but small strain) (see page 63 for more details). For instance, Etzmuss et al. (2003) used the linear Cauchy strain tensor but applied a polar decomposition on the deformation gradient to allow large deformations. The bending energy is separated from the membrane formulation and most of material parameters were measured from experiments on actual pieces of clothing. As an example, the computation of a shirt consisting of 8 300 triangles took an average of less than 3 s per animation frame with a time step of 0.02 s (see an illustration Fig. 7.6). While still being far from achieving real-time, the intention of the authors was clearly to propose an efficiency model for cloth simulation.

Thomaszewski et al. (2006) based their model on the Kirchhoff-Love shell theory and built a subdivision-based finite element formulation similar to Cirak et al. (2000). They also employed a co-rotational framework to simulate wrinkles and folds of textiles. Although the rotation field is computed for each vertex, they found that using the rotation obtained for the barycentre of all vertices involved was sufficient. More recently, Allard et al. (2009) also used a co-rotational finite element formulation to model the membrane of the lens capsule in a cataract surgery simulator. The goal was to simulate the anisotropic fracture progation during capsulorhexis, the technique used to create a circular opening in the lens capsule. The formulation

(a) (b) (c)

Figure 7.6: A shirt shown with different material properties: (a) wool/viscose, (b) wool and (c) polyester/polyacrylics/acetate. Image courtesy of Etzmuss et al. (2003).

did not feature bending however.

(a) Under gravity (b) Torsion (c) Compression

Figure 7.7: Different types of folds on a garment's sleeve. Images courtesy of Thomaszewski et al. (2006).

Regarding medical applications, some attempts have been made to apply complex shell finite element model to simulate anatomical structures. For instance, Lim et al. (2005) used an asymmetrical shell element model to simulate the deformation of the mitral valve. The computation of the finite element model was carried out with the commercial software ANSYS. Conti et al. (2010) modelled the non-linear and anisotropic mechanical response of the aortic leaflets during the entire cardiac cycle using a shell finite element model in Abaqus FEA. However, these models are not designed for interactive simulation and may need up to dozens of hours to be computed.

7.5 Objectives

We seek to propose a solution for simulating, in real-time, the deformation of thin anatomical structures. Since Black et al. (1991) showed that formulations with only membrane energy were likely to produce significant errors in the stress distribution, this model is required to feature bending. Ideally, this solution would be versatile and allow us to simulate thin objects with various shapes and material properties with good accuracy. In this regard, methods derived from continuum mechanics are the most appropriate. In some situations, a thin plate model would not correctly capture the deformation of some objects and therefore a shell formulation was retained for more flexibility. To the best of our knowledge, no real-time shell finite element method was presented in the field of medical simulation. TODO: need to triple-check this...

In the next section, we will first introduce our co-rotational triangular shell finite element model and show how to efficiently apply this model to simulate the implant deployment in cataract surgery. In a second time, we will present a technique to mesh complex anatomical shapes with curved triangles, hence allowing us to model, in real-time, any anatomical structures with our co-rotational shell FEM in an optimal way.

Chapter 8

TODO: Mass-spring + mapping for colon on GPU?

A short abstract for the upcoming chapter

8.1 Colon is big, we need a fast model: mass-spring on coarse mesh+ mapping for high res mesh

8.2 Limitations... But difficult since it was somewhat sufficient for the colonoscopy simulator.

Chapter 9

A co-rotational triangular shell FEM model

In the field of medical simulation, very few models have been proposed for simulating, in real-time, the deformation of thin anatomical structures. In fact, to our knowledge, no model based on the equations of continuum mechanics was described and applied to simulating the deformation of objects. In this chapter, we propose to rely on continuum mechanics to accurately describe the complex deformations of the implant. Our approach extends the co-rotational method used in finite element analysis of in-plane deformations to incorporate a bending energy. We will first describe the model in details before discussing its implementation in the open source framework SOFA. Lastly, this co-rotational shell FEM was applied to the simulation of the lens deployment during catarct surgery. This simulation also accounts for the complex contacts that take place during the injection phase. This work was the object of two publications (Comas et al., 2010a,b).

9.1 Model description

We propose to define a triangular shell element by combining a two-dimensional in-plane membrane energy, with an off-plane energy for describing bending and twist. To allow for real-time simulation, a computationally efficient formulation is needed. We therefore propose to extend the co-rotational idea introduced by Felippa (2000). Indeed, as we have seen in the previous chapters, co-rotational approaches have been successfully applied to real-time simulation over the last few years. They offer a good trade-off between computational efficiency and accuracy by allowing small deformations but large displacements. We propose to improve and extend a plate model first introduced by Przemieniecki (1985) to a co-rotational formulation. Once combined with an in-plane membrane formulation we obtain an accurate, yet computationally efficient, shell finite element method featuring both membrane and bending energies.

9.1.1 Triangular elastic membrane

The computation of the triangular elastic membrane stiffness matrix can be derived from previous works dealing with tetrahedral co-rotational elements (like Müller and Gross, 2004, for instance). The element stiffness matrix $\mathbf{K}_e$ can be computed as follows:

$$\mathbf{K}_e = \int_v \mathbf{J}\boldsymbol{\chi}\mathbf{J}^T dV, \tag{9.1}$$

where $\mathbf{J}$ is the strain-displacement matrix and $\boldsymbol{\chi}$ embodies the material's behaviour. The implant is very stiff and we therefore assume that the local deformations remain limited during the deployment and a linear constitutive law is sufficient. Thus in the simple case of Hooke's law we have:

$$\boldsymbol{\chi} = \frac{E}{12(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1}{2}(1-\nu) \end{bmatrix} \tag{9.2}$$

The stiffness matrix in the global frame is eventually obtained using the rotation matrix of the element: $\mathbf{K} = \mathbf{R}\mathbf{K}_e\mathbf{R}^T$ where $\mathbf{R}$ describes the rotation of the (triangular) element with respect to its initial configuration.

9.1.2 Triangular plate bending

To calculate the stiffness matrix for the transverse deflections and rotations shown on Fig. 9.1, we start with the assumed deflection u_z of the form

$$u_z = c_1 + c_2 x + c_3 y + c_4 x^2 + c_5 xy + c_6 y^2 + c_7 x^3 + c_8 xy^2 + c_9 y^3, \tag{9.3}$$

where $c_1, \ldots, c_9$ are constants. Equation 9.3 solves an issue of symmetry which was observed with the deflection function proposed by Przemieniecki (1985). These

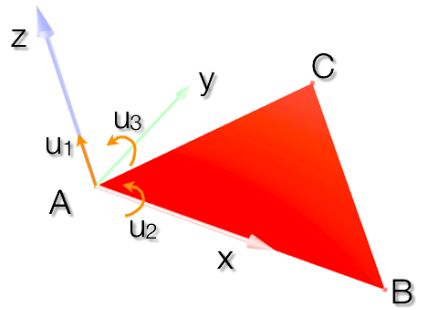


Figure 9.1: The different degrees of freedom u of a triangular thin plate in bending.

constants can be evaluated in terms of the displacements and slopes at the three corners of the triangular plate using

$$\mathbf{u} = \mathbf{C}\mathbf{c} \tag{9.4}$$

where $\mathbf{u} = \{u_1 u_2 \dots u_9\}$ and $\mathbf{c} = \{c_1 c_2 \dots c_9\}$. Matrix $\mathbf{C}$ derives from (9.4):

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & x_2 & 0 & x_2^2 & 0 & 0 & x_2^3 & 0 & 0 \\ 0 & 0 & 1 & 0 & x_2 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -2x_2 & 0 & 0 & -3x_2 & 0 & 0 \\ 1 & x_3 & y_3 & x_3^2 & x_3 y_3 & y_3^2 & x_3^2 & x_3 y_3^2 & y_3^3 \\ 0 & 0 & 1 & 0 & x_3 & 2y_3 & 0 & 2x_3 y_3 & 3y_3^2 \\ 0 & -1 & 0 & -2x_3 & -y_3 & 0 & -3x_3^2 & -y_3^2 & 0 \end{bmatrix} \tag{9.5}$$

We used the following notations:

$$\begin{aligned} u_1 &= (u_z)_{x_1,y_1}, & u_2 &= \left(\frac{\partial u_z}{\partial y}\right)_{x_1,y_1}, & u_3 &= -\left(\frac{\partial u_z}{\partial x}\right)_{x_1,y_1} \\ u_4 &= (u_z)_{x_2,y_2}, & u_5 &= \left(\frac{\partial u_z}{\partial y}\right)_{x_2,y_2}, & u_6 &= -\left(\frac{\partial u_z}{\partial x}\right)_{x_2,y_2} \\ u_7 &= (u_z)_{x_3,y_3}, & u_8 &= \left(\frac{\partial u_z}{\partial y}\right)_{x_3,y_3}, & u_9 &= -\left(\frac{\partial u_z}{\partial x}\right)_{x_3,y_3} \end{aligned} \tag{9.6}$$

(9.7)

We can calculate the strains from the flat-plate theory using:

$$e_{xx} = -z\frac{\partial^2 u_z}{\partial x^2} \tag{9.8}$$

$$e_{yy} = -z\frac{\partial^2 u_z}{\partial y^2} \tag{9.9}$$

$$e_{xy} = -2z\frac{\partial^2 u_z}{\partial x \partial y} \tag{9.10}$$

Hence, using the above equations and (9.3), we have

$$\begin{bmatrix} e_{xx} \\ e_{yy} \\ e_{xy} \end{bmatrix} = -z \begin{bmatrix} 0 & 0 & 0 & 2 & 0 & 0 & 6x & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 2x & 6y \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 4y & 0 \end{bmatrix} \mathbf{c} \tag{9.11}$$

or symbolically $\mathbf{e} = \mathbf{D}\mathbf{c}$ where $\mathbf{D}$ stands for the 3×9 matrix in (9.11), including the pre-multiplying constant $-z$. Noting from (9.4) that $\mathbf{c} = \mathbf{C}^{-1}\mathbf{u}$, we have:

$$\mathbf{e} = \mathbf{D}\mathbf{C}^{-1}\mathbf{u} = \mathbf{b}\mathbf{u}, \tag{9.12}$$

where $\mathbf{b} = \mathbf{D}\mathbf{C}^{-1}$.
Knowing that the stiffness matrix $\mathbf{K}_e$ for an element is obtained from

$$\mathbf{K}_e = \int_v \mathbf{b}^T \boldsymbol{\chi} \mathbf{b} dV, \tag{9.13}$$

where $\boldsymbol{\chi}$ is the material matrix, the substitution of $\mathbf{b}$ into this expression yields

$$\mathbf{K}_e = (\mathbf{C}^{-1})^T \int_v \mathbf{D}^T \boldsymbol{\chi} \mathbf{D} dV \, \mathbf{C}^{-1}. \tag{9.14}$$

Similarly to the membrane stiffness matrix, the bending stiffness matrix in the global frame is eventually obtained using the rotation matrix of the element: $\mathbf{K} = \mathbf{R}\mathbf{K}_e\mathbf{R}^T$.

9.2 Mechanical interactions with the curved surface of shells

The practical interest of modelling complex behaviours such as bending and twisting would remain fairly low for medical simulation if contacts and constraints were not handled properly. In our case the difficulty comes from different sources. First the collision detection must be carried out with the curved surface of shell elements as opposed to the classic detection on plane triangles. Then forces applied to a given triangle need to be distributed between linear forces and torques onto its three vertices. As we will see, the same polynomial interpolation function chosen to compute the bending energy in our FEM formulation is also used to capture the interactions between the curved surface and other objects.

In order to detect the collision with the bent surface, we have chosen the subdivision approach. We first sample the flat surface of each element by recursively dividing each triangle into four smaller ones and the deflection of each new vertex is computed using (9.3) according to the displacements and slopes at the three vertices of the triangular element. This process of subdivision may also allow us to render

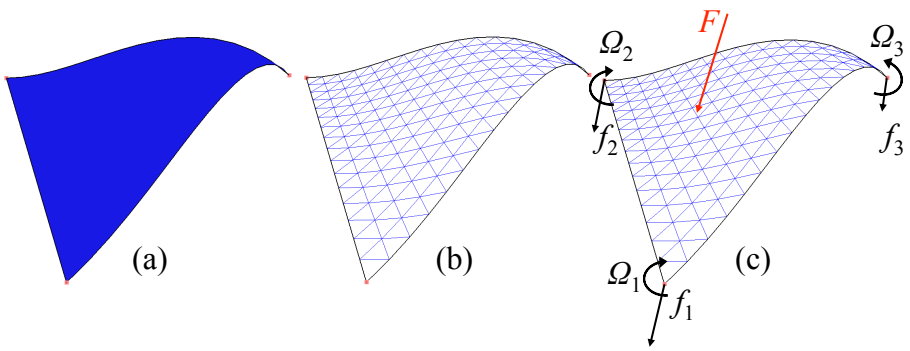


Figure 9.2: (a) The triangle formed by the three vertices of the shell has been recursively subdivided 3 times and the deflection of each new vertex was computed according to the same deflection function used in our shell FEM. (b) Sampling the actual surface of the shell allows more accurate rendering and collision detection. (c) The shape function is used to distribute an external force F onto the triangle nodes.

each shell as a curved triangle (Fig. 9.2 (a) and (b)) and detect any collision with the curved surface of the shell using any of the classic collision detection algorithms working on flat triangles.

Once a collision has been detected, it must be processed by distributing the linear force received on the bent surface between the three vertices of the triangle. First the linear part of the force is simply transmitted on each node using the barycentric coordinates of the contact point's projection onto the triangle.

The main difficulty is to convert the normal component of the force applied to the bent surface into a torque at each of the three nodes (Fig. 9.2 (c)). Our approach is the following: during force computation, we use the change in orientation measured at each node to compute the local deflection of each subvertex within the triangle. Differentiating the formulation twice yields a relation between the torque applied at each node and the generated force in bending. We therefore need to invert the latter formulation to convert a bending force into torques at each vertex. Thus, we are able to transmit any force coming from interactions with the curved surface of shells to the mechanical vertices used in our FEM formulation.

9.3 Implementation

The co-rotational shell formulation has been successfully implemented as a Force Field into the open-source framework SOFA (Allard et al., 2007). A description of SOFA's architecture was given in section 6.3.1. We will now review the key points of the implementation.

9.3.1 Evaluation of displacements

The computation of the stiffness matrix, and subsequently the internal forces, starts with the measure of the displacements $\{u_1 u_2 \dots u_9\}$ for every element. As defined by (9.6), we need to measure the displacement along the z axis and the rotations around local x and y axes for each vertex. For Przemieniecki (1985), the deflection z is measured in the global frame. But in a co-rotational formulation, a local frame is attached to each triangle and used to compute displacements and forces. In this local frame (formed from the 3 vertices) the deflection is therefore always null (hence, u_1, u_4, u_7 are null). To evaluate the remaining rotations, we first associate a quaternion at each vertex which embodies its frame (or its orientation). In mathematics, quaternions are an extension of complex numbers and may be written as follows:

$$q = w + xi + yj + zk \quad \text{with } i^2 = j^2 = k^2 = -1. \tag{9.15}$$

For use in geometry, quaternions are often written as a sum of a scalar and a vector:

$$q = w + (x, y, z) = \cos(\frac{\alpha}{2}) + \mathbf{u}\sin(\frac{\alpha}{2}), \tag{9.16}$$

where $\mathbf{u}$ is a unit vector and α a scalar. If we denote by $\mathbf{v}$ a vector in $\mathbb{R}^3$, it can be shown that the quaternion product $q\mathbf{v}q^{-1}$ yields a vector which is the rotation of $\mathbf{v}$ by an angle α around the axis $\mathbf{u}$. Quaternions require less storage than rotation matrices and are known to be more numerically stable.

From there, our approach to measure the rotations around local x and y axes is simple. At runtime and for each vertex, we compute the rotation that we need to transform the z axis of the triangle's frame into the z axis of the vertex's frame (see listing 9.1). The differences with the same quantities computed for the rest position during the initialisation lead to the desired displacements.

Listing 9.1: Method used to compute the rotation of the z axis between the frames embodied by two quaternions

```
1  inline Quat qDiffZ(const Quat& Qvertex, const Quat& Qtriangle)
2  {
3      // dQ is the quaternion that embodies the rotation between ↩
           the z axis of the vertex and the z axis of the local ↩
           triangle's frame (in local space)
4      Quat dQ;
5
6      // u = z axis of the triangle's frame (in local space)
7      Vec3d u(0,0,1);
8
9      // v = z axis of the vertex's frame is expressed into world ↩
           space
10     Vec3d v = Qvertex.rotate(Vec3d(0.0, 0.0, 1.0));
11
12     // v0 = v expressed into local triangle's frame
```

```
13      Vec3d v0 = Qtriangle.rotate(v);
14
15      // Axis of rotation between the 2 vectors u and v lies into ↩
            the plan of the 2 vectors
16      Vec3d axis = cross(u, v0);
17
18      // Shortest angle between the 2 vectors
19      double angle = acos(dot(u, v0));
20
21      // Quaternion associated to this axis and this angle
22      if (fabs(angle)>1e-6)
23      {
24          dQ.axisToQuat(axis,angle);
25      }
26      else
27      {
28          dQ = Quat(0,0,0,1);
29      }
30
31      return dQ;
32  }
```

9.3.2 Inversion of matrix C

Przemieniecki (1985) described the condition of non-inversion for the matrix **C** as the following: TODO: add condition of inversion... but need the book :)

Each triangle of a mesh obtained from a regular grid of points fits this condition of non-inversion (right triangles). We fixed this limitation by choosing a different local coordinate system that the one introduced by Przemieniecki.

TODO: add figure with the two different coordinate system

9.3.3 Numerical integration

In practice, the integration of the integral in (9.14) must be carried out numerically using Gaussian quadrature, a numerical technique which allows to calculate the value of an integral. The Gaussian quadrature states that the integral of a function f over the area A of a triangular surface may be evaluated as a weighted sum of function values at specific points within the area of integration (Cowper, 1973). In other words, we have

$$\int_A f dA = A \sum_{i=1}^{n} w_i f_i, \tag{9.17}$$

where w_i is a weight factor and f_i is the evaluation of f at the integration point i (also called Gauss points). The choice of the number and the locations of the integration points depends on the order of accuracy desired. We classically choose 3 integration points located at the middle of each edge of the triangle, for which the weight factors w_i are 1/3. The required integration over the volume is achieved

by multiplying the result by the thickness t of the shell. Therefore, (9.14) may be numerically evaluated as

$$\mathbf{K}_e = \frac{At}{3}(\mathbf{C}^{-1})^T \sum_{i=1}^{3} \mathbf{D}_i^T \boldsymbol{\chi} \mathbf{D}_i \, \mathbf{C}^{-1}, \tag{9.18}$$

where D_i is the evaluation of the matrix $\mathbf{D}$ defined by (9.11) at each Gauss point i.

9.3.4 Computation of stiffness matrix: summary

In practical terms, the different computations associated with each triangular shell element can be described as follows:

1. Compute the rotation matrix $\mathbf{R}$ from global to triangle (local) frame
2. Compute the local displacement vector $\mathbf{u} = \{v_1, v_2, 0, u_2, u_3, v_3, v_4, 0, u_5, u_6, v_5, v_6, 0, u_8, u_9\}$ for each triangle, where $(v_1, v_2), (v_3, v_4) and (v_5, v_6)$ are the in-plane displacements on x and y local axes and u_i were defined by (9.6). As we are in a co-rotational framework, we recall that the normal displacements u_1, u_4, u_7 at each of the three nodes are null in the local frame of the triangle.
3. Compute matrix $\mathbf{D}_i$ at each Gauss point i
4. The strain-displacement matrix at each Gauss point i is computed with $\mathbf{J}_i = \mathbf{D}_i \mathbf{C}^{-1}$
5. Compute the local stiffness matrix $\mathbf{K}_e$ of the element as $\mathbf{K}_e = \sum_{i=1}^{3} \mathbf{J}_i \boldsymbol{\chi} \mathbf{J}_i^T$
6. Transform the local element stiffness matrix into the global frame and add it to the global stiffness matrix

9.3.5 Concept of mappings in SOFA

Let us explain the concept of *mappings* and *mechanical mappings* in SOFA. In SOFA, the state of a simulated system can be described by the values and time derivatives of its independent degrees of freedom (DOF) gathered in two vectors x_0 and v_0. Geometry can be attached to the DOF for visualisation or contact computation. We call it the *shape*. It is typically defined by points, such as triangle vertices or sphere centers, and additional data such as triangle connectivity or sphere radii. We call these points the vertices. Their positions, velocities and associated forces are stored in vectors x_1 , v_1 and f_1, respectively. They are not independent variables, since the positions and velocities are bound to the DOF using kinematic operators which we call the *mappings*:

$$\begin{aligned} x_1 &= \mathcal{J}_1(x_0) \\ v_1 &= J_1 v_0 \end{aligned}$$

When the vertices and the DOF are the same, the mapping is the identity. Matrix $J_1 = \frac{\partial x_1}{\partial x_0}$ encodes the linear relation between the DOF velocities and the shape velocities. Due to linearity, the same relation holds on elementary displacements dx. It also holds on accelerations, with an additional offset due to velocities when the position mapping $\mathcal{J}$is nonlinear. In most cases, operators $\mathcal{J}$ and J are the same, but in the case of rigid bodies, $\mathcal{J}$ is nonlinear with respect to x and it can not be written as a matrix.

When shapes collide, additional geometry can be necessary to model the contact. For instance, when an edge intersects another one, a contact force is applied to the intersection points. These points are defined by their barycentric coordinates with respect to their edge vertices. Other relations can be used, depending on the kind of geometrical primitives in contact. This additional geometry requires another geometrical layer connected to the shape by a mapping, as illustrated in Fig. 9.3.

Positions and velocities can be propagated top-down through our layer hierarchy using the relations presented. In order to take the contact forces into account in the dynamics equation, we have to convert the contact forces applied to the contact points to forces applied to the DOF, where our mechanical model is applied. This requires an extension to the position and velocity mappings presented: we call it *mechanical mapping*. This is a general method to propagate forces bottom-up through the layers of the geometrical hierarchy. Given forces f_n applied to a geometry layer n, we derive the equivalent force applied to its parent layer $n-1$. Equivalent forces must have the same power. Thus, we have to compute f_{n-1} such that:

$$v_{n-1}^T f_{n-1} = v_n^T f_n$$

The relation $v_n = J_n v_{n-1}$ allows us to rewrite the previous equation as

$$v_{n-1}^T f_{n-1} = v_{n-1}^T J_n^T f_n$$

Since this relation must hold for any velocity v_{n-1}, we simplify it and get

$$f_{n-1} = J_n^T f_n \tag{9.19}$$

This corresponds to the principle of virtual work.

9.3.6 Visualisation

Because shells have a curved surface, a specific rendering technique is required to visualise the deformation in bending. Without any consideration, the shells would appear flat (as simple triangles). Our idea is to make use of a higher resolution mesh whose deformation would be controlled by the underlying physical mesh of shells. In other words, a mapping was implemented to connect the DOFs of our mechanical model to a set of vertices useful for rendering. In SOFA, a mapping has only 2 methods: init(), which allows computations during the initialisation phase, and apply() called at runtime and in charge of updating the positions of the high resolution mesh according to the coarser mesh constituted by the shell elements.

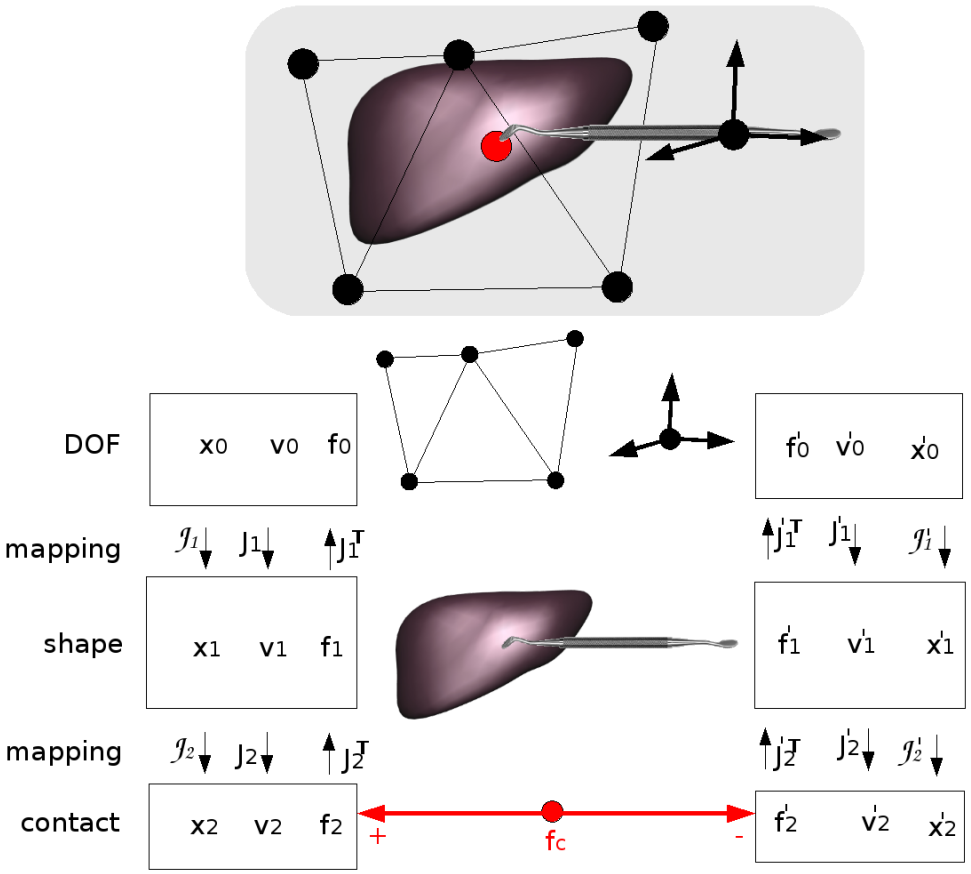


Figure 9.3: Mappings from the DOFs to a contact point. Top: two simulated objects in contact (red point). Bottom: hierarchy of geometrical layers. Positions and velocities are propagated top-down. The contact force f_c is accumulated in the contact layers. Forces are then propagated bottom-up. Image courtesy of SOFA's documentation.

- **init()**
  During the initialisation phase, each vertex from the high resolution mesh is projected onto the coarse mesh. Thus, for each vertex we find the closest primitive (DOF, edge or triangle) on the coarse mesh and we take the list of triangles attached to it. Then, the barycentric coordinates of the projected point is computed within each triangle of this list.

- **apply()** deals with the relation between the DOFs and the vertices used for rendering (shape vertices). The position of each vertex from the high resolution mesh must be updated according to the underlying mesh. After the initialisation phase, we have associated each rendering vertex to a triangle and its barycentric coordinates within this triangle are known. Therefore, we first start by computing the new position of the rendering vertex by weighting the coordinates of each triangle's vertex with the associated barycentric weight. Hence, we obtain a new position in the plane of the triangle. If we were to stop the procedure here, the shell elements would still be rendered as flat triangles since all rendering vertex are located in the plane of the triangles. Consequently, the additional step of computing the deflection is required. We know that the deflection at any point of local coordinates (x, y) may be computed from $u_z = c_1 + c_2x + c_3y + c_4x^2 + c_5xy + c_6y^2 + c_7x^3 + c_8xy^2 + c_9y^3$ as described by (9.3). Hence, we calculate the local coordinates (x, y) of each rendering vertex within its associated triangle. Moreover, we notice that the coefficients $\mathbf{c} = \{c_1 c_2 \dots c_9\}$ may be evaluated using (9.4) from

$$\mathbf{c} = \mathbf{C}^{-1}\mathbf{u}, \tag{9.20}$$

  The deflection may therefore be computed for each vertex and added to the in-plane position previously calculated.

It is worth noting that a high resolution mesh may be obtained by recursively subdividing each triangle. However, in this case the vertices are closely related to the structure of the mechanical mesh and rendering artifacts may appear. As we can notice on Fig. 9.4, the quality of the visualisation may be improved by using a totally different mesh.

9.3.7 Contacts with the curved surface of shells

If a specific rendering technique was developed to visualise the curved shape of the shells, it is also essential to interact with it. Indeed, as explained section 9.2, the force applied onto the curved surface of the shell must be processed by distributing the linear force received on the bent surface between the three vertices of the triangle. Consequently, the mapping previously described is extented to a mechanical mapping and two additional methods need to be implemented: applyJ and applyJT.

- **applyJ()** encodes the linear relation between the DOF velocities and the shape velocities as described in section 9.3.5. applyJ() basically embodies

the same computations than apply() except that the vector of displacement **u** is different. Whereas the local displacement in rotation is measured from the positions to create the vector **u**, this time the vector is directly obtained from the angular velocities we had at each vertex (once converted into the local frame). We therefore have a relation between the angular velocities at each vertex and the velocity of the deflection in the direction normal to each triangle.

- **applyJT()** must convert the contact forces applied to the contact points to forces applied to the DOF. If the linear forces are easily distributed between DOFs using barycentric coordinates, it also encodes the relation between the vertical force **F** applied onto the bent surface of the plate and the torque at each DOF. We start by retrieving the normal component of the applied force vector F_z. We project the application point of the force into the triangle's plane and compute its local coordinates (x, y). We create the polynom P such as

$$P = F_z(1 \;\; x \;\; y \;\; x^2 \;\; xy \;\; y^2 \;\; x^3 \;\; xy^2 \;\; y^3)^T. \tag{9.21}$$

The moments at each vertex are then obtained with

$$\Omega = (\mathbf{C}^{-1})^T P. \tag{9.22}$$

9.3.8 Allowing a curved rest configuration

In Przemieniecki (1985), the rest and initial configurations are both assumed to be the flat one. The only way to have an initial deformed shape is to apply a torque on the vertices of the shells. However, in this configuration we would compute a displacement vector **u** that is not null, which leads to a bending energy. In order to define a curved rest configuration for our shells, we propose to compute the bending energy between the initial and rest configuration during the initialisation phase, and add the energy between the current and initial configurations at runtime. Consequently, by separating initial and rest configurations, it is possible to have an initial deformed shape without any initial bending energy. In other words, we shifted the zero energy configuration.

9.3.9 Possible extension: parallel implementation on GPU

If we look closely at all the steps required to compute the local stiffness matrix for each element (see section 9.3.4 for a summary), we can notice that the operations carried out for a given shell element are independent of the other elements. This feature makes our co-rotational shell FEM implementation highly parallelisable. One potential problem however, is the assembly of the global stiffness matrix. Indeed, in a parallel implementation, many elements are processed concurrently. Inevitably, two elements sharing a node may be processed in the same time and may therefore try to both write their element contribution into the exact same location in the global stiffness matrix, and therefore the same location in GPU's memory. As with

the TLED implementation described in chapter 6, there is two solutions. The first one would be to re-order the elements organisation so that no concurrent writes will occur. Unfortunately, this re-organisation of the element reduces the number of elements that can be processed in parallel. The second solution would be to take advantage of a new GPU NVIDIA architecture called Fermi, which added the capability atomic writes for floats. We recall that the write operation is said to be atomic in the sense that it is guaranteed to be performed without interference from other threads. In other words, the concurrent writes are simply serialised in hardware. Obviously, an additional overhead is to be expected.

To conclude on a possible parallelisation of the algorithm, an optimal GPU implementation is always a challenge to achieve. One needs to choose wisely the type of memory to be used for each variable and optimise the access patterns. However, it seems that there is no major obstacle to a GPU implementation and our co-rotation shell FEM will eventually be ported to GPU.

TODO: enough details?

9.4 Validation

We compared our model with some theoretical results reported by Zhongnian (1986) to assess its quality in modelling bending. The test that we carried out uses a square shape mesh clamped on all four edges. A uniform load is then applied on the square and the maximum deflection z_{max} at the centre can be calculated. Several simulations are performed with increasing load values q (ranging from 1 to 5 $\mathrm{N/m^2}$) and the following parameters were used:

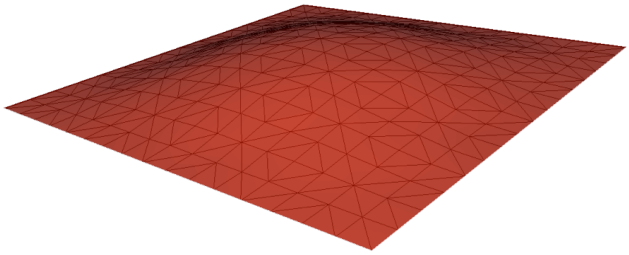
- Young Modulus $E = 1.092 \times 10^6\,\mathrm{Pa}$
- Poisson's ratio $\nu = 0.3$
- edge of the square $L = 10\,\mathrm{m}$
- thickness $h = 0.1\,\mathrm{m}$
- pressure q is a uniformly distributed load per unit area

Using these particular values it can be shown that $z_{max} = 0.126\,q$. The maximum deflection obtained in our simulations are reported in Table 9.1. In average we found $z_{max} = 0.1248\,q$, resulting in a 0.93 % error between our model and theoretical results on that test.

TODO: maybe a bit short for a so-called validation...

9.5 Application to implant deployment simulation in cataract surgery

Cataract surgery consists in three main steps: capsulorhexis, phacoemulsification, and implantation of an intra-ocular lens. Prior to starting the surgery, a viscoelastic



q	z_{max}
1	0.1218
2	0.2475
3	0.3747
4	0.5050
5	0.6374

Table 9.1: Comparison of our shell model with theoretical results on the bending of a square plate. An error of less than 1 % was found between our simulation and theoretical results.

fluid is introduced into the capsule to facilitate capsulorhexis creation and provide protection during phacoemulsification. This fluid remains in the capsule for the duration of the surgery, including the injection of the implant. Capsulorhexis is the technique used to remove a part of the anterior lens capsule. Phacoemulsification consists in using a surgical device which tip vibrates at an ultrasonic frequency to emulsify the natural lens material and then aspirate the fragments of the cortical material. After the removal of the diseased lens, an intra-ocular lens is implanted into the eye, through a small incision (about 2 mm) using a foldable intra-ocular lens (see Fig. 9.5). The foldable lens, usually made of acrylic material, is then implanted within the lens capsule through the incision used during phacoemulsification. In some cases the implant can be flipped or the hooks (also called *haptics*) can break. Therefore the simulation of the insertion and deployment of the implant is crucial for achieving a successful surgery.

To simulate the insertion and deployment of an intra-ocular lens, we first created a triangulation of the lens surface. Particular care was given to the mesh, to ensure that areas where large stresses occur contain a higher density of elements (see Fig. 9.6). This was done by noting the constraints applied by the surgeon to the haptics while inserting the implant within the injection device. During this stage, the haptics are folded onto the implant body, leading to high stresses at the junctions. The lens mesh contains 743 triangles and 473 nodes. Models of the injection device and the entire eye anatomy were also created. Physical parameters of the lens implant have been provided by the manufacturer Alcon and they are presented in Table 9.2.

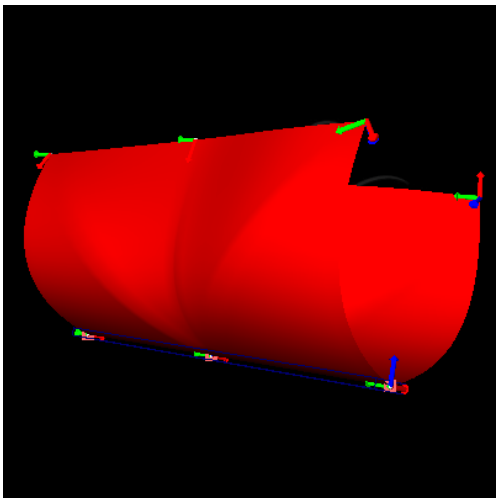
Young's modulus	Poisson's ratio	Mass density
1 MPa	0.42	1.2 g/cm^3

Table 9.2: Physical parameters of the intra-ocular implant (source: Alcon)

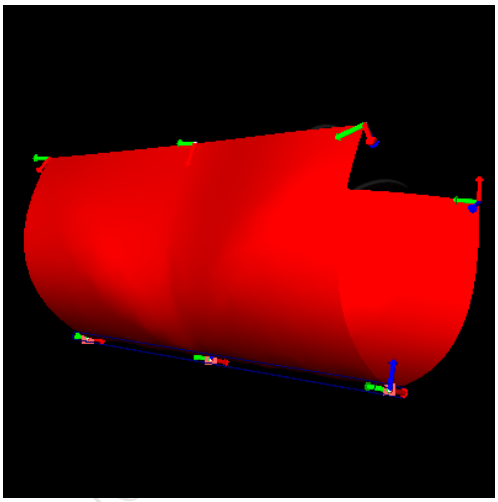
The first difficulty is to obtain the folded geometry of the lens within the injection

device. This step is not important for the training process and does not need to be interactive. Indeed the surgeon does not always have to prepare the implant as some injection devices are readily available with a folded implant already in place. We simulate the folding process by first folding the haptics onto the implant body. The body was then bent while keeping the haptics inside to obtain the shape described in Fig. 9.7. The whole process was carried out by applying the necessary forces and boundary conditions on the body and haptics of the implant. The folded implant was then placed into the injection device. The simulation of the injection consists in pushing the intra-ocular implant within the injection device into the lens capsule. During these stages of the simulation, complex contacts occur and consist of self collisions of the lens as well as collisions between the lens and the injector and later with the capsule. To solve the contacts we use the contact warping method proposed by Saupin et al. (2008) as it offers an efficient way to compute physically correct contact responses in the case of co-rotational models TODO: is it still true? I'm not even sure we used this technique in the final submission to ISBMS . As adhesions between the haptics and the body is often observed in surgery, friction is also taken into account in the contact response process.

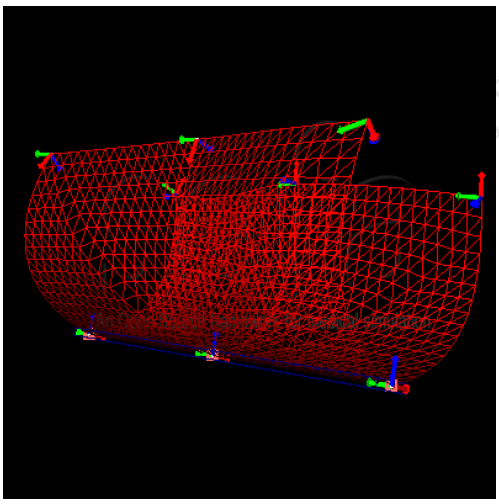
Results of our simulation are illustrated in Fig. 9.8. We can notice the progressive deployment of the implant when it exits the injector. The shape of the intra-ocular lens remains very close to that of a real one during all stages of the simulation: within the injector, during the ejection phase, and when in place within the capsule. Due to the high stiffness and low mass of the lens, a direct sparse solver was used at each time step ($dt = 0.01\,s$) rather than an iterative solver, resulting in a more accurate and more stable simulation, to the detriment of computation times (about 5 FPS for the complete simulation, and about 10 FPS for the deformation only, on a 2.4 GHz processor).



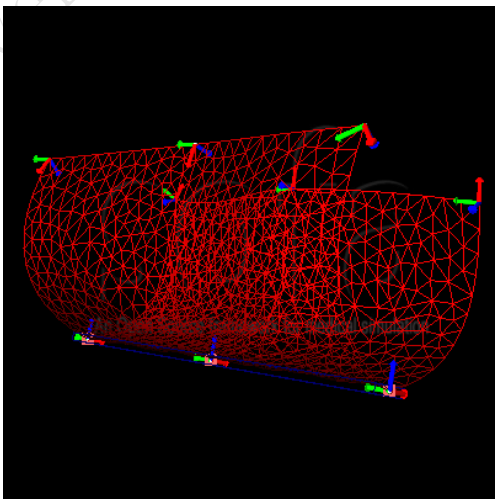
(a) Subdivided triangles



(b) Irregular grid

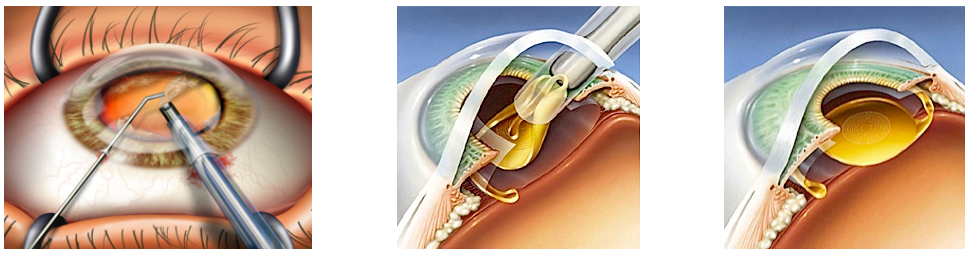


(c) Subdivided triangles



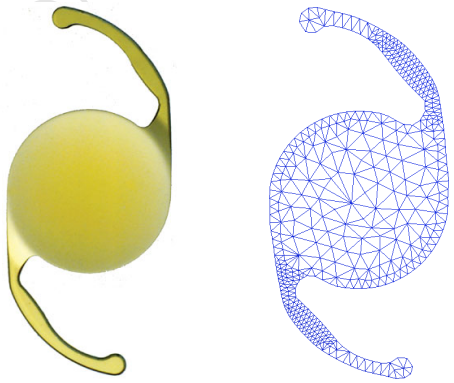
(d) Irregular grid

Figure 9.4: In some situations, some rendering artifacts may appear when we use subdivided triangles to render the shells (a,c). Using an irregular mesh which is not closely related to the structure of the mechanical mesh reduces the impact (b,d).



(a) (b) (c)

Figure 9.5: (a) removal of the opacified lens by phacoemulsification. (b) insertion of the lens implant which is folded inside the injection device and then deploys within the lens capsule. (c) the lens in place in the capsule.



(a) Real implant (b) Meshed model

Figure 9.6: An actual intra-ocular implant (a) and the triangular (b) mesh used in our simulations. Notice the higher density of elements in areas where large deformations will take place. Image of implant courtesy of Alcon.

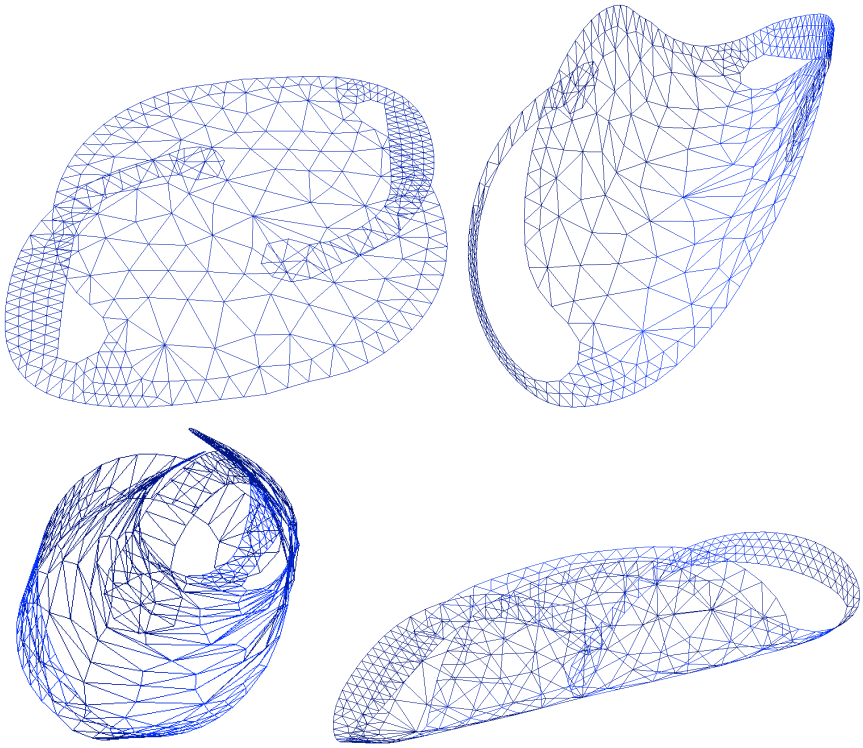


Figure 9.7: Top: intermediary steps of the intra-ocular implant folding. Bottom: fully folded implant ready to be placed into the injection device.

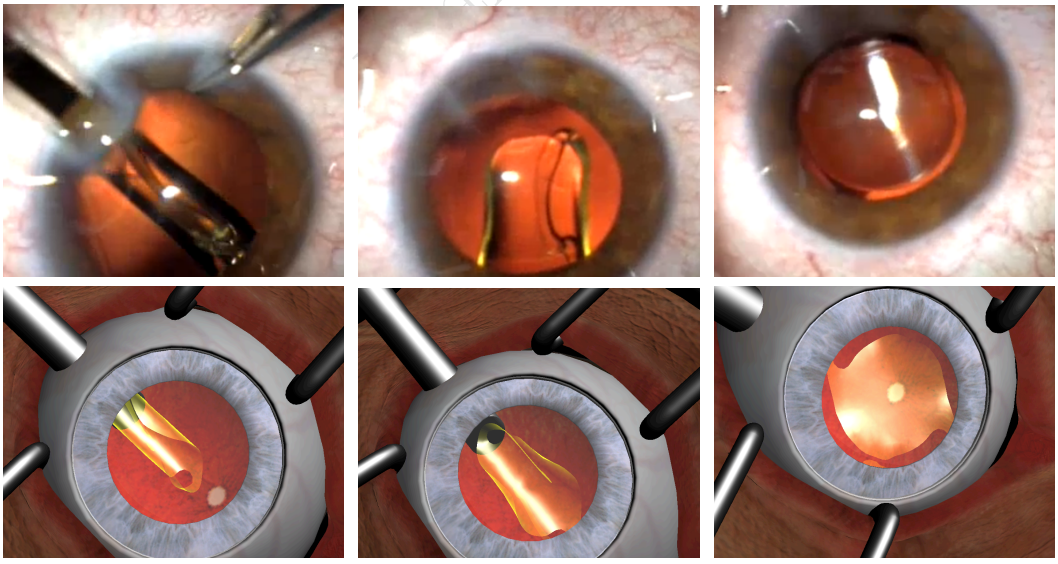


Figure 9.8: Three steps of the simulation of the intra-ocular lens implant injection and its deployment within the lens capsule. Top: images from a real cataract surgery, courtesy of Dr. Tarek Youssef, ophtalmologist in Canada. Below: our simulation of the implant's deployment.

Chapter 10

Physics-based reconstruction using shell elements

In the previous chapter, we proposed a co-rotational formulation for shell elements by extending a classical in-plane triangular finite element approach. This simple shell element can efficiently handle both membrane and offplane forces (bending and twist). The validity of our approach has been demonstrated though comparison with theoretical results. It was then applied it to a rather complex application: the simulation of intra-ocular lens implant deployment in a cataract surgery simulation. These preliminary results are very encouraging and show the potential of such models. Examples include the modelling of hollow anatomical structures (stomach, colon, etc.), the simulation of cardiac valve leaflets, and blood vessels. However, to model the deformation of complex anatomical structures using shell elements, the first step is to describe its surface with curved patches. In the next section, we propose to study how to approximate the surface of anatomical structures with (curved) shell elements. This work was the object of one publication (Comas et al., 2010b).

10.1 Techniques of mesh simplification

10.1.1 Introduction: our motivation

Our idea is the following: while many flat triangles are required to describe highly curved surfaces, fewer triangular shell elements are needed to describe the given geometry with the same precision since they can be curved. Our goal is to create a mesh featuring the optimal number of shell elements while staying as close as possible to our targeted geometry. Literature about volumic mesh generation algorithms is fairly dense. However, there are only a few that are concerned about the generation of meshes over curved surfaces. One of the most widely used techniques for the creation of surface meshes is the plane to surface transformation method (Zienkiewicz and Phillips, 1971), mesh is first generated on a two-dimensional domain and then mapped onto the surface. If this method gives reasonably good meshes on smooth surfaces, the results are usually quite poor with more complex curved surfaces. The finite elements may be generated directly on the curved surfaces based on the advancing front technique (Lo, 1985; Lau and Lo, 1996). The main issue with this approach is that an analytical description of the geometry is needed, which is not the case in medical simulation. Another method consists of using an a priori error estimator to build an adaptive mesh generation (Baumann and Schweizerhof, 1997). However, the tolerance of this indicator should be chosen depending on the desired accuracy of the finite element solution, and therefore requires some knowledge about the problem in order to choose an effective tolerance. Béchet et al. (2002) start from an existing triangular mesh created with a CAD (computer-aided design) software and refine and smooth the mesh based on element quality and surface curvature. While all those methods allow their authors to get satisfactory meshes over curved surfaces according to their needs, they all make use of flat elements to mesh geometries and do not generate actual shells.

In fact, it is interesting to note that this process of meshing a curved surface is similar to the reconstruction of the surface of objects in computer vision. Indeed, calculating curvature maps of 3D surfaces represented as digitised data (point clouds or triangulated meshes) has been extensively studied. One of the most common approach is to use continuous surface patches (Kolb et al., 1995; Douros and Buxton, 2002). The surface is locally approximated by an analytic representation called surface patch, usually chosen to be an implicit surface. Tang and Medioni (1999) presented an elegant and non noise-sensitive approach that infers sign and direction of principal curvatures directly from the input and the authors used this information for coherent surface and curve extraction from 3D data.

However, we need to keep in mind that our situation is substantially different than all previously mentioned geometrical approaches as we want to model the deformation of the structure. In that regard, the curvature of the surface has a physical meaning: it represents the mid-surface of the shell. By keeping this in mind, we propose an approach to mesh a curved surface based on the same polynomial shape function (9.3) used in our co-rotational shell FEM.

10.2 Our method

10.2.1 A simple algorithm

In the following, we assume that we have a high resolution triangular mesh obtained from a binary segmented image of the object we want to simulate (via a Marching Cube algorithm for instance, Lorensen and Cline (1987)). We propose a technique that can simplify the high resolution mesh of the object to be simulated and approximate the surface of anatomical structures with shell elements whose each surface is described by the shape function used in our shell formulation. Thus, the first step in the process of generating a shell-based mesh is an important decimation of the high resolution mesh, using quadric edge collapse technique implemented in Meshlab. MeshLab is an open source program for the processing and editing of unstructured 3D triangular meshes. The algorithm tries as much as possible to preserve mesh boundaries and generates high quality triangular elements. We then apply a heuristic method derived from the work of Saupin et al. (2007) with tetrahedral meshes based on simple geometrical rules. For each node of the coarse mesh, we find the three closest triangles on the high resolution mesh and we move the node to the barycentre of the three centres of mass of those triangles. This technique locally smoothes the surface of the mesh while converging towards the desired high resolution mesh.

At each iteration of this algorithm, we want to measure the error between the curved surface of shells and the target. Indeed, we need to ensure that the distance between the surface of our shell-based mesh and the targeted high resolution mesh will be minimal. An efficient technique for measuring the error between two surfaces is the Hausdorff distance (Klein et al., 1996). As a reminder the Hausdorff distance between two meshes is the maximum between the two so-called one-sided Hausdorff distances:

$$d_{\mathrm{H}}(X,Y) = \max\left\{ \sup_{x\in X}\inf_{y\in Y} d(x,y), \sup_{y\in Y}\inf_{x\in X} d(x,y) \right\}, \tag{10.1}$$

where $d()$ is the Euclidian distance between two points. The same technique of subdivision used for rendering allows us to sample the actual surface described by the shells to compute the Hausdorff distance with the targeted high resolution mesh.

By using the Hausdorff distance between two iterations, the process may be stopped when the required precision has been reached. A simple example is shown Fig. 10.1 to illustrate the method.

10.2.2 Results on complex anatomical structures

Meshing of anatomical structures.

This approach has been applied to approximate more complex anatomical geometries with curved shell elements. Two examples are given with the Glisson's capsule, which is the membrane surrounding the liver (Fig. 10.2) and an aneurysm (Fig. 10.3). In

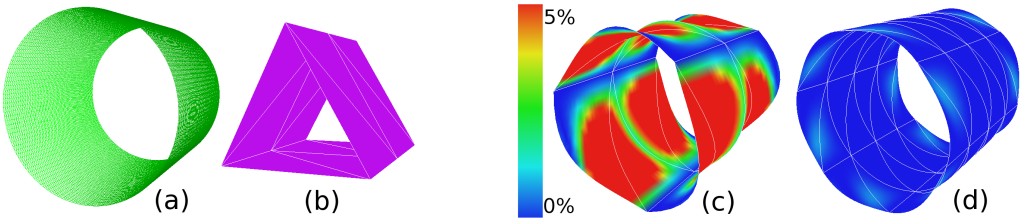


Figure 10.1: The target (a) is a high resolution cylinder mesh of 16,384 triangles and we start from a very coarse mesh (12 triangles), rendered with flat triangles here (b). In (c) the coarse mesh is rendered with shells and a one-sided Hausdorff distance colour map is applied to show the initial error with the high resolution mesh. (d) One-sided Hausdorff distance colour map after one iteration of our algorithm (48 shells).

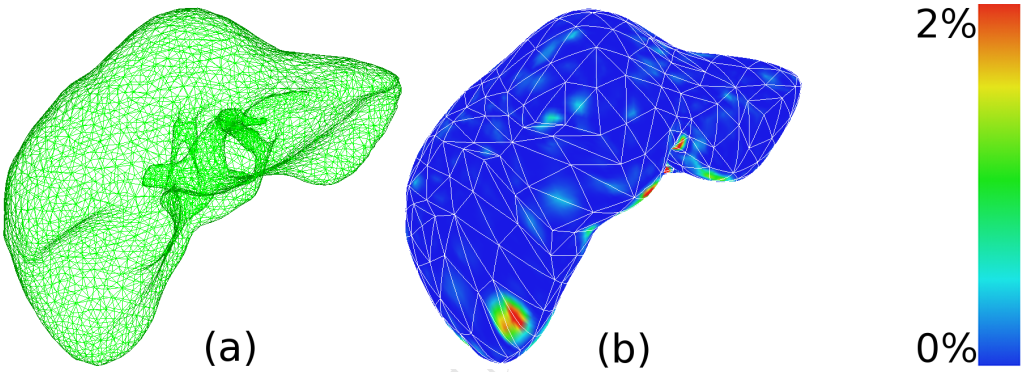


Figure 10.2: (a) the targeteted high resolution Glisson's capsule mesh (8 000 triangles). (b) the one-sided Hausdorff distance error map after applying only one iteration of our algorithm to the coarse mesh (1 200 shells).

each case, the error is computed through a Hausdorff distance and expressed as a percentage of the diagonal of the object's bounding box.

Computation times.

We perform several tests on the aneurysm model at different resolutions to measure computation times. The shells are resisting to a uniform pressure load and solved using a Conjugate Gradient (CG) iterative solver. The computation times are reported in Fig. 10.4. Implicit integration allows for large time steps ($40\,ms$) and the computation is real-time for 800 shell elements and a reasonable error criterion (5 %). When the computation time must be bounded (critical real-time applications), one can fix the number of CG iterations to, for instance, 100 and remains real-time for 1 000 shell elements. However, in that case the accuracy of the results is not checked.

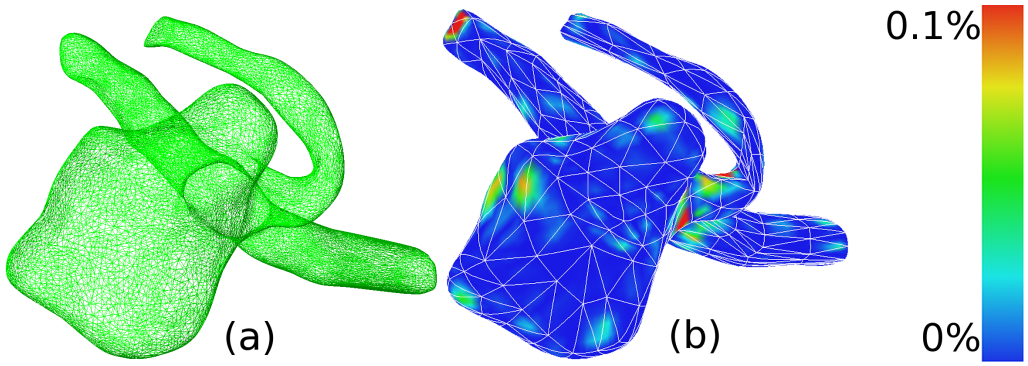


Figure 10.3: (a) the targeteted high resolution aneurysm mesh (28 368 triangles). (b): the one-sided Hausdorff distance error map on a mesh of 772 shells generated with our method.

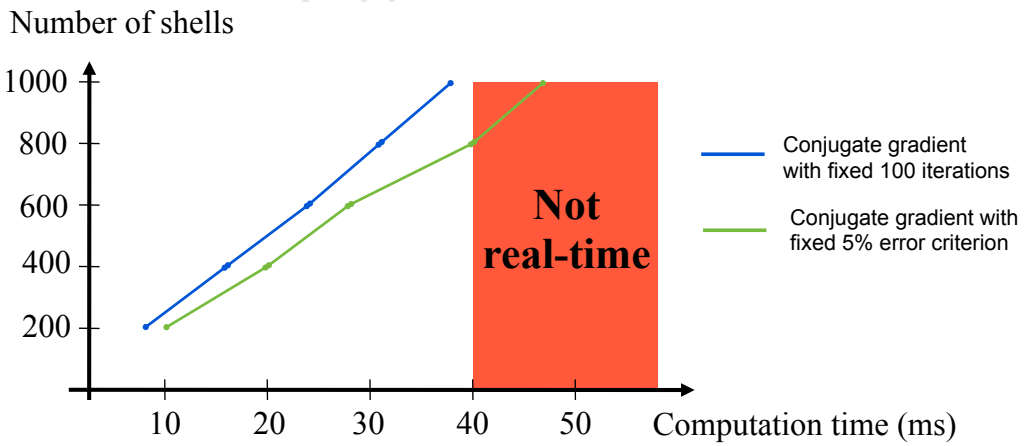


Figure 10.4: Computation time on meshes of 200, 400, 600, 800 and 1 000 elements.

.

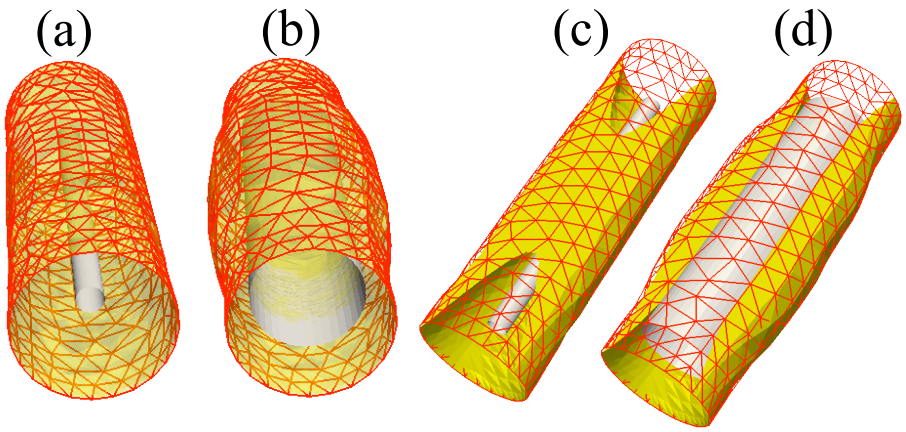


Figure 10.5: Simulation of an angioplasty procedure. (a, c): A collapsed stent is inserted into the blood vessel simulated with our shell FEM formulation. The fatty deposits (in yellow) are modelled with tetrahedral FEM. (b, d): The stent is crushing the fatty deposits which creates a pressure onto the interior wall and widens the blood vessel.

10.2.3 Coupling between tetrahedra and shells for advanced modelling

Structures in human body can be either solid (brain, liver, prostate etc.) or hollow (colon, blood vessels, stomach etc.). However, knowing how to model the two kind of structures is not sufficient to reach a high degree of accuracy, real life situations are more complex. As an example, the external surface of the liver is covered by a layer of cells called Glisson's capsule. Its interaction with the liver plays an important role into the overall structure's mechanical behaviour. Therefore considering the interaction between solid and hollow objects is as crucial as modelling the two structures separately.

An example of medical procedure to illustrate this point even further is angioplasty. Angioplasty is the technique of mechanically widening a narrowed or obstructed blood vessel, typically as a result of atherosclerosis. An empty and collapsed balloon on a guide wire is passed into the narrowed locations and then inflated to a fixed size. The balloon crushes the fatty deposits, so opening up the blood vessel to improved flow. As a proof of concept, we tried to simulate an angioplasty. Our preliminarty results are presented on Fig. 10.5. The blood vessel is modelled using the shell FEM formulation described in this paper and the fatty deposits are simulated with a tetrahedral FEM method and are fixed to the interior wall of the blood vessel. When the balloon inflates it crushes the deposits and they then apply a pressure onto the curved surfaces of shells modelling the interior wall. The forces are then distributed onto the mechanical nodes of the blood vessel mesh as detailed in section 9.2, which widens the blood vessel as expected.

10.3 Discussion

We proposed a framework for real-time modelling of thin anatomical structures. The novelty of our method relies on the combination of a shell finite element formulation and a geometric surface reconstruction both based on the same polynomial interpolation function used to describe the surface of shells. We also show how contacts and interactions with the curved surfaces of shells can be handled using the same function. In summary, these polynomial shape functions are used in three different ways in our computational model:

(a) to approximate complex geometrical shapes

(b) to compute internal forces via a co-rotational shell FEM

(c) to compute contact forces onto a curved triangle

We achieved our objectives to propose a versatile solution which can simulate, in real-time, thin objects with various shapes and material properties with good accuracy. The efficiency of the method was illustrated through shell-based reconstruction and real-time simulation of the deformations of various anatomical structures and other thin objects (like an intra-ocular implant). We also presented preliminary results on the coupling between solid (tetrahedra) and thin objects (shells) for the advanced modelling of anatomical structures via the simulation of an angioplasty procedure.

The finite element procedure that we propose is largely developed based on physical understanding without the use of mathematical shell theories. Indeed, our shell FEM was obtained by simply superimposing a plane stress and bending energies. According to Chapelle and Bathe (2003), this type of approach has a limited level of accuracy. The effective study of shell structures clearly requires a deep physical understanding of shell structural behaviours. The complexity of the physical behaviours of shells requires advanced mathematical analyses from theories that are still in development. The interested reader may refer to the book of Chapelle and Bathe (2003) for more details on shell theory. If the range of application of our shell formulation might be somewhat limited, we did not find out any inconsistencies through our tests. We also need to keep in mind that the accuracy required in medical simulation is not the same than the one demanded in structural analyses for instance. Moreover, the additional complexity of a shell finite element formulation based on true shell theory may also very well jeopardise our real-time constraints. Therefore, we do not think that the use of this simple shell formulation significantly diminishes our contribution to the field of medical simulation. Indeed, to our knowledge, our co-rotational shell FEM is the first description of a shell finite element model applied to simulating the deformation of thin anatomical structures.

Nevertheless, a few improvements are conceivable. Large strain could be allowed by using a non-linear strain tensor and more complex material laws (non-linear, viscoelasticity, etc.) could also be added to our shell formulation. However, those

changes are substantial and their implementations are not straightforward. In addition, it is not clear how the benefits would be the greatest: whether from enhancing our model to a non-linear formulation or completely changing to another model based on true shell theory (where membrane and bending energies are not separated). The answer may depend on the type of constraints applied on the object. Also (as already mentioned section 9.3.9), the real-time capacity of our algorithm could be further improved by implementing our shell FEM on GPU. Although an optimal parallel re-implementation is always challenging, we have already reviewed why there was no major obstacle to a GPU implementation.

Although we were able to mesh complex anatomical structures with a fairly small number of shell elements, our approach suffers from two major drawbacks. In our current implementation, if the error measured with the Hausdorff distance is greater than a threshold (which is also difficult to choose), all triangular elements are subdivised into 4 new triangles and each of the new vertices is moved towards the targeted geometry based on our heuristic algorithm. Obviously, this is not optimal as the error between the two meshes may very well be local and therefore the use of a global mesh refinement may be very limited. The second drawbrack is that we also do not check the quality of the elements during our mesh process. If some elements are stretched from the heavy decimation, these elements are not corrected during the process of remeshing. Our algorithm could even worsen the situation by splitting those stretched elements into four smaller (and more stretched) elements. One solution for the latter issue would be to enhance our algorithm with a relaxation phase where a repulsion force is applied between each vertex (constrained to remain on the surface of the object). This physical approach would allow a more even distribution of the vertices across the surface and would contribute to obtain better quality elements overall. Yet, meshing a surface with curved elements is a very complex problem and still an active area of research. In contrast to all previous geometrical techniques, we suggested a more physical approach consistent with our shell FEM formulation and obtained good results in meshing complex anatomical structures.

Part IV

Conclusion

Chapter 11

Summary

A short abstract for the upcoming chapter

Chapter 12

Discussion and perspectives (Interaction solid/hollow organs)

A short abstract for the upcoming chapter

Appendix A

Tensors TODO: appendix on tensors necessary?

Appendix B

The weighted residual method

The *method of weighted residuals* is an approximation technique for solving differential equations. A subclass of this method, the *Galerkin method of weighted residuals*, is often used to derive the element equations for the finite element method.

Let us suppose that we have a linear differential operator D acting on a function u to produce a function p:

$$D(u(x)) = p(x). \tag{B.1}$$

We wish to approximate u by a function $\tilde{u}$, which is a linear combination of basis functions chosen from a linearly independent set, that is:

$$u \approx \tilde{u} = \sum_{i=1}^{n} a_i \phi_i. \tag{B.2}$$

Now, when substituted into the differential operator D, the result of the operation is not, in general, $p(x)$. Hence, an error or *residual* will exist:

$$R(x) = D(\tilde{u}(x) - p(x)) \neq 0. \tag{B.3}$$

The idea behind the weighted residual method is to force the residual to zero in some average sense over the domain:

$$\int_X R(x) W_i(x) dx = 0, \quad i = 1, 2, \ldots, n \tag{B.4}$$

where the number of *weight functions* W_i is exactly equal to the number of unknown constants a_i in $\tilde{u}$. The result is a set of n algebraic equations for the unknown constants a_i. Depending on the choice for W_i, several sub-methods can be derived: collocation, sub-domain, least squares, Ritz, Galerkin etc. For instance, the weight function used by the Galerkin method TODO: need to check this: Wi are the shape functions? is:

$$W_i = \frac{\partial \tilde{u}}{\partial a_i}. \tag{B.5}$$

References

[Dir, 2010] 2010.
URL http://msdn.microsoft.com/library/ee663275(VS.85).aspx

[Allard et al., 2007] J. Allard, S. Cotin, F. Faure, P. j. Bensoussan, F. Poyer, C. Duriez, H. Delingette and L. Grisoni. *SOFA – an Open Source Framework for Medical Simulation*. In *In Medicine Meets Virtual Reality (MMVR 15*, 2007.

[Allard et al., 2009] J. Allard, M. Marchal and S. Cotin. *Fiber-Based Fracture Model for Simulating Soft Tissue Tearing*. In *Medicine Meets Virtual Reality (MMVR)*, pages 13–18, IOS Press, 2009.

[Aloisio et al., 2004] G. Aloisio, L. De Paolis, A. Mongelli and L. Provenzano. *Artery Soft-Tissue Modelling for Stent Implant Training System*. Journal of Systemics, Cybernetics and Informatics, vol. 2, no. 4, 2004.

[Argyris et al., 1964] J. Argyris, S. Kelsey and H. Kamel. *Matrix Methods of Structural Analysis, A Precis of Recent Developments*. Pergamon Press, 1964.

[Baraff and Witkin, 1998] D. Baraff and A. Witkin. *Large steps in cloth simulation*. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54, ACM, New York, NY, USA, 1998.

[Barbič and James, 2005] J. Barbič and D. L. James. *Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models*. ACM Transactions on Graphics (SIGGRAPH 2005), vol. 24, no. 3, pages 982–990, 2005.

[Barr, 1984] A. H. Barr. *Global and local deformations of solid primitives*. SIGGRAPH Comput. Graph., vol. 18, no. 3, pages 21–30, 1984.

[Basafa and Farahmand, 2010] E. Basafa and F. Farahmand. *Real-time simulation of the nonlinear visco-elastic deformations of soft tissues*. International Journal of Computer Assisted Radiology and Surgery, pages 1–11, 2010.

[Bastard, 2009] C. Bastard. *Elastographie impulsionnelle quantitative : caractérisation des propriétés viscoélastiques des tissus et application à la mesure de contact*. Ph.D. thesis, Université François Rabelais de Tours, 2009.

[Bathe, 1995] K.-J. Bathe. *Finite Element Procedures*. Prentice Hall, second ed., 1995.

[Baudet et al., 2007] V. Baudet, M. Beuve, F. Jaillet, B. Shariat and F. Zara. *Integrating Tensile Parameters in 3D Mass-Spring System*. Tech. rep., LIRIS - CNRS, 2007.

[Baumann and Schweizerhof, 1997] M. Baumann and K. Schweizerhof. *Adaptive mesh generation on arbitrarily curved shell structures*. Computers & Structures, vol. 64, no. 1–4, pages 209–220, 1997.

[Béchet et al., 2002] E. Béchet, J.-C. Cuilliere and F. Trochu. *Generation of a finite element MESH from stereolithography (STL) files*. Computer-Aided Design, vol. 34, no. 1, pages 1–17, 2002.

[Belytschko and Hsieh, 1973] T. Belytschko and B. J. Hsieh. *Non-linear transient finite element analysis with convected co-ordinates*. International Journal for Numerical Methods in Engineering, vol. 7, no. 3, pages 255–271, 1973.

[Belytschko et al., 2000] T. Belytschko, W. K. Liu and B. Moran. *Nonlinear Finite Elements for Continua and Structures*. Wiley, first ed., 2000.

[Bianchi et al., 2004] G. Bianchi, B. Solenthaler, G. Székely and M. Harders. *Simultaneous Topology and Stiffness Identification for Mass-Spring Models Based on FEM Reference Deformations*. In *MICCAI 2004*, vol. 3217, pages 293–301, 2004.

[Biswas et al., 1976] D. Biswas, K. S. Ram and S. S. Rao. *Application of 'natural coordinate system' in the finite element solution of multigroup neutron diffusion equation*. Annals of Nuclear Energy, vol. 3, no. 11-12, pages 465–469, 1976.

[Black et al., 1991] M. M. Black, I. C. Howard, X. Huang and E. A. Patterson. *A three-dimensional analysis of a bioprosthetic heart valve*. Journal of Biomechanics, vol. 24, no. 9, pages 793–795, 797–801, 1991.

[Board et al., 2005] O. A. R. Board, D. Shreiner, M. Woo, J. Neider and T. Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Addison-Wesley Professional, fith ed., 2005.

[Bonnet and Wood, 1997] J. Bonnet and R. D. Wood. *Nonlinear Continuum Mechanics For Finite Element Analysis*. Cambridge University Press, 1997.

[Bourguignon and Cani, 2000] D. Bourguignon and M.-P. Cani. *Controlling Anisotropy in Mass-Spring Systems*. In *Eurographics Workshop on Computer Animation and Simulation*, pages 113–123, Springer-Verlag, 2000.

[Bridson et al., 2003] R. Bridson, S. Marino and R. Fedkiw. *Simulation of Clothing with Folds and Wrinkles*. In *Symposium on Computer Animation*, pages 28–36, 2003.

[Bro-Nielsen and Cotin, 1996] M. Bro-Nielsen and S. Cotin. *Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation*. In *Computer Graphics Forum*, pages 57–66, 1996.

[Chapelle and Bathe, 2003] D. Chapelle and K. J. Bathe. *The finite element analysis of shells - Fundamentals*. Springer, 2003.

[Chen and Zeltzer, 1992] D. T. Chen and D. Zeltzer. *Pump it up: computer animation of a biomechanically based model of muscle using the finite element method*. SIGGRAPH Comput. Graph., vol. 26, no. 2, pages 89–98, 1992.

[Chhatkuli et al., 2009] S. Chhatkuli, H. Kojima, S. Koshizuka, and M. Uesaka. *Mesh-free Simulation of Lung Deformation during Inspiration*. In *World Congress on Engineering and Computer Science*, vol. 2, 2009.

[Choi et al., 2007] M. G. Choi, S. Y. Woo and H.-S. Ko. *Real-Time Simulation of Thin Shells*. Computer Graphics Forum, vol. 26, no. 3, pages 349–354, 2007.

[Chowdhury and Dasgupta, 2003] I. Chowdhury and S. Dasgupta. *Computation of Rayleigh damping coefficients for large systems*. The Electronic Journal of Geotechnical Engineering, vol. 8, no. C, 2003.

[Chui et al., 2007] C. Chui, E. Kobayashi, X. Chen, T. Hisada and I. Sakuma. *Transversely isotropic properties of porcine liver tissue: experiments and constitutive modelling*. Medical and Biological Engineering and Computing, vol. 45, no. 1, pages 99–106, 2007.

[Cirak et al., 2000] F. Cirak, M. Ortiz and P. Schröder. *Subdivision Surfaces: A New Paradigm For Thin-Shell Finite-Element Analysis*. International Journal for Numerical Methods in Engineering, vol. 47, no. 12, pages 2039–2072, 2000.

[Comas et al., 2008] O. Comas, Z. Taylor, J. Allard, S. Ourselin, S. Cotin and J. Passenger. *Efficient nonlinear FEM for soft tissue modelling and its GPU implementation within the open source framework SOFA*. In *Proceedings of ISBMS 2008*, pages 28–39, London, United Kingdom, 2008.

[Comas et al., 2010a] O. Comas, S. Cotin and C. Duriez. *A Shell Model for Real-Time Simulation of Intra-ocular Implant Deployment*. In *Proceedings of ISBMS 2010*, pages 160–170, Phoenix, United States, 2010a.

[Comas et al., 2010b] O. Comas, C. Duriez and S. Cotin. *Shell Model for Reconstruction and Real-Time Simulation of Thin Anatomical Structures*. In *Proceedings of MICCAI 2010*, pages 371–379, Bejing, China, 2010b.

[Conti et al., 2010] C. A. Conti, E. Votta, A. Della Corte, L. Del Viscovo, C. Bancone, M. Cotrufo and A. Redaelli. *Dynamic finite element analysis of the aortic root from MRI-derived parameters*. Medical Engineering & Physics, vol. 32, no. 2, pages 212–221, 2010.

[Coquillart, 1990] S. Coquillart. *Extended free-form deformation: a sculpturing tool for 3D geometric modeling*. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 187–196, ACM, New York, NY, USA, 1990.

[Coquillart and Jancéne, 1991] S. Coquillart and P. Jancéne. *Animated free-form deformation: an interactive animation technique*. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 23–26, ACM, New York, NY, USA, 1991.

[Cotin et al., 1999] S. Cotin, H. Delingette and N. Ayache. *Real-time elastic deformations of soft tissues for surgery simulation.* IEEE Transactions On Visualization and Computer Graphics, vol. 5, no. 1, pages 62–73, 1999.

[Cotin et al., 2000] S. Cotin, H. Delingette and N. Ayache. *A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation.* The Visual Computer, vol. 16, no. 8, pages 437–452, 2000.

[Cowper, 1973] G. R. Cowper. *Gaussian quadrature formulas for triangles.* International Journal for Numerical Methods in Engineering, vol. 7, no. 3, pages 405–408, 1973.

[De et al., 2001] S. De, J. Kim and M. A. Srinivasan. *A meshless numerical technique for physically based real time medical simulations.* In *Medicine Meets Virtual Reality (MMVR)*, vol. 81, pages 113–118, 2001.

[Debunne et al., 2001] G. Debunne, M. Desbrun, M.-P. Cani and A. H. Barr. *Dynamic real-time deformations using space & time adaptive sampling.* In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 31–36, ACM, New York, NY, USA, 2001.

[Douros and Buxton, 2002] L. Douros and B. Buxton. *Three-dimensional surface curvature estimation using quadric surface patches.* In *Proceedings of Scanning*, 2002.

[Eischen et al., 1996] J. W. Eischen, S. Deng and T. G. Clapp. *Finite-Element Modeling and Control of Flexible Fabric Parts.* IEEE Comput. Graph. Appl., vol. 16, no. 5, pages 71–80, 1996.

[Etzmuss et al., 2003] O. Etzmuss, M. Keckeisen and W. Strasser. *A Fast Finite Element Solution for Cloth Modelling.* In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 244, 2003.

[Felippa, 2000] C. A. Felippa. *A systematic approach to the element independent corotational dynamics of finite elements.* Tech. rep., Department of Aerospace Engineering Sciences and Center for Aerospace Structures. University of Colorado, 2000.

[Felippa and Haugen, 2005] C. A. Felippa and B. Haugen. *A unified formulation of small-strain corotational finite elements: I. Theory.* Computer Methods in Applied Mechanics and Engineering, vol. 194, no. 21-24, pages 2285–2335, 2005.

[Flanagan and Belytschko, 1981] D. P. Flanagan and T. Belytschko. *A uniform strain hexahedron and quadrilateral with orthogonal hourglass control.* International Journal for Numerical Methods in Engineering, vol. 17, no. 5, pages 679–706, 1981.

[Fung, 1993] Y. C. Fung. *Biomechanics: Mechanical Properties of Living Tissues.* Springer, second ed., 1993.

[Gent, 2001] A. N. Gent. *Engineering With Rubber: How to Design Rubber Components.* Hanser Gardner Publications, 2001.

[Gibson, 1997] S. F. Gibson. *3D chainmail: a fast algorithm for deforming volumetric objects*. In *I3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 149–154, ACM, New York, NY, USA, 1997.

[Gourret et al., 1989] J.-P. Gourret, N. M. Thalmann and D. Thalmann. *Simulation of object and human skin formations in a grasping task*. SIGGRAPH Comput. Graph., vol. 23, no. 3, pages 21–30, 1989.

[Grinspun et al., 2003] E. Grinspun, A. N. Hirani, M. Desbrun and P. Schröder. *Discrete shells*. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 62–67, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2003, ISBN 1-58113-659-5.

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

[Hammer et al., 2008] P. E. Hammer, D. P. Perrinb, P. J. del Nidob and R. D. Howe. *Image-based mass-spring model of mitral valve closure for surgical planning*. In *Proc. of SPIE Medical Imaging*, vol. 6918, 2008.

[Hauser et al., 2003] K. K. Hauser, C. Shen and J. F. O'Brien. *Interactive Deformation Using Modal Analysis with Constraints*. In *Graphics Interface*, pages 247–256, CIPS, Canadian Human-Computer Commnication Society, 2003.

[Hauth and Strasser, 2004] M. Hauth and W. Strasser. *Corotational Simulation of Deformable Solids*. Journal of WSCG, pages 137–145, 2004.

[Holzapfel et al., 2000] G. Holzapfel, T. Gasser and R. Ogden. *A New Constitutive Framework for Arterial Wall Mechanics and a Comparative Study of Material Models*. Journal of Elasticity, vol. 61, no. 1, pages 1–48, 2000.

[Holzapfel et al., 2002] G. Holzapfel, M. Stadler and C. Schulze-Bauer. *A layer-specific three-dimensional model for the simulation of balloon angioplasty using magnetic resonance imaging and mechanical testing*. Annals of Biomedical Engineering, vol. 30, pages 753–767, 2002.

[Holzapfel, 1996] G. A. Holzapfel. *On large strain viscoelasticity: continuum formulation and finite element applications to elastomeric structures*. International Journal for Numerical Methods in Engineering, vol. 39, no. 22, pages 3903–3926, 1996.

[Horton et al., 2006] A. Horton, A. Wittek and K. Miller. *Towards Meshless Methods for Surgical Simulation*. 2006.

[Horton et al., 2007] A. Horton, A. Wittek and K. Miller. *Subject-specific biomechanical simulation of brain indentation using a meshless method*. In *MICCAI'07: Proceedings of the 10th international conference on Medical image computing and computer-assisted intervention*, pages 541–548, Springer-Verlag, Berlin, Heidelberg, 2007.

[Horton et al., 2010] A. Horton, A. Wittek, G. R. Joldes and K. Miller. *A meshless Total Lagrangian explicit dynamics algorithm for surgical simulation*. International Journal for Numerical Methods in Biomedical Engineering, vol. 26, no. 8, pages 977–998, 2010.

[Hughes, 2000] T. J. R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, 2000.

[Hutchinson et al., 1996] D. Hutchinson, M. Preston and T. Hewitt. *Adaptive refinement for mass/spring simulations*. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, pages 31–45, Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[James and Pai, 1999] D. L. James and D. K. Pai. *ArtDefo: accurate real time deformable objects*. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 65–72, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[James and Pai, 2002] D. L. James and D. K. Pai. *DyRT: dynamic response textures for real time deformation simulation with graphics hardware*. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 582–585, ACM, New York, NY, USA, 2002.

[Ji et al., 2010] W. Ji, A. M. Waas and Z. P. Bazant. *Errors Caused by Non-Work-Conjugate Stress and Strain Measures and Necessary Corrections in Finite Element Programs*. Journal of Applied Mechanics, vol. 77, no. 4, pages 044,504–5, 2010.

[Keeve et al., 1998] E. Keeve, S. Girod, R. Kikinis and B. Girod. *Deformable Modeling of Facial Tissue for Craniofacial Surgery Simulation*. Computer Aided Surgery, vol. 3, pages 3–228, 1998.

[Kerdok et al., 2006] A. E. Kerdok, M. P. Ottensmeyer and R. D. Howe. *Effects of perfusion on the viscoelastic characteristics of liver*. Journal of Biomechanics, vol. 39, no. 12, pages 2221–2231, 2006.

[Klein et al., 1996] R. Klein, G. Liebich and W. Straßer. *Mesh Reduction with Error Control*. In *Visualization 96. ACM*, pages 311–318, 1996.

[Kolb et al., 1995] A. Kolb, H. Pottmann and H. peter Seidel. *Fair Surface Reconstruction Using Quadratic Functionals*. Computer Graphics Forum, vol. 14, no. 3, pages 469–479, 1995.

[Lai et al., 1996] W. Lai, D. Rubin and E. Krempl. *Introduction to Continuum Mechanics*. Butterworth-Heinemann, third ed., 1996.

[Lau and Lo, 1996] T. S. Lau and S. H. Lo. *Finite element mesh generation over analytical curved surfaces*. Computers & Structures, vol. 59, no. 2, pages 301–309, 1996.

[Lautrup, 2005] B. Lautrup. *Physics of continuous matter: exotic and everyday phenomena in the macroscopic world*. Institute of Physics Publishing, 2005.

[Lim et al., 2005] K. Lim, J. Yeo and C. Duran. *Three-dimensional asymmetrical modeling of the mitral valve: a finite element study with dynamic boundaries*. J Heart Valve Dis., vol. 14, no. 3, pages 386–392, 2005.

[Lim and De, 2004] Y. J. Lim and S. De. *On the use of meshfree methods and a geometry based surgical cutting algorithm in multimodal medical simulations*. Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2004. HAPTICS '04. Proceedings. 12th International Symposium on, pages 295–301, 2004.

[Liu and Gu, 2005] G. Liu and Y. Gu. *An Introduction to Meshfree Methods and Their Programming*. Springer, 2005.

[Liu and Quek, 2003] G. Liu and S. S. Quek. *Finite Element Method: A Practical Course*. Butterworth-Heinemann, 2003.

[Lloyd et al., 2007] B. A. Lloyd, G. Székely and M. Harders. *Identification of spring parameters for deformable object simulation*. IEEE Transactions on Vizualisation and Computer Graphics, pages 1081–1094, 2007.

[Lo, 1985] S. Lo. *A new mesh generation scheme for arbitrary planar domains*. International Journal for Numerical Methods in Engineering, vol. 21, no. 8, pages 1403–1426, 1985.

[Lorensen and Cline, 1987] W. E. Lorensen and H. E. Cline. *Marching cubes: A high resolution 3D surface construction algorithm*. SIGGRAPH Comput. Graph., vol. 21, no. 4, pages 163–169, 1987.

[Lubliner, 2006] J. Lubliner. *Plasticity Theory*. MacMillan Publishing Company, revised ed., 2006.

[MacCracken and Joy, 1996] R. MacCracken and K. I. Joy. *Free-form deformations with lattices of arbitrary topology*. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 181–188, ACM, New York, NY, USA, 1996.

[MacDonald, 2007] B. J. MacDonald. *Practical Stress Analysis with Finite Elements*. Glasnevin Publishing, first ed., 2007.

[Magjarevic et al., 2009] R. Magjarevic, J. H. Nagel, J. Sloten, P. Verdonck, M. Nyssen, J. Haueisen, O. Jarrousse, T. Fritz and O. Dösssel. *A modified Mass-Spring system for myocardial mechanics modeling*. In *4th European Conference of the International Federation for Medical and Biological Engineering*, vol. 22, pages 1943–1946, Springer Berlin Heidelberg, 2009.

[Mark et al., 2003] W. R. Mark, R. S. Glanville, K. Akeley and M. J. Kilgard. *Cg: a system for programming graphics hardware in a C-like language*. ACM Trans. Graph., vol. 22, no. 3, pages 896–907, 2003.

[Melvin et al., 1973] J. Melvin, R. Stalnaker and V. Roberts. *Impact injury mechanisms in abdominal organs*. SAE Transactions, , no. 730968, pages 115–126, 1973.

[Meshlab] Meshlab. *MeshLab*. Visual Computing Lab ISTI - CNR, http://meshlab.sourceforge.net/.

[Miller, 1988] G. S. P. Miller. *The motion dynamics of snakes and worms*. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 169–173, ACM, New York, NY, USA, 1988.

[Miller, 2000] K. Miller. *Constitutive modelling of abdominal organs*. Journal of Biomechanics, vol. 33, no. 3, pages 367 – 373, 2000, ISSN 0021-9290.
URL http://www.sciencedirect.com/science/article/B6T82-3Y6Y6WS-D/2/d62ff415a78f2aed9c18401fa2a35cda

[Miller and Chinzei, 1997] K. Miller and K. Chinzei. *Constitutive modelling of brain tissue: Experiment and theory*. Journal of Biomechanics, vol. 30, no. 11-12, pages 1115 – 1121, 1997.

[Miller and Chinzei, 2002] K. Miller and K. Chinzei. *Mechanical properties of brain tissue in tension*. Journal of Biomechanics, vol. 35, no. 4, pages 483–490, 2002.

[Miller et al., 2007] K. Miller, G. Joldes, D. Lance and A. Wittek. *Total Lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation*. Communications in Numerical Methods in Engineering, vol. 23, no. 2, pages 121–134, 2007.

[Misra et al., 2007] S. Misra, A. M. Okamura and K. T. Ramesh. *Force Feedback is Noticeably Different for Linear versus Nonlinear Elastic Tissue Models*. In *WHC '07: Proceedings of the Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 519–524, IEEE Computer Society, Washington, DC, USA, 2007.

[Mor and Kanade, 2000] A. B. Mor and T. Kanade. *Modifying Soft Tissue Models: Progressive Cutting with Minimal New Element Creation*. In *MICCAI '00: Proceedings of the Third International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 598–607, Springer-Verlag, London, UK, 2000.

[Mosegaard and Sørensen, 2005] J. Mosegaard and T. Sørensen. *GPU accelerated surgical simulators for Complex Morphology*. In *IEEE Virtual Reality*, pages 147–153, 2005.

[Mosegaard et al., 2005] J. Mosegaard, P. Herborg and T. Sørensen. *A GPU accelerated spring-mass system for surgical simulation*. In *13th Medicine Meets Virtual Reality. Studies in Health Technology and Informatics*, vol. 111, pages 342–348, 2005.

[Müller and Gross, 2004] M. Müller and M. Gross. *Interactive Virtual Materials*. In *Proceedings of Graphics Interface (GI 2004)*, pages 239–246, 2004.

[Müller et al., 2002] M. Müller, J. Dorsey, L. McMillan, R. Jagnow and B. Cutler. *Stable real-time deformations*. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 49–54, ACM, New York, NY, USA, 2002.

[Müller et al., 2005] M. Müller, B. Heidelberger, M. Teschner and M. Gross. *Meshless deformations based on shape matching*. ACM Trans. Graph., vol. 24, no. 3, pages 471–478, 2005.

[Nava et al., 2008] A. Nava, E. Mazza, M. Furrer, P. Villiger and W. H. Reinhart. *In vivo mechanical characterization of human liver*. Medical Image Analysis, vol. 12, no. 2, pages 203–216, 2008.

[Ng-Thow-Hing and Fiume, 1997] V. Ng-Thow-Hing and E. Fiume. *Interactive display and animation of B-spline solids as muscle shape primitives*. In *Eurographics Workshop on Animation and Simulation*, pages 81–97, 1997.

[Nienhuys and van der Stappen, 2001] H.-W. Nienhuys and A. F. van der Stappen. *A Surgery Simulation Supporting Cuts and Finite Element Deformation*. In *Medical Image Computing and Computer-Assisted Intervention –MICCAI 2001*, vol. 2208, pages 145–152, Springer Berlin / Heidelberg, 2001.

[Oshita and Makinouchi, 2001] M. Oshita and A. Makinouchi. *Real-time cloth simulation with sparse particles and curved faces*. Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings, pages 220–227, 2001.

[Owens et al., 2007] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn and T. J. Purcell. *A Survey of General-Purpose Computation on Graphics Hardware*. Computer Graphics Forum, vol. 26, no. 1, pages 80–113, 2007.

[Pabst et al., 2008] S. Pabst, S. Krzywinski, A. Schenk and B. Thomaszewski. *Seams and Bending in Cloth Simulation*. In *Workshop in Virtual Reality Interactions and Physical Simulation VRIPHYS*, 2008.

[Pathmanathan et al., 2004] P. Pathmanathan, D. Gavaghan, J. Whiteley, S. M. Brady, M. Nash, P. Nielsen and V. Rajagopal. *Predicting Tumour Location by Simulating Large Deformations of the Breast Using a 3D Finite Element Model and Nonlinear Elasticity*. pages 217–224, Springer Berlin / Heidelberg, 2004.

[Pentland and Williams, 1989] A. Pentland and J. Williams. *Good vibrations: modal dynamics for graphics and animation*. SIGGRAPH Comput. Graph., vol. 23, no. 3, pages 207–214, 1989.

[Picinbono et al., 2003] G. Picinbono, H. Delingette and N. Ayache. *Non-linear anisotropic elasticity for real-time surgery simulation*. Graph. Models, vol. 65, no. 5, pages 305–321, 2003.

[Pieper et al., 1992] S. Pieper, J. Rosen and D. Zeltzer. *Interactive graphics for plastic surgery: a task-level analysis and implementation*. In *I3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 127–134, ACM, New York, NY, USA, 1992.

[Poon and Ahmad, 1998] H. Poon and M. F. Ahmad. *A material point time integration procedure for anisotropic, thermo rheologically simple, viscoelastic solids*. Computational Mechanics, vol. 21, no. 3, pages 236–242, 1998.

[Pouliquen et al., 2005] M. Pouliquen, C. Duriez, C. Andriot, A. Bernard, L. Chodorge and F. Gosselin. *Real-Time Finite Element Finger Pinch Grasp Simulation*. In *WHC '05: Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 323–328, IEEE Computer Society, Washington, DC, USA, 2005.

[Prange and Margulies, 2002] M. T. Prange and S. S. Margulies. *Regional, Directional, and Age-Dependent Properties of the Brain Undergoing Large Deformation*. Journal of Biomechanical Engineering, vol. 124, no. 2, pages 244–252, 2002.

[Press et al., 1992] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling. *Numerical Recipes in FORTRAN 77: The Art of Scientific Computing*. Cambridge University Press, second ed., 1992.

[Press et al., 2002] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, second ed., 2002.

[Provot, 1995] X. Provot. *Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior*. In *Graphics Interface '95*, pages 147–154, 1995.

[Przemieniecki, 1985] J. Przemieniecki. *Theory of matrix structural analysis*. McGraw-Hill, 1985.

[Reddy, 1993] J. N. Reddy. *Introduction to the Finite Element Method*. McGraw-Hill, second ed., 1993.

[Reddy, 2007] J. N. Reddy. *An Introduction to Continuum Mechanics*. Cambridge University Press, 2007.

[Rivers and James, 2007] A. R. Rivers and D. L. James. *FastLSM: fast lattice shape matching for robust real-time deformation*. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 82, ACM, New York, NY, USA, 2007.

[Rumpf and Strzodka, 2001] M. Rumpf and R. Strzodka. *Using Graphics Cards for Quantized FEM Computations*. In *Proceedings of IASTED Visualization, Imaging and Image Processing Conference (VIIP'01)*, pages 193–202, 2001.

[Sagar et al., 1994] M. A. Sagar, D. Bullivant, G. D. Mallinson and P. J. Hunter. *A virtual environment and model of the eye for surgical simulation*. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 205–212, ACM, New York, NY, USA, 1994.

[San Vicente et al., 2009] G. San Vicente, C. Buchart, D. Borro and J. Celigüeta. *Maxillofacial surgery simulation using a mass-spring model derived from continuum and the scaled displacement method*. International journal of computer assisted radiology and surgery, vol. 4, no. 1, pages 89–98, 2009.

[Saupin et al., 2007] G. Saupin, C. Duriez and L. Grisoni. *Embedded Multigrid Approach for Real-Time Volumetric Deformation*. In *International Symposium on Visual Computing*, vol. 4841, pages 149–159, Springer Berlin / Heidelberg, 2007.

[Saupin et al., 2008] G. Saupin, C. Duriez, S. Cotin and L. Grisoni. *Efficient Contact Modeling using Compliance Warping*. In *Computer Graphics International Conference (CGI)*, 2008.

[Schein and Elber, 2004] S. Schein and G. Elber. *Discontinuous Free Form Deformations*. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, pages 227–236, IEEE Computer Society, Washington, DC, USA, 2004.

[Schill et al., 1998] M. A. Schill, S. F. F. Gibson, H.-J. Bender and R. Männer. *Biomechanical Simulation of the Vitreous Humor in the Eye Using and Enhanced Chain-Mail Algorithm*. In *MICCAI '98: Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 679–687, Springer-Verlag, London, UK, 1998.

[Sedef et al., 2006] M. Sedef, E. Samur and C. Basdogan. *Real-Time Finite-Element Simulation of Linear Viscoelastic Tissue Behavior Based on Experimental Data*. IEEE Comput. Graph. Appl., vol. 26, no. 6, pages 58–68, 2006, ISSN 0272-1716.

[Sederberg and Parry, 1986] T. W. Sederberg and S. R. Parry. *Free-form deformation of solid geometric models*. SIGGRAPH Comput. Graph., vol. 20, no. 4, pages 151–160, 1986.

[Serby et al., 2001] D. Serby, M. Harders and G. Székely. *A New Approach to Cutting into Finite Element Models*. In *MICCAI '01: Proceedings of the 4th International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 425–433, Springer-Verlag, London, UK, 2001.

[Shewchuk, 1994] J. R. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Tech. rep., Shool of Computer Science - Carnegie Mellon University, 1994.

[Shinya, 2005] M. Shinya. *Theories for Mass-Spring Simulation in Computer Graphics: Stability, Costs and Improvements*. IEICE - Trans. Inf. Syst., vol. E88-D, no. 4, pages 767–774, 2005.

[Sørensen and Mosegaard, 2006] T. Sørensen and J. Mosegaard. *An Introduction to GPU Accelerated Surgical Simulation*. Biomedical Simulation, pages 93–104, 2006.

[Spencer, 1980] A. Spencer. *Continuum Mechanics*. Longman Group Limited, 1980.

[Steinemann et al., 2008] D. Steinemann, M. A. Otaduy and M. Gross. *Fast adaptive shape matching deformations*. In *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 87–94, Eurographics Association, 2008.

[Stiles and Alexander, 1972] R. N. Stiles and D. M. Alexander. *A viscoelastic-mass model for muscle*. Mathematical Biosciences, vol. 14, no. 3-4, pages 343–354, 1972.

[Székely et al., 2000] G. Székely, C. Brechbühler, J. Dual, R. Enzler, J. Hug, R. Hutter, N. Ironmonger, M. Kauer, V. Meier, P. Niederer, A. Rhomberg, P. Schmid, G. Schweitzer, M. Thaler, V. Vuskovic, G. Tröster, U. Haller and M. Bajka. *Virtual Reality-Based Simulation of Endoscopic Surgery*. Presence: Teleoper. Virtual Environ., vol. 9, no. 3, pages 310–333, 2000.

[Talwalkar et al., 2007] J. Talwalkar, M. Yin, J. Fidler, S. Sanderson, P. Kamath and R. Ehman. *Magnetic resonance imaging of hepatic fibrosis: Emerging clinical applications*. Hepatology, vol. 47, no. 1, 2007.

[Tamura et al., 2005] N. Tamura, N. Tsumura, T. Nakaguchi and Y. Miyake. *Spring-bead animation of viscoelastic materials*. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, page 64, ACM, New York, NY, USA, 2005.

[Tang and Medioni, 1999] C.-K. Tang and G. Medioni. *Robust Estimation of Curvature Information from Noisy 3D Data for Shape Description*. In *Proceedings of IEEE ICCV*, 1999.

[Taskiran and Güdükbay, 2005] H. D. Taskiran and U. Güdükbay. *Physically-based Simulation of Hair Strips in Real-Time*. In *WSCG (Short Papers)*, pages 153–156, 2005.

[Taylor et al., 2007a] Z. Taylor, M. Cheng and S. Ourselin. *Real-Time Nonlinear Finite Element Analysis for Surgical Simulation Using Graphics Processing Units*. Medical Image Computing and Computer-Assisted Intervention –MICCAI 2007, pages 701–708, 2007a.

[Taylor et al., 2008a] Z. Taylor, O. Comas, M. Cheng, J. Passenger, D. Hawkes, D. Atkinson and S. Ourselin. *Modelling anisotropic viscoelasticity for real-time soft tissue simulation*. In *Proceedings of MICCAI 2008*, pages 703–710, New York, USA, 2008a.

[Taylor et al., 2009] Z. Taylor, O. Comas, M. Cheng, J. Passenger, D. Hawkes, D. Atkinson and S. Ourselin. *On modelling of anisotropic viscoelasticity for soft tissue simulation: Numerical solution and GPU execution*. Medical Image Analysis, vol. 13, no. 2, pages 234–244, 2009.

[Taylor, 2006] Z. A. Taylor. *A synopsis of the total Lagrangian explicit dynamic finite element algorithm*. Not published, 2006.

[Taylor and Hawkes, 2007] Z. A. Taylor and D. J. Hawkes. *Efficient large strain viscoelastic modelling of soft tissues using explicit dynamic finite element analysis*. Communications in Numerical Methods in Engineering, vol. 00, pages 1–14, 2007.

[Taylor et al., 2007b] Z. A. Taylor, T. B. Kirk and K. Miller. *Confocal arthroscopy-based patient-specific constitutive models of cartilaginous tissues - I: development of a microstructural model*. Computer Methods in Biomechanics and Biomedical Engineering, vol. 10, no. 4, pages 307–316, 2007b.

[Taylor et al., 2008b] Z. A. Taylor, M. Cheng and S. Ourselin. *High-Speed Nonlinear Finite Element Analysis for Surgical Simulation Using Graphics Processing Units*. IEEE Transactions on Medical Imaging, vol. 27, no. 5, pages 650–663, 2008b.

[Terzopoulos et al., 1987] D. Terzopoulos, J. Platt, A. Barr and K. Fleischert. *Elastically Deformable Models*. Computer Graphics, vol. 21, pages 205–214, 1987.

[Terzopoulos et al., 1991] D. Terzopoulos, J. Platt and K. Fleischer. *Heating and melting deformable models*. The Journal of Visualization and Computer Animation, vol. 2, no. 2, pages 68–73, 1991.

[Thomaszewski et al., 2006] B. Thomaszewski, M. Wacker and W. Strasser. *A consistent bending model for cloth simulation with corotational subdivision finite elements*. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 107–116, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2006, ISBN 3-905673-34-7.

[Truesdell and Toupin, 1960] C. Truesdell and R. Toupin. *The classical field theories*. Flügge's Handbuch der Physik, vol. 3, no. 1, pages 226–793, 1960.

[Volino and Magnenat-Thalmann, 1997] P. Volino and N. Magnenat-Thalmann. *Developing Simulation Techniques for an Interactive Clothing System*. In *VSMM '97: Proceedings of the 1997 International Conference on Virtual Systems and MultiMedia*, page 109, IEEE Computer Society, Washington, DC, USA, 1997.

[Volino and Magnenat-Thalmann, 2006] P. Volino and N. Magnenat-Thalmann. *Simple linear bending stiffness in particle systems*. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 101–105, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2006, ISBN 3-905673-34-7.

[Wang et al., 2007] F. Wang, L. Duratti, E. Samur, U. Spaelter and H. Bleuler. *A Computer-Based Real-Time Simulation of Interventional Radiology*. In *The 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEEE-EMBS)*, pages 1742–1745, 2007.

[Wempner, 1969] G. Wempner. *Finite elements, finite rotations and small strains of flexible shells*. International Journal of Solids and Structures, vol. 5, no. 2, pages 117–153, 1969.

[Wicke et al., 2005] M. Wicke, D. Steinemann and M. Gross. *Efficient Animation of Point-Sampled Thin Shells*. In *Eurographics*, vol. 24, 2005.

[Wu and Heng, 2004] W. Wu and P. A. Heng. *A hybrid condensed finite element model with GPU acceleration for interactive 3D soft tissue cutting*. Comput. Animat. Virtual Worlds, vol. 15, no. 3-4, pages 219–227, 2004.

[Wu et al., 2001] X. Wu, M. S. Downes, T. Goktekin and F. Tendick. *Adaptive Nonlinear Finite Elements for Deformable Body Simulation Using Dynamic Progressive Meshes*. In *Computer Graphics Forum*, pages 349–358, 2001.

[Yan et al., 2007] Z. Yan, L. Gu, P. Huang, S. Lv, X. Yu and X. Kong. *Soft tissue deformation simulation in virtual surgery using nonlinear finite element method*. In *The 29th Annual International Conference of the IEEE EMBS*, pages 3642–3645, 2007.

[Zhong et al., 2005] H. Zhong, M. P. Wachowiak and T. Peters. *A real time finite element based tissue simulation method incorporating nonlinear elastic behavior*. Comput Methods Biomech Biomed Engin., vol. 8, no. 3, pages 177–189, 2005.

[Zhongnian, 1986] X. Zhongnian. *A simple and efficient triangular finite element for plate bending*. Acta Mechanica Sinica, vol. 2, no. 2, pages 185–192, 1986.
URL http://dx.doi.org/10.1007/BF02485859

[Zhu et al., 2010] B. Zhu, L. Gu, X. Peng and Z. Zhou. *A Point-Based Simulation Framework for Minimally Invasive Surgery*, vol. 5958, pages 130–138. Springer Berlin / Heidelberg, 2010.

[Zienkiewicz and Phillips, 1971] O. C. Zienkiewicz and D. V. Phillips. *An automatic mesh generation scheme for plane and curved surfaces by isoparametric co-ordinates*. International Journal for Numerical Methods in Engineering, vol. 3, no. 4, pages 519–528, 1971.