Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Improving Realism of a Medical Surgery Simulator: From Linear to Quadratic Interpolation and its Significance in the Cutting Problem

Tesis presentada para optar al título de
Licenciada en Ciencias de la Computación

Sabrina Izcovich

Directores: Christoph Paulus y Alejandro D. Otero

Buenos Aires, 2018

En la actualidad, la simulación pre-operativa comienza a ganar terreno, y su correcto funcionamiento en tiempo real [10] cobra cada vez mayor importancia a la hora de querer estudiar posibles comportamientos de los órganos en los quirófanos [18]. Es por esto que se vuelve fundamental conseguir una representación acertada de los mismos a medida que son sometidos a deformaciones.

Basado en los modelos matemáticos y físicos de deformación de objetos, la mayoría de los simuladores quirúrgicos utilizan actualmente el método de los elementos finitos (FEM) [9]. En el caso de la simulación de cortes en cirugías, la mayoría de los métodos publicados requieren de un reajuste de las topologías de malla [19] con el fin de alinear los bordes de los elementos con el corte.

Existen diversos tipos de elementos finitos para modelización. En particular, la utilización de elementos lineales requiere de una gran cantidad de los mismos para lograr una descripción realista de los contornos. Esto produce un gran retardo a la hora de calcular la posición de los elementos luego de la deformación, empeorando la performance de la simulación en tiempo real. Por otro lado, al momento de querer realizar cortes curvos, los elementos lineales no otorgan una precisión rigurosa. El uso de elementos cuadráticos debería conllevar a mejoras dado que se espera que se alcance una alta precisión con menos elementos [11].

El trabajo de tesis se basa en mejorar la simulación de tejidos blandos con el fin de que ésta sea lo más cercana posible a la realidad, sin empeorar la performance del simulador. El mismo se basa en la implementación de elementos cuadráticos (en particular, tetraedros) y su utilización en los objetos a modelar en 3D. Por otro lado, se implementó la adaptación de los métodos necesarios para el reajuste de las topologías de malla a elementos cuadráticos con el fin de lograr cortes de órganos modelados con los mismos.

Si bien existen numerosos *frameworks* dedicados a la representación y simulación de objetos, nos centramos en SOFA[1] [8]. Éste fue desarrollado por el INRIA para simulación y modelización física interactiva [12], y es de libre distribución. En particular, se trabajó con tres plugins: *Quadratic Tetrahedra* para la representación de elementos cuadráticos, *CGoGN* para la incorporación de tetraedros cuadráticos a la más reciente topología de SOFA y *Cutting Plugin* para la preparación de los objetos para cortes [20].

Las implementaciones y adaptaciones realizadas fueron luego estudiadas analíticamente para comprobar su correctitud y permitieron comprobar, por medio de distintos análisis, que las deformaciones de elementos curvos representadas con elementos cuadráticos se asemejan más a la realidad que las realizadas con elementos lineales para la misma cantidad de elementos. Luego, pudimos observar que la cantidad de elementos necesaria para una simulación acertada de órganos y cortes de tejidos en cirugías es menor cuando éstos son cuadráticos que para el caso de lineales. Esto último nos permitió concluir que el uso de elementos cuadráticos presenta una mejora para la simulación de cirugías médicas en tiempo real sin empeorar su performance.

Palabras claves: Simulación, Elementos Cuadráticos, Cirugías, Tetraedros, SOFA.

[1] https://www.sofa-framework.org/

In the last decades, the use of minimally invasive surgery has grown exponentially due to its benefits for the patients in term of haemorrhaging risk reduction and shortened recovery time, but it remains complex from a surgical point of view because of the reduced field of view which considerably impacts depth perception and surgical navigation. Nowadays, preoperative simulation begins to gain ground, and its correctness in real time [10] is crucial while studying possible organs behaviors in operating rooms [18]. That's why it's fundamental to find a good representation of them while being deformed.

Based on the mathematical and physical models of objects deformation, most surgical simulators use nowadays the method of finite elements (FEM) [9]. In the case of cutting in surgeries simulations, most of the published methods require a readjustment of the mesh topologies [19] for aligning the element's borders with every cut.

While using linear elements for modelization, it's necessary to add a lot of them to get a realistic description of their outlines. This can induce a big retardation while calculating the element's position after a deformation, blowing the simulation's performance in real time. On the other hand, while trying to do curved cuts, the linear elements don't answer to a rigorous precision. The use of quadratic elements should bring improvements due to his high precision with less elements [11].

The thesis work is based on the improvement of the soft-tissues simulation in order to make it as real as possible, without deteriorating the simulator's performance. The goal of it consisted of the quadratic elements implementation (in particular, tetrahedra) and its utilization in 3D modelated objects. On the other hand, all the necessary method's adaptations were implemented to readjust the mesh topologies to quadratic elements to get cut organs modelated with them.

Even if many objects representation and simulation *frameworks* exist nowadays, we focus in SOFA[2] [8], developed by INRIA for interactive physical simulation and modelization [12], and is open-source. In particular, we worked with three plugins: *Quadratic Tetrahedra* for the representation of quadratic elements, *CGoGN* for the incorporation of quadratic tetrahedra to the latest SOFA topology and *Cutting Plugin* for the preparation of objects to be cut [20].

The implementations and adaptations made were then analyzed analytically to verify their correctness and allowed to verify, by means of different analyzes, that the deformations of curved elements represented with quadratic elements resemble reality more than those made with linear elements for the same amount of elements. In addition, we observed that the amount of elements necessary for a successful simulation of organs and tissue cuts in surgeries is less when they are quadratic than in the case of linear. The latter allowed us to conclude that the use of quadratic elements presents an improvement for the simulation of medical surgeries in real time.

Keywords: Simulation, Quadratic Elements, Surgery, Tetrahedra, Modelling, SOFA, Medicine.

[2] https://www.sofa-framework.org/

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank everyone involved in my career since the very early stages.

In particular, I would like to point out a few people who supported me in crucial ways.

Christoph Paulus, my director, for his dedication and patience. For answering each and every one of my emails, always in a positive way and motivating me with his constant follow-up. Without his help, this would not have been possible.

Alejandro D. Otero, my director, for his time, for being attentive and patient, and for helping me complying within my timeframes.

My family, my parents and my siblings, for their love, for supporting me in the most difficult times.

Matías García Marset, for always being there for me, for listening and accompanying me throughout my career.

Lautaro Petaccio, for being my right hand, for helping me whenever I needed it, advising me and, in particular, nurturing my curiosity.

Francisco Dorr, for coming along with me in the final leg.

Martina Faverio, for helping me make beautiful figures.

All my school friends for making this path much more easy-going.

My close friends Camila Salazar, Agustina García and Victoria Cravino, for being my ground wire.

My professors, for their constant help and their willingness to share their knowledge.

The University of Buenos Aires, for changing my entire way of thinking and giving me everything, including the possibility of teaching.

CONTENTS

INTRODUCTION

In the last decades, surgery is experiencing stronger mechanization. In addition to the development of diagnostic devices that help to reduce the need for surgery, exercise, planning and assistance software become more relevant. For example, the use of minimally invasive surgery has grown exponentially due to its benefits for the patients in term of haemorrhaging risk reduction and shortened recovery time.

In order for the software to be used in surgery, the three-dimensional simulation of soft tissue deformations with a realistic view of sections under the influence of external and internal forces and marginal displacements is required.

In addition to the theoretical training of physicians, such an exercise software with sectional simulations can strengthen the practical training with realistic situations early on. In particular, computer-based training systems offer an elegant solution to the current need for better training in Medicine, since realistic and configurable training environments can be created. This can bridge the gap between basic training and performing the actual intervention on patients, without any restriction for repetitive training.

The combination of visual and haptic perception in surgery is being trained and the surgeons have already performed many virtual operations of this kind prior to a real operation.

With planning systems that include three-dimensional simulations, critical situations can already be identified, discussed, and audited before an intervention. They allow the planning of the procedure during operations and thus an optimized intervention time on the patient. The patient is exposed to less stress and thereby hope for medical professionals and researchers hope for a faster recovery.

Assistance systems can warn of wrong cuts (for example, through vessels) during surgery and prevent mistakes. If complications occur, the simulation software helps to quickly and efficiently prevent the right action.

Therefore, modeling soft tissue by means of continuum mechanics has become an active area of research in many fields such as patient-specific surgery simulation, non-rigid image registration and cardiovascular diagnostics.

The systems mentioned are constantly evolving. The simulation of a cut is and remains an open challenge, since existing methods suffer from stability or runtime problems, and organ motion due to respiration and contact with surgical instruments can significantly degrade the accuracy of image-guided surgery.

In a simulation of the deformation of a body, it must first be discretized. The body is replaced by a finite number of nodes at which the deformation is calculated. For this there are two different approaches: First, the finite element method, which connects the nodes with elements and applies the material properties to these elements. Second, the so-called mesh free methods, which work without a mesh and apply physical properties directly to the nodes.

In this thesis, we seek to improve the representation of organs and simulation of their cuts in the SOFA[3] framework. To achieve it, we make use of the finite element method since it can be used to calculate the deformation of any body under load. Therefore, in order to be able to simulate cutting problems with the FEM, the elements must be adapted in a remeshing process.

The objective of this work is to incorporate the quadratic elements[4] (tetrahedra) into SOFA in order to improve organs simulation in the context of medical operations. Since a quadratic approximation should be more accurate than the linear one with fewer elements [14], we adapted an implementation of the quadratic tetrahedra plugin to represent objects using a quadratic mesh. Beyond that, we incorporated the quadratic tetrahedra in the CGoGN Plugin[5] [15] in order to be able to cut quadratic objects by using the Cutting Plugin. Then, we modified the remeshing functions to prepare objects to be cut in real-time.

Within the results of this thesis, the convergence analysis of the quadratic tetrahedra incorporated in SOFA once added to CGoGN and the evaluation of the remeshing before cutting them can be found.

Outline

In the **first chapter**, we lay the physical basis of this work. We begin with the introduction of the elasticity theory, including the concepts of the strain and the stress; and the stress-strain law. Then, we formulate the cutting problem in the differential - *strong* - and variational - *weak* - form.

In the **second chapter**, we explain the finite element method in a generic way. To do so, we introduce the definitions of **element**, **node** and **degree of freedom**. Then, we present the finite element discretization of the formulations exhibited in the first chapter. Finally, we present the linear and quadratic tetrahedra shape functions in order to be able to apply the finite element method to those types of elements.

In the **third chapter**, we describe the simulation open framework architecture, its functionality and its basic structure. The CGoGN combinatorial map is also introduced in this chapter due to its important role throughout this thesis.

In the **fourth chapter**, we outline the methods used to implement quadratic elements in the second and third dimensions in space, in Python and C++ respectively. Then, we explain the CGoGN adaptation pursued in order to introduce quadratic elements without

[3] https://www.sofa-framework.org/api/SOFA/index.html

[4] Elements with shape functions using quadratic polynoms

[5] CGoGN (Combinatorial and Geometric modeling with Generic N-dimensional Maps) is a C++ library that provides an efficient index-based implementation of n-dimensional combinatorial maps.

including a new topology.

In the **fifth chapter**, we present the cutting procedure and its previous remeshing process with the quadratic incorporation. The methods are explained in the second and third dimensions in space.

In the **sixth chapter**, we exhibit the results. We begin by analyzing the implemented methods convergence. Then, we compare the linear and quadratic beams deformations convergence. Finally, we show a visual comparison between linear and quadratic representations, and an example of a liver cut.

We finish this thesis by summarizing the work done, presenting relevant conclusions and pointing out possible future directions.

1. THEORY OF ELASTICITY

In order to achieve an efficient computation and simulation of the deformations of elastic bodies as a reaction to external forces and to cuts, an interdisciplinary approach has to be taken. In this chapter, ideas from fields of mechanical engineering, computer science and mathematics are combined to introduce the key concepts for a better understanding of this work.

The theory of elasticity considers the deformation of elastic bodies under the influence of external forces. As it sees the bodies as a continuum, it belongs to the continuum mechanics [13]. With regard to the deformation of a body, elasticity is the property of a substance to react to any external force in such a way that the process is reversible [17]. This means that an elastic body returns to its pre-stressed state after being deformed by a force.

In this case, we restrict detailed examination to elastic isotropic materials. Also, we make the following material behavior assumptions:

- Macroscopic Model: The material is mathematically modeled as a continuum body.
- Elasticity: The stress-strain response is reversible and consequently the material has a preferred initial state of stable equilibrium. By convention we assume that the material is unstressed and undeformed. On applying loads, and possibly temperature changes, the material develops nonzero stresses and strains, and moves to occupy a deformed configuration.
- Linearity: The relationship between strains and stresses is linear. Doubling stresses doubles strains, and viceversa.
- Isotropy: The properties of the material are independent of the direction.
- Mass conservation: We assume that the mass stays the same during a motion.

The following section briefly outlines the mathematical and physical notions underlying our soft body simulator.

1.1 Kinematics

In this section, we analyze the variables that interfere in a body displacement.

We consider a continuum body B that moves and deforms in space, changing its configuration dependent on the time t. Hereby, Ω_0 is the reference (or undeformed) configuration ($t = 0$) and Ω is the current (or deformed) configuration. Points in the reference configurations are denoted as $\mathbf{X}$, while points in the current configurations are written as $\mathbf{x}$. We assume a uniquely inversible vector field $\boldsymbol{\varphi}$, called motion, that relates each position in the reference to a position in the current configuration [22].

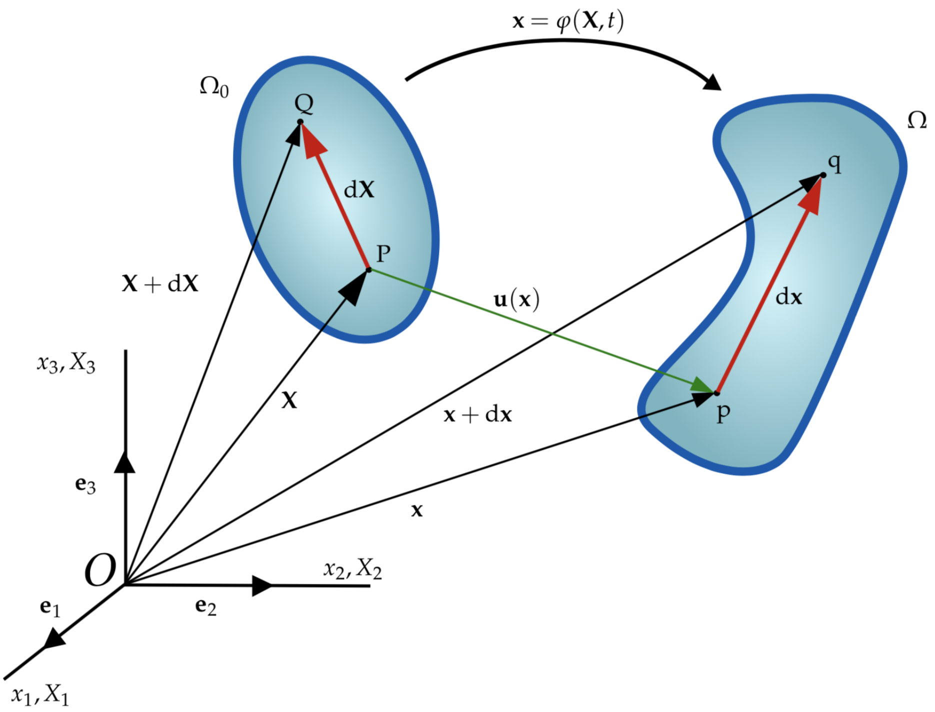


Fig. 1.1: Deformation of the body B with material points P, Q and infinitesimal line element $\mathbf{dX}$ from the reference configuration Ω_0 to the current configuration Ω. Figure extracted from [21]

In Figure 1.1, the deformation of a point P or $Q \in B$, from position $\mathbf{X}$ in the reference configuration to position $\mathbf{x}$ in the current configuration, can be perceived.

In elasticity theory, the body is seen as a continuum with reversible deformation, which is a deterministic approach. The *deformation* $\boldsymbol{\varphi}$ is defined in 1.1 using the *displacement function* $\boldsymbol{u}$, which is sufficient for the calculation since the initial state is already known as shown in Figure 1.1 [16].

Quantities in relation to the deformed state of the body (e.g. strain) are commonly expressed in terms of a fixed reference configuration $\Omega_0 \supset \mathbb{R}^3$. The configuration mapping $\boldsymbol{\varphi} : \Omega_0 \times [0, T]$ transforming material particles from their reference positions $\mathbf{X}$ to current positions $\mathbf{x}$ can be written as [10]

$$\mathbf{x}(t) = \boldsymbol{\varphi}(\mathbf{X}, t) = \mathbf{X} + \boldsymbol{u}(\mathbf{X}, t) \tag{1.1}$$

where $\boldsymbol{u}$ is a displacement field from the initial configuration.

For later use, we define the deformation gradient $\nabla\boldsymbol{\varphi}$ as

$$\boldsymbol{\nabla}\boldsymbol{\varphi} = \frac{d\mathbf{x}}{d\mathbf{X}} = \mathbf{I} + \boldsymbol{\nabla}\boldsymbol{u} \tag{1.2}$$

The deformation function $\boldsymbol{\varphi}$ should keep the orientation of the normal of the surfaces and must not destroy any volume, so $det(\boldsymbol{\nabla}\boldsymbol{\varphi}) > 0$ has to be valid.

We start by presenting three important notions to define the main principles of the elasticity problem: a dimensionless deformation measure, called strain ε, the force per unit area reprented by the stress σ, and a material law relating the two to each other. Then, we introduce the elasticity problem to be considered: the deformation of a continuum body and its differential and weak formulation while subjected to a cut.

1.2 The strain

To illustrate the strain, we consider the example of a rod of Figure 1.2 with initial length L which is stretched to a length L'.

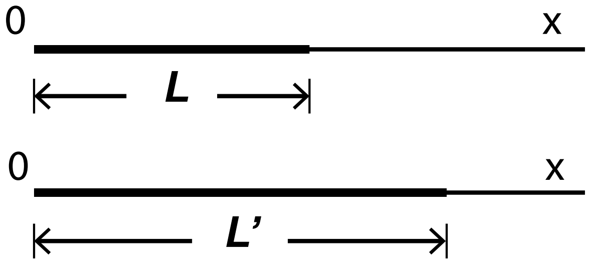


Fig. 1.2: Rod before and after being stretched. Figure inspired from [2]

The strain measure ε, a dimensionless ratio, is defined as the ratio of elongation with respect to the original length [2]:

$$\varepsilon = \frac{L' - L}{L} \tag{1.3}$$

Under the action of applied forces, solid bodies change in shape and volume and its deformation is described mathematically in the following way: every component X_i will have a different value x_i after the deformation:

$$u_i = \mathrm{x}_i - \mathrm{X}_i$$

The vector $\boldsymbol{u}$ is called the *displacement vector* [1]. As the coordinates x_i are functions of the coordinates X_i, then the displacement u_i is also a function of the coordinates X_i.

We consider an arbitrary point P in the bar shown in Figure 1.2, which has a position vector X, and its infinitesimal neighbor dX. Point P shifts to p, which has a position vector x after the stretch. In the meantime, the small "step" dX is stretched to dx.

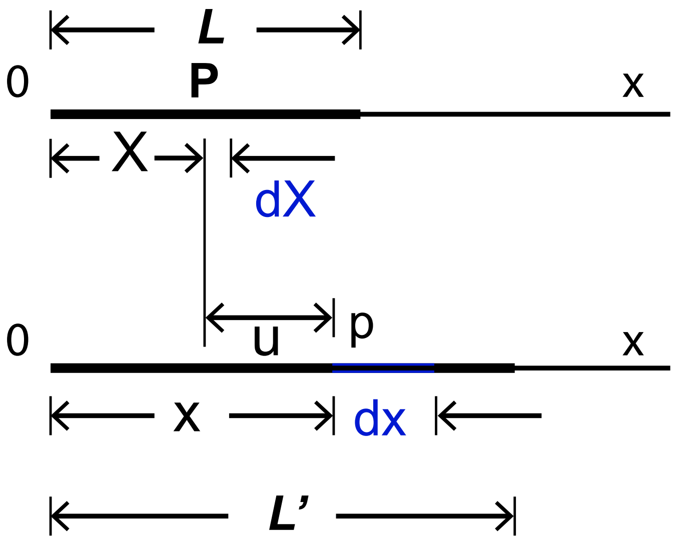


Fig. 1.3: Strain calculation. Figure inspired from [2]

The strain at point P can be defined as in the global strain measure,

$$\varepsilon = \frac{\mathrm{dx}_1 - \mathrm{dX}_1}{\mathrm{dX}_1} \tag{1.4}$$

Since $u_1 = x_1 - X_1$, the strain can hence be rewritten as,

$$\varepsilon = \frac{\mathrm{dx}_1 - \mathrm{dX}_1}{\mathrm{dX}_1} = \frac{du_1}{\mathrm{dX}_1} \tag{1.5}$$

In the case of a three-dimensional material, the non-linear strain tensor $\boldsymbol{\varepsilon}$ is defined as

$$\boldsymbol{\varepsilon} = \frac{1}{2}(\boldsymbol{\nabla\varphi}^T\boldsymbol{\nabla\varphi} - \mathbf{I}) \quad \text{where} \quad \mathbf{I} = \mathrm{diag}(1)_{3\times 3} \tag{1.6}$$

Then,

$$\begin{aligned} \boldsymbol{\nabla\varphi}^T\boldsymbol{\nabla\varphi} &= (\mathbf{I} + \boldsymbol{\nabla u})^T(\mathbf{I} + \boldsymbol{\nabla u}) \\ &= \mathbf{I} + \boldsymbol{\nabla u}^T + \boldsymbol{\nabla u} + \boldsymbol{\nabla u}^T\boldsymbol{\nabla u} \end{aligned} \tag{1.7}$$

When the deformation is small, $\boldsymbol{\nabla}u^T\boldsymbol{\nabla}u$ is negligible, then, from Equation 1.6 we get

$$\begin{aligned} \boldsymbol{\varepsilon} &= \frac{1}{2}(\boldsymbol{\nabla\varphi}^T\boldsymbol{\nabla\varphi} - \mathbf{I}) \\ &= \frac{1}{2}(\mathbf{I} + \boldsymbol{\nabla u}^T + \boldsymbol{\nabla u} - \mathbf{I}) \\ &= \frac{1}{2}(\boldsymbol{\nabla u}^T + \boldsymbol{\nabla u}) \end{aligned} \tag{1.8}$$

The tensor components are

$$\varepsilon_{i,j} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{1.9}$$

which is the *linearized Cauchy elasticity tensor*[21].

We obtain a linear dependency between tension and distortion because of the linearization.

Usually, the strain is written as an array in the so-called engineering strain, as defined by [6]

$$\{\varepsilon\} = \{\varepsilon_{xx}\ \varepsilon_{yy}\ \varepsilon_{zz}\ 2\varepsilon_{xy}\ 2\varepsilon_{yz}\ 2\varepsilon_{xz}\}^T \tag{1.10}$$

with only six unique components because of its symmetry and the shear components with twice their value.

1.3 The stress

The stress, represented by σ, can be simply thought as a force distributed over an area. For example, the force applied to a cylinder in the following way:

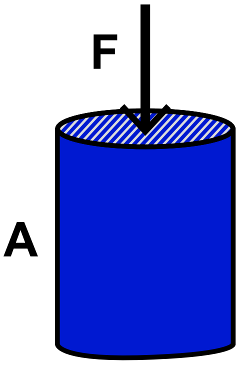


Fig. 1.4: Force being applied to a cylinder.

To calculate the stress, one would divide the force by the cross-sectional area of the cylinder because the force is constant and perpendicular to the body:

$$\sigma = F/A \tag{1.11}$$

The arrangement of the molecules of a body that is not deformed corresponds to a state of thermal equilibrium [1]. When a deformation occurs, the arrangement of the body's molecules is changed, and the body ceases to be in its original state of equilibrium. Forces therefore arise which tend to return the body to equilibrium. These internal forces which occur when a body is deformed are called *internal stresses*.

The internal stresses are due to molecular forces (i.e. the forces of interaction between the molecules). An important fact in the theory of elasticity is that the molecular forces have a very short range of action. Their effect extends only to the neighbourhood of the molecule exerting them, over a distance of the same order as that between the molecules, whereas in the theory of elasticity, which is a macroscopic theory, the only distances considered are large compared with the distances between the molecules.

The stress σ results from the deformation of the body, which is characterized by the displacements of the points in the body u.

The stress is a symmetric second order tensor with axial: those with repeated indices, and tangential or shear components; those with non repeated indices. It can be seen as if one of index indicates the normal direction of face where the tension is applied and the other index indicates the force component in the respective direction. In Figure 1.5, we can see an illustration of it.

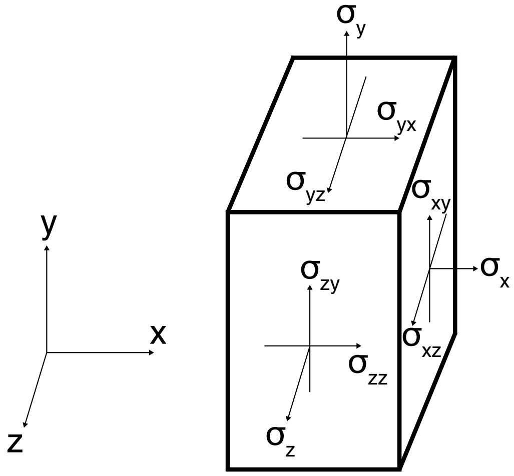


Fig. 1.5: Stress tensor components. Components with non repeated indices are symmetric. Figure extracted from [3].

In the case of a three-dimensional material, the stress is defined by [6] as

$$\{\sigma\} = \{\sigma_{xx}\ \sigma_{yy}\ \sigma_{zz}\ \sigma_{xy}\ \sigma_{yz}\ \sigma_{xz}\}^T \tag{1.12}$$

In the following section, the relationship between the strain and the stress is described.

1.4 Elastic stress-strain law

We suppose that the body is not subject to tension in the initial configuration and behaves the same in all directions and at all points, i.e., the material is isotropic[1] and homogeneous.

When applying a load a part of the body, the relationship between stress and strain is initially linear. While the relationship remains linear, it is considered the elastic region of the material.

In the 1-D context, the elastic stress-strain law, known as Hooke's law is $\sigma = E\varepsilon$, where E is the Young's modulus[2].

[1] Material where properties are independent of direction, such as metals, concrete and plastics.

[2] The Young's modulus is the change in stress over the change in strain.

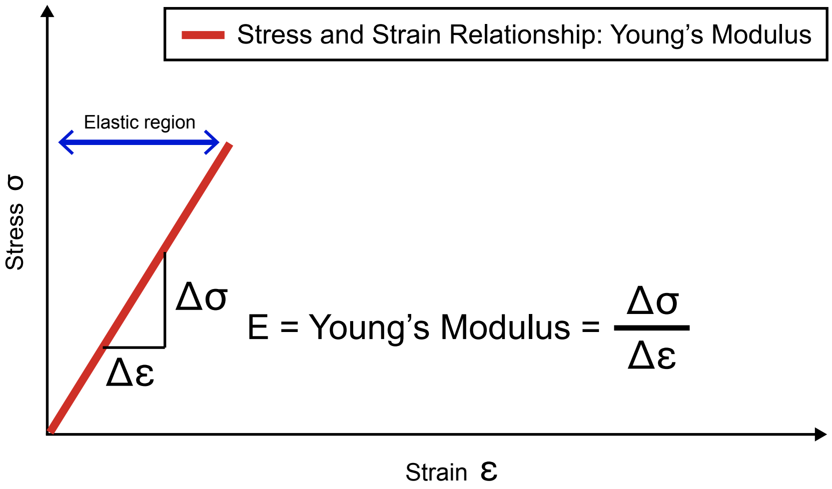


Fig. 1.6: Stress vs strain relationship in the elastic region of a material.

This linear stress-strain relationship yields E, which is the Young's modulus of the part or material.

In 3-D, the linear relationship between the stress and the strain is a tensorial form of the isotropic linear elastic stress-strain relation that connects components of both tensors:

$$\sigma_{ij} = 2\mu\varepsilon_{ij} + \delta_{ij}\lambda\sum_{k=1}^{3}\varepsilon_{kk} \tag{1.13}$$

where ν is the Poisson's ratio[3] and λ and μ are the Lamé constants[4], where

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{1.14}$$

$$\mu = \frac{E}{2(1+\nu)} \tag{1.15}$$

While the Young's modulus can take any positive values, the Poisson's ratio is inferior to 0.5 and incompressible materials have a Poisson's ratio close to 0.5 [22].

In the 2-D and 3-D cases, the constitutive equation written in matricial form becomes

$$\{\sigma\} = \boldsymbol{D}\{\varepsilon\} \tag{1.16}$$

where

$$\{\sigma\} = \{\sigma_x \; \sigma_y \; \sigma_z \; \sigma_{xy} \; \sigma_{yz} \; \sigma_{xz}\}^T$$
$$\{\varepsilon\} = \{\varepsilon_{xx} \; \varepsilon_{yy} \; \varepsilon_{zz} \; 2\varepsilon_{xy} \; 2\varepsilon_{yz} \; 2\varepsilon_{xz}\}^T$$

and the material property matrix $\boldsymbol{D}$ is defined as

[3] The Poisson's ratio quantifies the change of the volume, i.e. the expansion of an object perpendicular to a compression or stretch.

[4] The Lamé constants are two material-dependent quantities denoted by λ and μ that arise in strain-stress relationships.

$$\boldsymbol{D} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \tag{1.17}$$

1.5 Elasticity problem - differential formulation

In this context, the balance of linear momentum yields [16]

$$\nabla.\boldsymbol{\sigma} + \boldsymbol{g} = \rho\frac{\partial \mathbf{v}}{\partial t} \tag{1.18}$$

where ρ is the material density in spatial form, $\boldsymbol{g} : \Omega \longrightarrow \mathbb{R}^3$ is the body force and v the velocity dependent on time t; which is Cauchy's equation of motion (CEM) in Eulerian form.

We consider object deformations enforced by boundary conditions like displacements $\boldsymbol{u}_D$ on Γ_D – *Dirichlet boundary conditions* – and surface forces $\boldsymbol{s}$ on Γ_N (where $\Gamma_N \cap \Gamma_D = \emptyset$) – *Neumann boundary conditions.* Then we can formulate all boundary conditions

$$\boldsymbol{u}_D : \Gamma_D \longrightarrow \mathbb{R}^3 \quad \text{and}$$
$$\boldsymbol{s} : \Gamma_N \longrightarrow \mathbb{R}^3.$$

The *differential* or *strong formulation* of the *elasticity problem* can be obtained by applying the force equilibrium on infinitesimal small volumes of the object, see [3]:

Find $\boldsymbol{u} = \boldsymbol{\varphi} - \mathbf{id} \in \mathbb{C}^2(\Omega) \cap \mathbb{C}^1(\overline{\Omega})$, such that

$$\begin{aligned} \nabla.\boldsymbol{\sigma} + \boldsymbol{g} &= \rho\frac{\partial \mathbf{v}}{\partial t} & \\ \boldsymbol{\sigma}(\mathbf{x})\boldsymbol{n}(\mathbf{x}) &= \boldsymbol{s}(\mathbf{x}) & \forall \mathbf{x} \in \Gamma_N \\ \boldsymbol{u}(\mathbf{x}) &= \boldsymbol{u}_D(\mathbf{x}) & \forall \mathbf{x} \in \Gamma_D \\ \partial\Omega &= \Gamma_N \cup \Gamma_D & \Gamma_D \cap \Gamma_N = \emptyset \end{aligned}$$

where $\boldsymbol{n}$ is the normal of Ω pointing outwards and $\partial\Omega$ is its boundary.

1.6 Cutting problem - differential formulation

In this section we analyze the cutting problem by firstly defining the cut[5] surface $\Gamma^c \subset \mathbb{R}^3$.

For simplicity reasons, we only consider one partial cut, i.e. the parameterized cut boundary is continuous. However, the approach can be extended in order to accomodate multiple cut configurations.

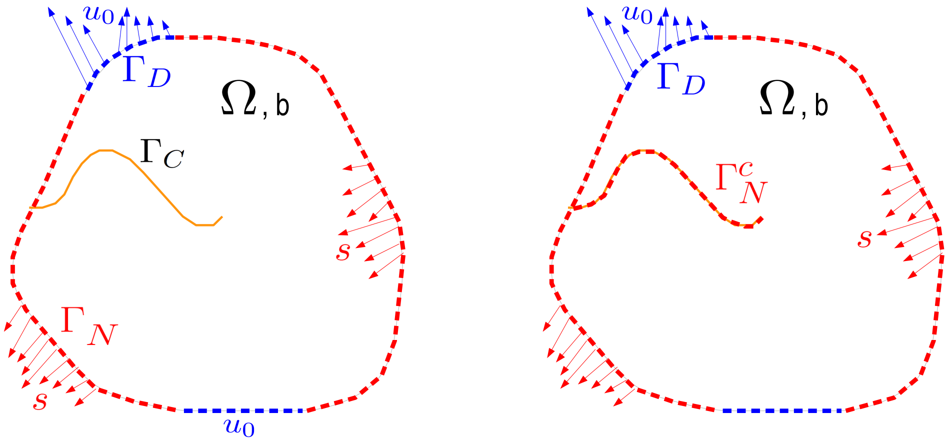
[5] cut related variables use a superscript c, for example v^c

Since the cut separates the body, the newly considered domain is $\Omega^c = \Omega \backslash \Gamma^c$ [16]. As a result of that, the boundary $\partial\Omega^c$ of the body is larger and more surface is added: $\partial\Omega^c = \partial\Omega \cup (\Gamma^c \cap \Omega)$.

Then, the cutting problem corresponds to an elasticity problem like the one mentioned above, where the Neumman boundary condition is

$$\Gamma_N^c = \Gamma_N \cup \Gamma^c \tag{1.19}$$

The introduced problem can be contemplated in the following figure:



(a) Elasticity problem with a cut.

(b) Interpretation as a cutting problem with extended boundary force condition.

Fig. 1.7: 2D example of a cutting problem with force and displacement boundary conditions. Figure extracted from [16]

1.7 Weak formulation

To develop the finite element equations, the partial differential equations must be restated in an integral form called the *weak form* [9]. A weak form of the differential equations is equivalent to the governing equation and boundary conditions, i.e. the strong form.

The weak form works as follows: By multiplying the above differential equation with an arbitrary test function $\delta\boldsymbol{u} \in \mathbb{C}_0^\infty(\Omega) \cap V_D(\Gamma_D, \mathbf{0})$ and integrating the equation over the space Ω, we obtain an integral equation that is equivalent to the differential equation. Using the *Gauß integration theorem*, this integral formulation can be transformed to the so-called *variational* or *weak formulation* of the problem

$$\boxed{\begin{gathered}\text{Find } \boldsymbol{u} = \boldsymbol{\varphi} - \mathbf{id} \in V = H^1(\Omega) \cap V_D(\Gamma_D, \boldsymbol{u}_D) \cap V_N(\Gamma_N^c, \boldsymbol{s}), \text{ such that} \\ \int_\Omega \partial\boldsymbol{u}^T . \rho \frac{\partial \mathbf{v}}{\partial t} = -\int_\Omega \nabla\delta\boldsymbol{u} : \boldsymbol{\sigma}\, d\Omega + \int_{\Gamma_N} \delta\boldsymbol{u}^T \cdot \boldsymbol{s}\, d\Gamma_N + \int_\Omega \delta\boldsymbol{u}^T \cdot \boldsymbol{g}\, d\Omega \quad (1.20) \\ \forall\delta\boldsymbol{u} \in \mathbb{C}_0^\infty(\Omega) \cap V_D(\Gamma_D, \mathbf{0})\end{gathered}}$$

where $H^1(\Omega)$ is the *Sobolev space* [21] over Ω.

We can notice that

$$\nabla\delta\boldsymbol{u} : \boldsymbol{\sigma} = \sum_{i,j} (\nabla\delta\boldsymbol{u})_{ij}\sigma_{ji} \tag{1.21}$$

The previous introduction presented all the concepts of elasticity necessary to understand the main problem of this thesis and, in the following chapter, its discretization using the finite element method is explained.

2. THE FINITE ELEMENT METHOD

In this chapter, we introduce the main notions of the finite element method (FEM), that replaces an object by a finite number of nodes connected by so-called finite elements that use the material laws explained in the previous section.

2.1 Method description

The FEM is used to approximate the solution of problems described by partial differential equations, like the elasticity deformation problem defined in chapter 1. In particular, it is a numerical approach by which these partial differential equations can be solved approximately. From an engineering standpoint, the FEM is a method for solving engineering problems such as stress analysis, heat transfer, fluid flow and electromagnetics by computer simulation. It is used to predict the behavior of structural, mechanical, thermal, electrical and chemical systems for both design and performance analysis as well.

The basic idea of the FEM is to divide the body (e.g. the plate with a hole of Figure 2.1) into *finite elements*, just called *elements* (Figure 2.2), connected by *nodes* (the element tips). For example, a plate with a hole viewed as a *source mathematical object* is replaced by triangles (*discrete approximations*) and elements could be separated (*disassembly*) by disconnecting nodes. This is called the finite element *mesh* and the process of making the mesh is called *mesh generation*.

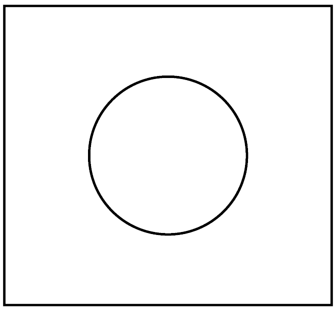


Fig. 2.1: Plate with a hole.

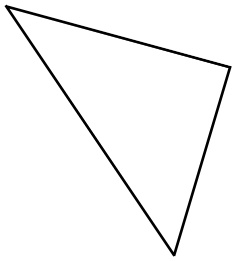


Fig. 2.2: Triangular Finite Element.

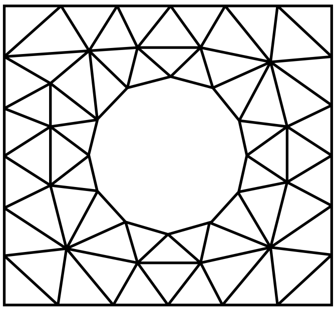


Fig. 2.3: Finite Element Model. Figure inspired from [9]

As a first step, an integral equation - the weak formulation - is derived based on the strong and variational formulation.

In order to discretize the continuous problem, the space is represented by points and elements that interpolate the positions between the points. Then, these discretizations are inserted inside of the weak formulation approximating the balance equation at a time t.

Specifically, the finite element method consists of the following five steps [9]:

1. Preprocessing: subdividing the problem domain into the finite elements.
2. Element formulation: development of equations for elements.
3. Assembly: obtaining the equations of the entire system from the equations of individual elements.
4. Solving the system of equations.
5. Postprocessing: determining quantities of interest, such as stress and strain, and obtaining visualizations of the response.

Step 1 is generally performed by automatic mesh generators. Step 2 generally requires the development of the partial differential equations for the problem and its weak form 1.20. Steps 3 and 5 are programmed and step 4 is solved automatically.

The finite element method has been introduced recently to the field of real-time soft body deformation [20] and has many benefits such as accuracy and robustness. However, it also shows some limitations, such as computation time and sensivity to the mesh resolution and mesh quality.

In what follows, some important concepts to understand the finite element discretization are defined.

2.2 Element

An element is the basic building block of the finite element analysis. It is a mathematical relation that defines how the degrees of freedom of a node relate to the next. It also relates how the deflections create stresses.

There are several types of elements: lines in 1-D, triangles or quadrilaterals in 2-D and tetrahedron, penta, hexahedron or pyramid in 3-D. Some of them are presented in Figure 2.4.

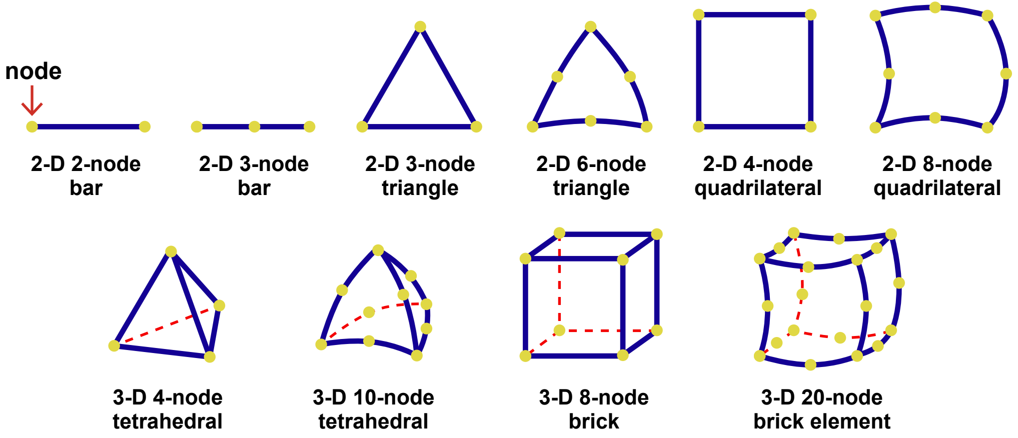


Fig. 2.4: Example of FEM element types.

The element type to be used for a finite elements analysis depends on the object to be modeled and the type of analysis to be performed. In our case, we use isoparametric triangles for 2-D and tetrahedra for 3-D, with conforming meshs.

2.3 Nodes and Degrees of Freedom

A **node** is a coordinate location in space where each of its entries is called a degree of freedom (DOF). Usually, nodes are located in the vertices of the elements. However, some elements have “midside” nodes – i.e. nodes positioned midway between the corner nodes. The edges of these “higher order” elements can therefore curve – making them suitable for capturing complex geometrical shapes, as shown in Figure 2.4.

This is possible since these elements permit the solution between the nodes to vary in non-linear ways [23], which is an important feature when field variables change rapidly.

In finite element analysis, a node may be limited in the calculated motions for a variety of reasons. For example, we don’t need to calculate the out of plane translation on a 2-D element because it would not be a 2-D element if its nodes were allowed to move out of the plane [7]. The results of finite element analysis (deflections and stresses) are usually given at the nodes.

2.4 Finite Element Discretization

The idea of the finite element discretization technique is to look for an approximated solution $\boldsymbol{u}_h$ in a suitable finite-dimensional space to the variational problem in the infinite-dimensional space V described in 1.20 of continuous functions. $\boldsymbol{u}_h$ is defined in the discrete sub-space $V_h \in V$.

The so called nodal basis functions N_J is zero at every other node except the associated J-th node, at which it is one. This is called the δ-property.

In order to discretize the weak formulation described in 1.20, $\boldsymbol{u}_h$ is expressed by the linear combination

$$\boldsymbol{u}_h = \sum_{J=1}^{n} \boldsymbol{U}_J N_J \tag{2.1}$$

of the n basis functions N_J with the time dependent coefficients $\boldsymbol{U}_J$ which represent the displacement vector on node J, $\boldsymbol{U}_J = (U_{J,1}, U_{J,2}, U_{J,3})$.

Considering $\boldsymbol{u}_h = (u_{h,1}, u_{h,2}, u_{h,3})$, equation 2.1 can be written, for a 4-node element example, in matrix form as:

$$\begin{bmatrix} u_{h,1} \\ u_{h,2} \\ u_{h,3} \end{bmatrix} = \begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 & 0 & 0 \\ 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 & 0 \\ 0 & 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 \end{bmatrix} \begin{bmatrix} U_{1,1} \\ U_{1,2} \\ U_{1,3} \\ U_{2,1} \\ U_{2,2} \\ U_{2,3} \\ U_{3,1} \\ U_{3,2} \\ U_{3,3} \\ U_{4,1} \\ U_{4,2} \\ U_{4,3} \end{bmatrix} \tag{2.2}$$

As a direct consequence of the δ-property, the coefficients $U_{i,j}$ coincide with the displacements of the element nodes when using nodal basis functions. The solution $\boldsymbol{u}_h$ is C^0-continuous and we have $V_h \in V$, allowing statements on uniqueness and existence of solutions to be directly transferred from the continuous to the discrete problem [21].

The figure below illustrates this principle for a 1-D problem. In this case, there are seven elements along the portion of the x-axis, where the function u is defined.

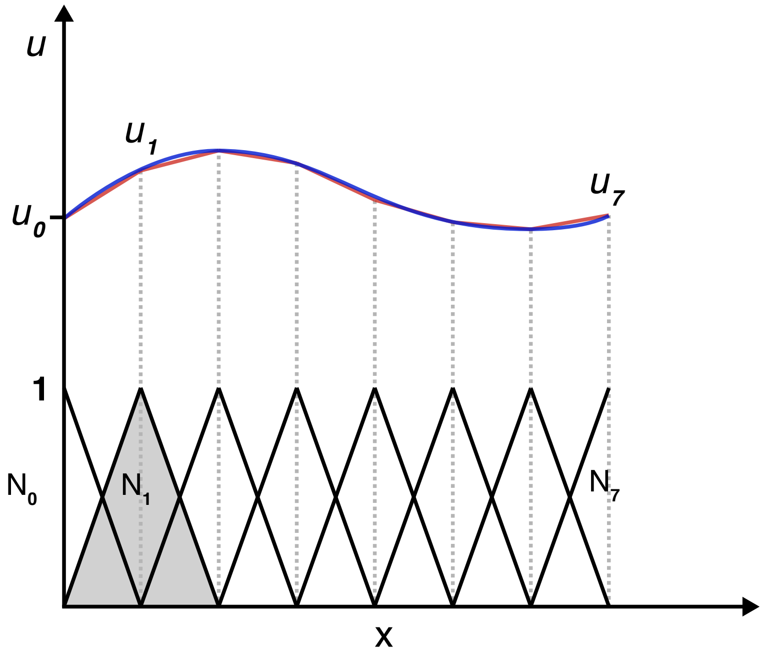


Fig. 2.5: The function u (blue line) is approximated by u_h (red line), which is a linear combination of basis functions N_i (represented by the solid black lines). The coefficients are denoted by u_0 through u_7. Figure inspired from [25]

For later use, we also introduce the displacement gradient in matrix form. For a 4-node element example it reads,

$$\nabla \boldsymbol{u}_h = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ \frac{\partial u}{\partial z} \\ \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial v}{\partial z} \\ \frac{\partial w}{\partial x} \\ \frac{\partial w}{\partial y} \\ \frac{\partial w}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} & 0 & 0 \\ \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} & 0 & 0 \\ \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z} & 0 & 0 \\ 0 & \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} & 0 \\ 0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} \\ 0 & 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} \\ 0 & 0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z} \end{bmatrix} \begin{bmatrix} U_{1,1} \\ U_{1,2} \\ U_{1,3} \\ U_{2,1} \\ U_{2,2} \\ U_{2,3} \\ U_{3,1} \\ U_{3,2} \\ U_{3,3} \\ U_{4,1} \\ U_{4,2} \\ U_{4,3} \end{bmatrix} \tag{2.3}$$

It is theoretically possible to use different function spaces for $\boldsymbol{u}_h$ and the test functions $\delta\boldsymbol{u}_h$. However, in practice we usually choose these spaces to be the same (Galerkin method). Thus we have

$$\nabla\delta\boldsymbol{u}_h = \begin{bmatrix}
\frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} & 0 & 0 \\
\frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} & 0 & 0 \\
\frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z} & 0 & 0 \\
0 & \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} & 0 \\
0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} & 0 \\
0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z} & 0 \\
0 & 0 & \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} \\
0 & 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} \\
0 & 0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z}
\end{bmatrix}
\begin{bmatrix} \delta U_{1,1} \\ \delta U_{1,2} \\ \delta U_{1,3} \\ \delta U_{2,1} \\ \delta U_{2,2} \\ \delta U_{2,3} \\ \delta U_{3,1} \\ \delta U_{3,2} \\ \delta U_{3,3} \\ \delta U_{4,1} \\ \delta U_{4,2} \\ \delta U_{4,3} \end{bmatrix} \tag{2.4}$$

Equations 2.3 and 2.4 can be compactly written as $\nabla\boldsymbol{u}_h = \boldsymbol{BU}$ and $\nabla\delta\boldsymbol{u}_h = \boldsymbol{B}\,\delta\boldsymbol{U}$, respectively, defining for a 4-node element example,

$$\boldsymbol{B} = \begin{bmatrix}
\frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} & 0 & 0 \\
\frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} & 0 & 0 \\
\frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z} & 0 & 0 \\
0 & \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} & 0 \\
0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} & 0 \\
0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z} & 0 \\
0 & 0 & \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} \\
0 & 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} \\
0 & 0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z}
\end{bmatrix} \tag{2.5}$$

We can then approximate

$$\boxed{\nabla\delta\boldsymbol{u} : \boldsymbol{\sigma}}$$

in 1.20 by

$$\boxed{\delta\boldsymbol{U}^T\boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}^*\boldsymbol{U}}$$

where

$$\boldsymbol{B}^*\boldsymbol{U} \approx \frac{\nabla\boldsymbol{u}^T + \nabla\boldsymbol{u}}{2} \text{ and } \boldsymbol{D} \text{ is defined in 1.17.} \tag{2.6}$$

Then, we obtain

$$\int_\Omega \nabla\delta\boldsymbol{u} : \boldsymbol{\sigma}\; d\Omega = \int_\Omega \delta\boldsymbol{U}^T\boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}^*\boldsymbol{U} d\Omega = \delta\boldsymbol{U}^T \underbrace{\int_\Omega \boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}^* d\Omega\, \boldsymbol{U}}_{\mathrm{K}} = \delta\boldsymbol{U}^T\boldsymbol{K}\boldsymbol{U} \tag{2.7}$$

The element stiffness matrix is of the form

$$\boldsymbol{K^e} = \int_{\boldsymbol{\Omega^e}} \boldsymbol{B^T}\boldsymbol{D}\boldsymbol{B}^* d\Omega \tag{2.8}$$

By putting matrix $\boldsymbol{D}$ from 1.17 and matrix $\boldsymbol{B}$ from 2.5 into 2.8, the element stiffness matrix is computed.

If we assemble all of the $\boldsymbol{K}^e$ for every e, we get $\boldsymbol{K}$:

$$\boldsymbol{K} = \sum_e \int_{\boldsymbol{\Omega}^e} \boldsymbol{B}^T \boldsymbol{D} \boldsymbol{B}^* d\Omega \tag{2.9}$$

On the other side, we get

$$\int_{\Gamma_N} \delta \boldsymbol{u}^T \cdot \boldsymbol{s} \; d\Gamma_N = \int_{\Gamma_N} \delta \boldsymbol{U}^T N_{\boldsymbol{s}}^T \cdot \boldsymbol{s} \; d\Gamma_N \tag{2.10}$$

where $N_{\boldsymbol{s}}$ interpolates a $\boldsymbol{u}$ in surface Γ_N. Similarly for the body force $\boldsymbol{g}$, we can write

$$\int_{\Omega} \delta \boldsymbol{u}^T \cdot \boldsymbol{g} \; d\Omega = \int_{\Omega} \delta \boldsymbol{U}^T N_{\boldsymbol{s}}^T \cdot \boldsymbol{g} \; d\Omega \tag{2.11}$$

And

$$\delta \boldsymbol{U}^T \int_{\Gamma_N} N_{\boldsymbol{s}}^T . \boldsymbol{s} \; d\Gamma_N - \delta \boldsymbol{U}^T \int_{\Omega} N_{\boldsymbol{s}}^T \cdot \boldsymbol{g} \; d\Omega = \delta \boldsymbol{U}^T \boldsymbol{F} \tag{2.12}$$

So we get the global system of equations

$$\boldsymbol{K}\boldsymbol{U} = \boldsymbol{F} \tag{2.13}$$

because the $\delta \boldsymbol{U}$ can be cancelled out.

2.5 Linear Elements

Linear elements are characterized by their linear polynomials shape functions.

In this case, we analyze the fournode tetrahedron because they are implemented in SOFA and are used to compare the quadratic tetrahedra efficiency.

On the other hand, linear elements geometry is the simplest one in three space dimensions, and no numerical integration is needed to construct element equations. Also, it is a good way to introduce the basic steps of formulation of 3-D solid elements, particularly as regards use of natural coordinate systems, node numbering conventions and computational ingredients.

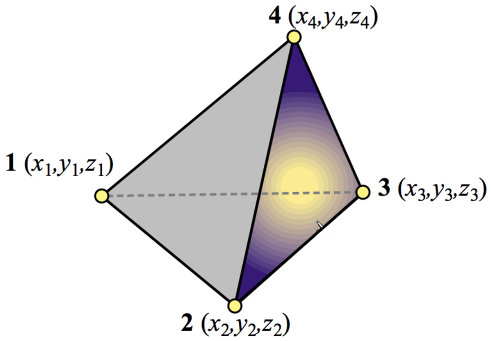


Fig. 2.6: The four node tetrahedron element.

2.5.1 Linear Tetrahedron Geometry

A tetrahedron has six edges, defined by their two end corners: 12, 23, 31, 14, 24 and 34.

The tetrahedron geometry is defined by giving the position of the four corner nodes (vertex) with respect to the global rectangular Cartesian coordinate system {x, y, z}: x_i, y_i, z_i (i = 1, 2, 3, 4).

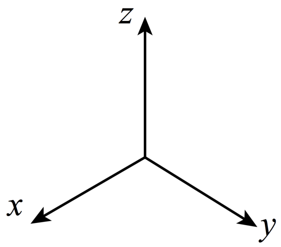


Fig. 2.7: Rectangular Cartesian coordinate system.

We often use the abbreviations for corner coordinate differences:

$$x_{ij} = x_i - x_j$$

$$y_{ij} = y_i - y_j$$

$$z_{ij} = z_i - z_j$$

with i, j = 1,.. 4.

The four corners are assumed not to be coplanar. The element has six edges and four faces. Sides are straight because they are defined by two corner points. Faces are planar because they are defined by three corner points. At each corner three sides and three faces meet.

The domain occupied by the tetrahedron is denoted by Ω^e.

The volume of the tetrahedron is denoted by V, which should not be confused with the domain identifier Ω or Ω^e. The volume is given by the following determinant in terms of the corner coordinate values:

$$V = \int_{\Omega^e} d\Omega^e = \tfrac{1}{6} det \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} = \tfrac{1}{6} det(\mathbf{J}) = \tfrac{1}{6} J \tag{2.14}$$

The above displayed matrix (without the $\frac{1}{6}$ factor) is called the Jacobian matrix $\mathbf{J}$ and its determinant the Jacobian determinant J .

We deduce from 2.14 that

$$det(\mathbf{J}) = J = 6V \tag{2.15}$$

We assume that V is positive. This can be insured if the nodes are not coplanar, and are appropriately numbered. A numbering rule that achieves this goal is as follows [17]:

- Pick a corner (any corner) as initial one. In the figure above, it's the number 1.
- Pick a face to contain the first three corners. The opposite corner will be numbered 4.
- Number those three corners in a counterclockwise sense when looking at the face from the opposite one.

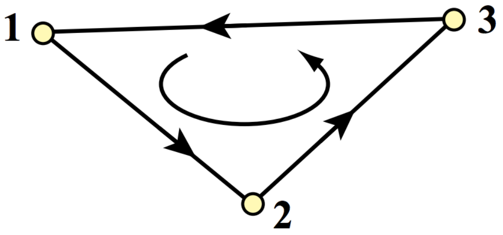


Fig. 2.8: Corner node numbering convention. Figure extracted from [17]

An explicit expression for the tetrahedron volume in terms of corner locations is

$$J = x_{21}(y_{23}z_{34} - y_{34}z_{23}) + x_{32}(y_{34}z_{12} - y_{12}z_{34}) + x_{43}(y_{12}z_{23} - y_{23}z_{12}) \tag{2.16}$$

in which the abbreviations for coordinate differences are used.

2.5.2 Tetrahedron coordinates

The position of a tetrahedron point may be specified either by its Cartesian coordinates $\{x, y, z\}$, or by its *tetrahedral natural coordinates.* The latter form a set of four dimensionless numbers denoted by

$$\zeta_1,\ \zeta_2,\ \zeta_3,\ \zeta_4$$

The value of ζ_i is 1 at corner i and 0 at the other 3 corners, including the entire opposite face [17]. It varies linearly with distance as one traverses the distance from the corner to that face.

Because four coordinates is one too many for 3-D space, there is a constraint between the ζ_i:

$$\zeta_1 + \zeta_2 + \zeta_3 + \zeta_4 = 1 \tag{2.17}$$

The equation $\zeta_i = \textit{constant}$ represents planes parallel to the face opposite to the $i_t h$ corner.

2.5.3 Linear Interpolation

Any function linear in x, y, z, say $F(x, y, z)$, that takes the values F_i (i = 1, 2, 3, 4) at the corners of the linear tetrahedron may be interpolated in terms of the tetrahedron natural coordinates as

$$F(\zeta_1, \zeta_2, \zeta_3, \zeta_4) = F_1\zeta_1 + F_2\zeta_2 + F_3\zeta_3 + F_4\zeta_4 = F_i\zeta_i. \tag{2.18}$$

2.5.4 Coordinate Transformations and Shape Functions

Quantities that are intrinsically linked to the element geometry (e.g., shape functions) are best expressed in tetrahedral coordinates. On the other hand, quantities such as displacement, strain and stress components are expressed in the Cartesian system $\{x, y, z\}$. Ergo, we need transformation equations to pass from one coordinate system to the other.

The element geometric description in terms of tetrahedral coordinates follows by applying the linear interpolation 2.18 to x, y and z, so that $x = x_i\zeta_i$, $y = y_i\zeta_i$, and $z = z_i\zeta_i$. Prepending the sum-of-tetrahedral-coordinates identity (2.17) as first row we build the matrix relation

$$\begin{bmatrix} 1 \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \\ \zeta_4 \end{bmatrix} \tag{2.19}$$

where the 4x4 matrix is the Jacobian matrix of the linear tetrahedron. Its inversion results in

$$\begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \\ \zeta_4 \end{bmatrix} = \begin{bmatrix} 6V_{01} & a_1 & b_1 & c_1 \\ 6V_{02} & a_2 & b_2 & c_2 \\ 6V_{03} & a_3 & b_3 & c_3 \\ 6V_{04} & a_4 & b_4 & c_4 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \\ z \end{bmatrix} \tag{2.20}$$

where

$$a_i = 6V\frac{\partial \zeta_i}{\partial x} = J\frac{\partial \zeta_i}{\partial x} \quad b_i = 6V\frac{\partial \zeta_i}{\partial y} = J\frac{\partial \zeta_i}{\partial y} \quad c_i = 6V\frac{\partial \zeta_i}{\partial z} = J\frac{\partial \zeta_i}{\partial z} \tag{2.21}$$

The Cartesian partial derivatives of the linear tetrahedron shape functions are

$$\frac{\partial F}{\partial x} = \{a_1, a_2, a_3, a_4\}; \quad \frac{\partial F}{\partial y} = \{b_1, b_2, b_3, b_4\}; \quad \frac{\partial F}{\partial z} = \{c_1, c_2, c_3, c_4\} \tag{2.22}$$

The linear tetrahedron is often shunned for stress analysis because of its poor performance. That is because derivatives of shape functions are constant over the element volume. Strains and stresses recovered from displacement derivatives can be highly inaccurate, even exhibiting wrong signs. This deficiency makes the element unreliable for stress analysis when strains and stresses exhibit significant gradients [17]. That is why we are going to introduce the quadratic elements in the next chapter.

2.6 Quadratic Elements

Quadratic Elements refer to elements with quadratic shape functions, which means that an edge between 2 points follows a quadratic deformation (ax^2+bx+c). In this section, the quadratic triangle (2-D) and quadratic tetrahedron (3-D) formulations are exposed.

2.6.1 Quadratic Triangle

The quadratic triangle has three corner nodes with local numbers 1 through 3 and three side nodes with local numbers 4 through 6, as shown in Figure 2.9.

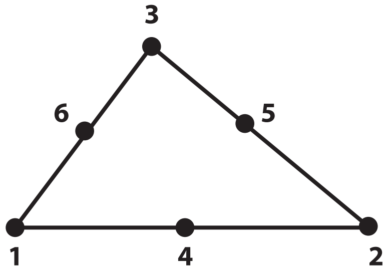


Fig. 2.9: The quadratic triangle.

For later use, we define the quadratic triangle shape functions as:

$$N^T = \begin{bmatrix} N_1^e \\ N_2^e \\ N_3^e \\ N_4^e \\ N_5^e \\ N_6^e \end{bmatrix} = \begin{bmatrix} \xi_1(2\xi_1 - 1) \\ \xi_2(2\xi_2 - 1) \\ \xi_3(2\xi_3 - 1) \\ 4\xi_1\xi_2 \\ 4\xi_2\xi_3 \\ 4\xi_3\xi_1 \end{bmatrix}$$

And their natural derivatives:

$$\frac{\partial N^T}{\partial \xi_1} = \begin{bmatrix} 4\xi_1 - 1 \\ 0 \\ 0 \\ 4\xi_2 \\ 0 \\ 4\xi_3 \end{bmatrix}, \frac{\partial N^T}{\partial \xi_2} = \begin{bmatrix} 0 \\ 4\xi_2 - 1 \\ 0 \\ 4\xi_1 \\ 4\xi_3 \\ 0 \end{bmatrix}, \frac{\partial N^T}{\partial \xi_3} = \begin{bmatrix} 0 \\ 0 \\ 4\xi_3 - 1 \\ 0 \\ 4\xi_2 \\ 4\xi_1 \end{bmatrix}$$

2.6.2 Quadratic Tetrahedron

The quadratic tetrahedron has four corner nodes with local numbers 1 through 4, which must be traversed following the same convention explained in Figure 2.8. It has six side nodes, with local numbers 5 through 10. Nodes 5, 6, 7 are located on sides 12, 23 and 31, respectively, while nodes 8, 9, 10 are located on sides 14, 24, and 34, respectively. The side nodes are not necessarily placed at the midpoints of the sides but may deviate from those locations, subjected to positive-Jacobian-determinant constraints. Each element face is defined by six nodes. These do not necessarily lie on a plane, but they should not deviate too much from it. This freedom allows the element to have curved sides and faces.

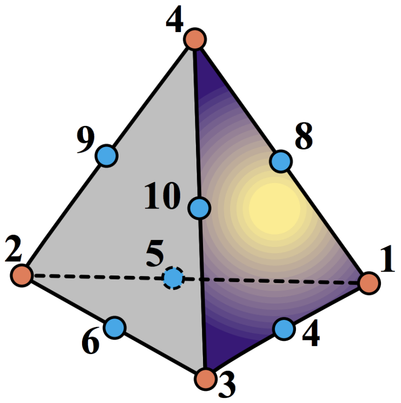


Fig. 2.10: The quadratic tetrahedron.

The definition of the quadratic tetrahedron as an isoparametric element is:

$$\begin{bmatrix} 1 \\ x \\ y \\ z \\ u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & \dots & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 & \dots & x_{10} \\ y_1 & y_2 & y_3 & y_4 & y_5 & \dots & y_{10} \\ z_1 & z_2 & z_3 & z_4 & z_5 & \dots & z_{10} \\ u_{x1} & u_{x2} & u_{x3} & u_{x4} & u_{x5} & \dots & u_{x10} \\ u_{y1} & u_{y2} & u_{y3} & u_{y4} & u_{y5} & \dots & u_{y10} \\ u_{z1} & u_{z2} & u_{z3} & u_{z4} & u_{z5} & \dots & u_{z10} \end{bmatrix} \begin{bmatrix} N_1^e \\ N_2^e \\ N_3^e \\ N_4^e \\ N_5^e \\ N_6^e \\ N_7^e \\ N_8^e \\ N_9^e \\ N_{10}^e \end{bmatrix}$$

where u_x, u_y and u_z are the three components of the element displacement field.

The conventional shape functions are given by:

$$N_1^e = \xi_1(2\xi_1 - 1) \quad N_2^e = \xi_2(2\xi_2 - 1) \quad N_3^e = \xi_3(2\xi_3 - 1) \quad N_4^e = \xi_4(2\xi_4 - 1)$$

$$N_5^e = 4\xi_1\xi_2 \quad N_6^e = 4\xi_2\xi_3 \quad N_7^e = 4\xi_3\xi_1 \quad N_8^e = 4\xi_1\xi_4 \quad N_9^e = 4\xi_2\xi_4 \quad N_{10}^e = 4\xi_3\xi_4$$

And the table of shape function derivatives for $n = 10$ is:

$$\Delta\mathbf{N} = \begin{bmatrix} 4\xi_1 - 1 & 0 & 0 & 0 & 4\xi_2 & 0 & 4\xi_3 & 4\xi_4 & 0 & 0 \\ 0 & 4\xi_2 - 1 & 0 & 0 & 4\xi_1 & 4\xi_3 & 0 & 0 & 4\xi_4 & 0 \\ 0 & 0 & 4\xi_3 - 1 & 0 & 0 & 4\xi_2 & 4\xi_1 & 0 & 0 & 4\xi_4 \\ 0 & 0 & 0 & 4\xi_4 - 1 & 0 & 0 & 0 & 4\xi_1 & 4\xi_2 & 4\xi_3 \end{bmatrix}$$

where the {i, j} entry is $\frac{\partial N_k}{\partial \xi_i}$. Taking dot products with the node coordinates $\{x_k, y_k, z_k\}$ yields

$$\mathbf{J} = 4\begin{bmatrix} \frac{1}{4} & x_1\bar{\zeta}_1 + x_5\zeta_2 + x_7\zeta_3 + x_8\zeta_4 & y_1\bar{\zeta}_1 + y_5\zeta_2 + y_7\zeta_3 + y_8\zeta_4 & z_1\bar{\zeta}_1 + z_5\zeta_2 + z_7\zeta_3 + z_8\zeta_4 \\ \frac{1}{4} & x_5\zeta_1 + x_2\bar{\zeta}_2 + x_6\zeta_3 + x_9\zeta_4 & y_5\zeta_1 + y_2\bar{\zeta}_2 + y_6\zeta_3 + y_9\zeta_4 & z_5\zeta_1 + z_2\bar{\zeta}_2 + z_6\zeta_3 + z_9\zeta_4 \\ \frac{1}{4} & x_7\zeta_1 + x_6\zeta_2 + x_3\bar{\zeta}_3 + x_{10}\zeta_4 & y_7\zeta_1 + y_6\zeta_2 + y_3\bar{\zeta}_3 + y_{10}\zeta_4 & z_7\zeta_1 + z_6\zeta_2 + z_3\bar{\zeta}_3 + z_{10}\zeta_4 \\ \frac{1}{4} & x_8\zeta_1 + x_9\zeta_2 + x_{10}\zeta_3 + x_4\bar{\zeta}_4 & y_8\zeta_1 + y_9\zeta_2 + y_{10}\zeta_3 + y_4\bar{\zeta}_4 & z_8\zeta_1 + z_9\zeta_2 + z_{10}\zeta_3 + z_4\bar{\zeta}_4 \end{bmatrix}^T$$

in which $\bar{\zeta}_i = \zeta_i - \frac{1}{4}$. Matrix $\mathbf{J}$ is displayed in transposed form to fit within page width.

The Jacobian determinant $J = 6V$ over the element domain is constant if and only if the six midside nodes are collocated at the midpoints between adjacent corners.

The shape function Cartesian derivatives are explicitly given by

$$\frac{\partial N_n}{\partial x} = (4\zeta_n - 1)\frac{a_n}{J}, \quad \frac{\partial N_n}{\partial y} = (4\zeta_n - 1)\frac{b_n}{J}, \quad \frac{\partial N_n}{\partial z} = (4\zeta_n - 1)\frac{c_n}{J},$$
$$\frac{\partial N_m}{\partial x} = 4\frac{a_i\zeta_j + a_j\zeta_i}{J}, \quad \frac{\partial N_m}{\partial y} = 4\frac{b_i\zeta_j + b_j\zeta_i}{J}, \quad \frac{\partial N_m}{\partial z} = 4\frac{c_i\zeta_j + c_j\zeta_i}{J}$$

For the corner node shape functions, $n = 1, 2, 3, 4$. For the side node shape functions, $m = 5, 6, 7, 8, 9, 10$; the corresponding right hand side indices being $i = 1, 2, 3, 1, 2, 3$ and $j = 2, 3, 1, 4, 4, 4$.

In the following chapter, we present SOFA, the framework where the finite element method for quadratic elements is implemented.

3. SOFA - AN OPEN SOURCE FRAMEWORK FOR MEDICAL SIMULATION

Our implementation of cutting quadratic tetrahedral elements is based on the SOFA (Simulation Open Framework Architecture[1]) Framework, which is mainly applied for medical simulation.

SOFA facilitates collaborations between specialists from various domains, by decomposing complex simulators into components designed independently and organized in a scenegraph data structure [12]. Each component encapsulates one of the aspects of a simulation, such as the degrees of freedom, the forces and constraints, the differential equations, the main loop algorithms, the linear solvers, the collision detection algorithms or the interaction devices. The simulated objects can be represented using several models, each of them optimized for a different task such as the computation of internal forces, collision detection, haptics or visual display. These models are synchronized during the simulation using a mapping mechanism. CPU and GPU implementations can be transparently combined to exploit the computational power of modern hardware architectures. Thanks to this flexible yet efficient architecture, SOFA can be used as a test-bed to compare models and algorithms, or as a basis for the development of complex, high-performance simulators.

Based on an advanced software architecture, SOFA [8] allows to

1. Create complex and evolving simulations by combining new algorithms with algorithms already included in SOFA;
2. Modify most parameters of the simulation – deformable behavior, surface representation, solver, constraints, collision algorithm, etc. – by editing the XML input file;
3. Build complex models from simpler ones using a scene-graph description;
4. Efficiently simulate the dynamics of interacting objects using abstract equation solvers;
5. Reuse and easily compare a variety of available methods.

SOFA allows independently developed algorithms to interact together within a common simulation while minimizing the development time required for integration.

The SOFA architecture relies on several innovative concepts, in particular the notion of multi-model representation. Most simulation components – deformable models, collision models, instruments, etc – can have several representations, connected together through a mechanism called mapping. Each representation can then be optimized for a particular task – e.g. collision detection, visualization – while at the same time improving interoperability by creating a clear separation between the functional aspects of the simulation

[1] https://www.sofa-framework.org/api/SOFA/index.html

components. As a consequence, it is possible to have models of very different nature interact together, for instance rigid bodies, deformable objects, and fluids. At a finer level of granularity, it also proposes a decomposition of physical models – i.e. any model that behaves according to the laws of physics – into a set of basic components. This decomposition leads for instance to a representation of mechanical models as a set of degrees of freedom and force fields acting on these degrees of freedom.

Another key aspect of SOFA is the use of a scene-graph to organize and process the elements of a simulation while clearly separating the computation tasks from their possibly parallel scheduling.

The main objectives of the SOFA framework are:

- Provide a common software framework for the medical simulation community
- Enable component sharing / exchange and reduce development time
- Promote collaboration among research groups
- Enable validation and comparison of new algorithms
- Help standardize the description of anatomical and biomechanical datasets

One of the most challenging aspect of medical simulation is the computation, in real-time, of accurate biomechanical models of soft-tissues. Such models being computationally expensive, many strategies have been used to improve computation times or to reduce the complexity of the original model: linear elastic models have often been used instead of more complex non-linear representations, mass-spring methods as an alternative to finite element methods, etc. Each of these simplifications induces drawbacks, yet the importance of these drawbacks depends largely on the context in which they are applied. It becomes then very difficult to choose which particular method is most likely to provide the best results for a given simulation.

To address this issue in SOFA, a fine level of granularity has been introduced for the Behavior Model. This permits for instance to switch from one solver to another in order to see the change in performance or robustness of the simulation, or to test different constitutive models in a matter of seconds, without having to recompile any of the code. To achieve this level of flexibility, a serie of generic primitives, or components, that are common to most physics-based simulations: the DoF, the mass, the force field, and the solver have been defined.

3.1 Basic structure

The DoF component *MechanicalObject* describes the degrees of freedom, and their derivatives, of the object. This includes positions, velocities, accelerations, as well as other auxiliary vectors. The Mass component represents the mass of the object. Depending on the model, the mass can be represented by a single value – all the DoFs have the same mass, a vector – the DoFs have a different mass, or even a matrix as used in complex finite element models. The Force Field describes both internal forces associated with the constitutive equations of the model, and external forces that can be applied to this object. A variety of forces are currently derived from the abstract force field representation, including springs, linear and corotational FEM and Mass-Tensor. The solver component handles the time step integration, i.e. advancing the state of the system from time t to time $t + \Delta$t. To this end, the solver sends requests to the other components to execute operations such as summation of forces, computation of accelerations, and vector operations on the DoFs such as $x = x + v\Delta$t. Currently SOFA integrates explicit Euler and Runge-Kutta 4 solvers, as well as implicit conjugate-gradient based Euler solver.

A simulation in SOFA is described as a scene with an intrinsic generalized hierarchy. This scene is composed of nodes organized as a tree or as a Directed Acyclic Graph (DAG). The different simulated objects are described in separate nodes, and different representations of a same object can be done in different sub-nodes.

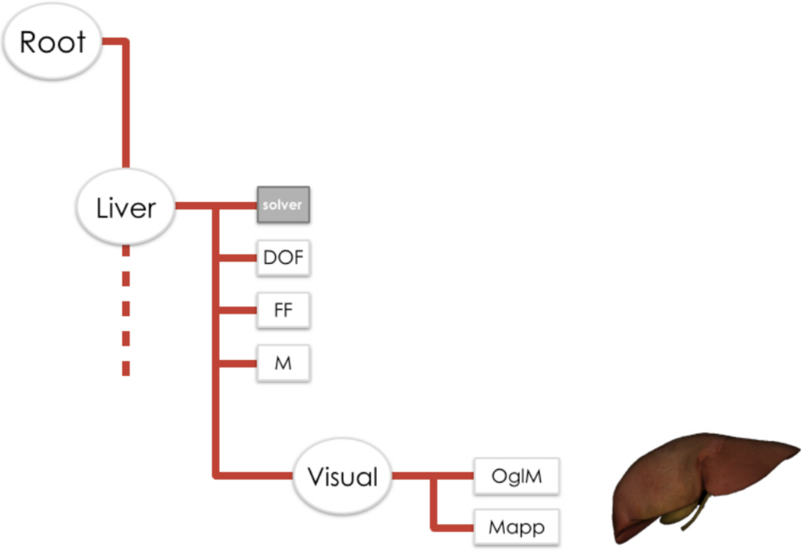


Fig. 3.1: A graph with one object and its two representations (mechanics and visual). Figure extracted from [24]

The scene starts from a parent node, called the "Root" node. All other nodes (called child nodes) inherit from this main node. In the Figure 3.1, a first child node "Liver" is defined and represents a first object. It implements the mechanical behavior of the liver (hexahedral mesh), whereas the sub-node "Visual" describes a surface model (triangular mesh) of the liver. Usually, one node gathers the components associated with the same object (same degrees of freedom).

This "Liver" node includes components (solvers, forcefield, mass) used to build the mechanical simulation of the liver. Each of these components contains attributes. For instance, a component of mass features an attribute for mass density; an iterative linear solver needs an attribute defining a maximum of iterations. These attributes are also called Data. These Data are containers providing a reflective API used for serialization in XML files and automatic creation of input/output widgets in the user interface.

It is possible to write your own SOFA components in C++, and to integrate them with SOFA by creating a plugin. A SOFA plugin is mainly a collection of SOFA components, that can be used in a scene. A plugin is actually a dynamic library that respects some conventions, so that SOFA-based applications can load it at runtime, and retrieve the components it provides.

In this project, we worked on 3 SOFA plugins: Quadratic Tetrahedra, Cutting Plugin and CGoGN. In the next section, we explain the CGoGN Plugin, which was adapted to incorporate the cut of objects represented with quadratic tetrahedra and their respective remeshing in SOFA.

3.2 Combinatorial Maps: CGoGN

Many data structures are available for the representation and manipulation of meshes, but CGoGN[2] uses the combinatorial map topological structures to be efficient in algorithms that need to traverse local neihborhoods, such as surface modeling, mesh generation, finite element analysis, geometry processing, visualization or computational geometry [15].

Combinatorial maps is a mathematical model for the representation of the topology of the subdivision of objects consistently defined in any dimension, as Figure 3.5 shows. It is valuable for topological changes in real-time.

CGoGN is a SOFA Plugin that provides the combinatorial map topological models for neighborhood relations between cells (vertices, edges, faces, volumes), which is mandatory information for many algorithms, especially in medical simulation.

In order to understand the topological structure, we need to describe what a cellular decomposition is. For this, we present an example:

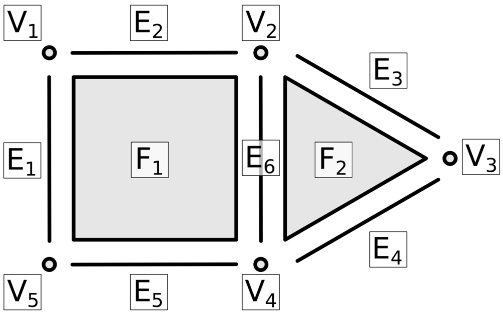


Fig. 3.2: Cellular decomposition of a 2-D object. Figure extracted from [15].

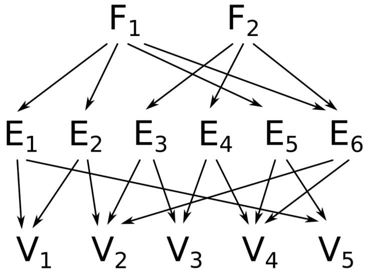


Fig. 3.3: Incidence graph. Figure extracted from [15].

Figure 3.2 shows a cellular decomposition of a 2-D object and 3.3 its incidence graph.

V_i ($i \in \{1, 2, 3, 4, 5\}$) represents the vertices, E_j ($j \in \{1, 2, 3, 4, 5\}$) the edges and F_k ($k \in \{1, 2\}$) the faces of the cellular decomposition.

[2] cgogn.unistra.fr

3.2.1 From Incidence Graph to Cell-Tuples

In a n-dimensional cellular decomposition, a cell-tuple is defined as an ordered sequence of cells $(C_n, C_{n-1}, ..., C_1, C_0)$ of decreasing dimensions such that $\forall i, 0 < i \leq n$, C_i is incident to C_{i-1}. In other words, a cell-tuple corresponds to a path in the incidence graph from a n-cell to a vertex. Figure 3.4 shows the iterative construction of all the cell-tuples generated by the cellular decomposition of Figure 3.2.

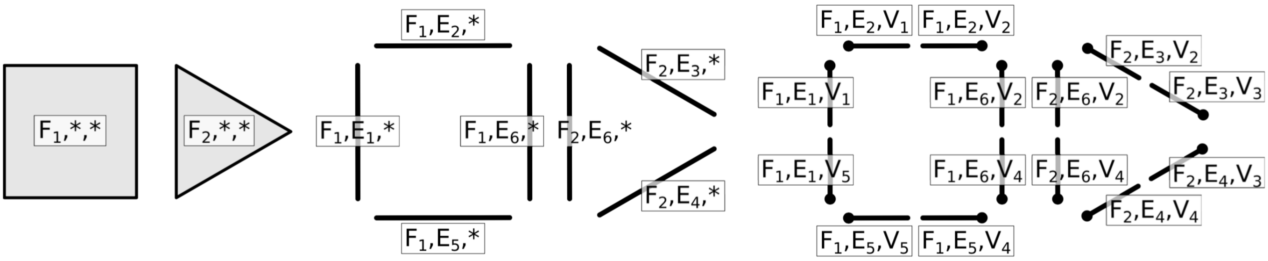


Fig. 3.4: Iterative construction of the cell-tuples corresponding to the cellular decomposition of 3.2. Figure extracted from [15]

Adjacency relations are defined on the cell-tuples: two cell-tuples are said to be i-adjacent if their path in the incidence graph share all but the i-dimensional cell. For example, (F_1, E_2, V_1) and (F_1, E_2, V_2) are 0-adjacent.

These maps encode a cellular decomposition with a set D of abstract elements called *darts* that are one-to-one correspondance with the cell-tuples and follow the rules:

1. each dart identifies a set of n cells of different dimension, i.e. those contained in the corresponding cell-tuple.
2. each k-cell is represented by a set of darts, i.e. all the darts whose corresponding cell-tuple contains this cell.

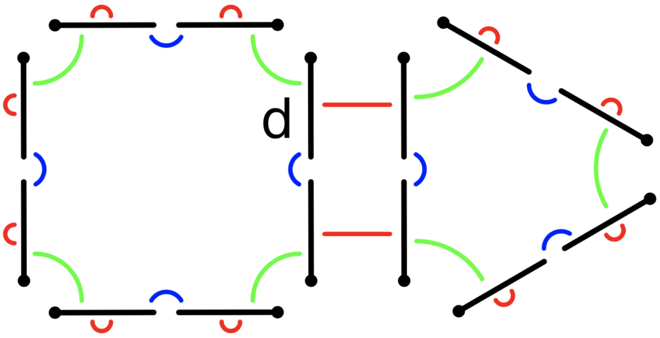


Fig. 3.5: Combinatorial Map corresponding to the cellular decomposition of Figure 3.2. Figure extracted from [15]

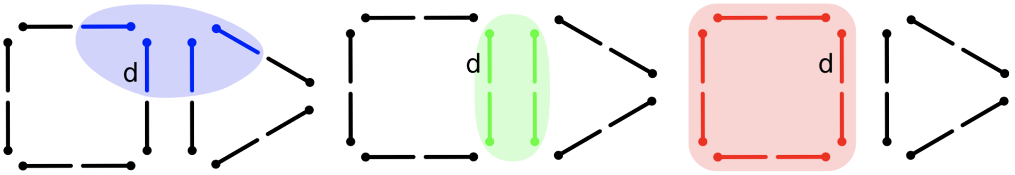


Fig. 3.6: Set of darts representing the vertex, edge and face of dart *d*. Figure extracted from [15]

In the figure above, we can see that darts are represented like the cell-tuples in Figure 3.4.

The CGoGN library provides a convenient way to traverse all the elements of the represented maps. The simplest global traversal consists in traversing all the darts of the map, but any container can be traversed in the same way. Also, the library provides a set of objects to handle adjacent cells and incident cells.

It is important to notice that a dart is not the half of an edge. A dart is to be considered as a cell-tuple. This implies that the model is able to represent consistently objects of any dimension and that each dart represents at the same time a vertex, an edge, a face and a volume of the mesh.

In this thesis, we incorporated the quadratic tetrahedra to the CGoGN topology by following a series of steps described in the following chapter.

4. IMPLEMENTATION

One of the objectives of this thesis is the organs modeling, as shown in Figures 4.1 and 4.2.

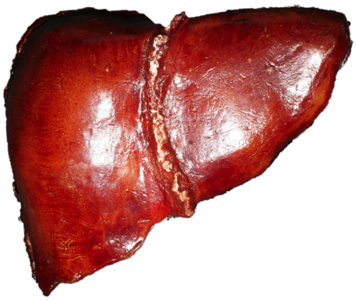


Fig. 4.1: Organ

→

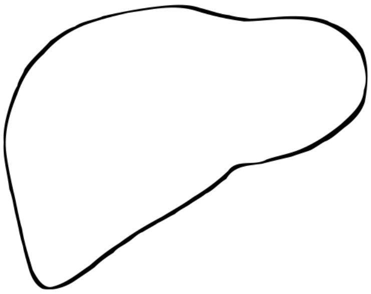


Fig. 4.2: Organ representation.

In order to simulate curved geometries, any element shown in Figure 2.4 can be used.

In the case of linear tetrahedra, many elements need to be used to represent smooth surfaces (Figure 4.3). In this chapter, we describe the methods applied to implement quadratic tetrahedra in 2 and 3 dimensions in order to achieve an improved simulation of curved surfaces with less tetrahedra (Figure 4.4).

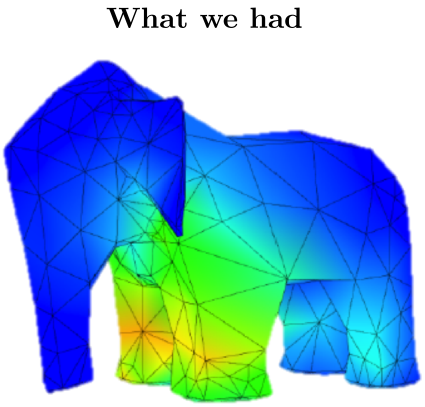


Fig. 4.3: 1200 linear tetrahedra.

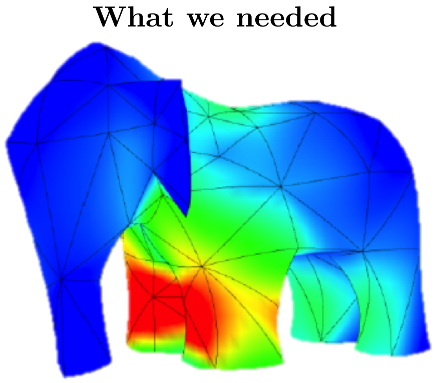


Fig. 4.4: 234 quadratic tetrahedra.

In the 2-D case, the algorithm was implemented in Python. In the 3-D case, a Stefan Suwelack plugin [11] was adapted in order to make it functional in SOFA and, then, the quadratic tetrahedra were incorporated into the CGoGN Plugin described in 3.2.

In the last case, we already had a linear tetrahedra implementation and we wanted to include the quadratic version in order to be able to represent quadratic objects with the CGoGN combinatorial map. Also, we wanted to be able to cut objects represented by quadratic tetrahedra by using the *Cutting Plugin*, which works with the CGoGN topology.

On the other hand, in order to be able to cut objects represented by quadratic tetrahedra by using the *Cutting Plugin*, we had to adapt the remeshing functions to incorporate the element involved.

Besides including quadratic tetrahedra to SOFA and the possibility of cutting objects represented by them, one of the purposes of the study was to confirm that quadratic elements have:

- Better shape approximation
- Less stiff elements
- Less degrees of freedom for the same or a better accuracy
- Better convergence
- Less quantity of elements that implies smaller amount of operations

than linear elements, as Suwelack et al [14] proposes.

In the following sections, we present the idea behind the implementations mentioned.

4.1 2-D Implementation - Python

For a better understanding, the project started with the implementation of quadratic triangles in Python.

The idea was to get the result of a beam deformation, caused by an element displacement.

The steps used were:

1. Defining the coordinates of the beam without deformation depending on the input.
2. Calculating the shape functions (2.6.1).
3. Implementing Gauss integration points and weights: In this case, the third Gauss quadrature rule (Figure 4.5) for curved sided 6-node triangles was used.

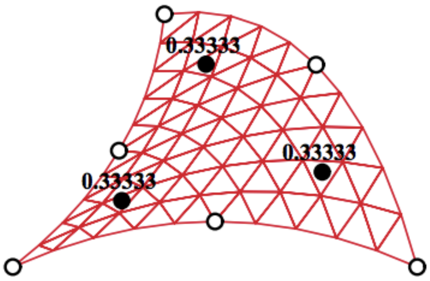


Fig. 4.5: Location and weights of sample points (dark circles) of rule 3 Gauss quadrature.

Then, the integration points iP and the integration weights iW used were

$$\text{iP} = \begin{bmatrix} 2/3 & 1/6 & 1/6 \\ 1/6 & 2/3 & 1/6 \\ 1/6 & 1/6 & 2/3 \end{bmatrix}$$

$$\text{iW} = \begin{bmatrix} 1/3 & 1/3 & 1/3 \end{bmatrix}$$

4. Calculating the global stiffness matrix[1] from every local stiffness matrix.

   To do so, we defined the Young's modulus $E = 288$ and the Poisson's ratio $\nu = 1.0/3$ to get the Lamé parameters defined in 1.14.

   Then, we calculated each local stiffness matrix using the following relationship:

$$\boldsymbol{K}^e = \int_{\Omega^e} h\boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}d\Omega \approx \sum_{i=1}^{p} w_i F(\xi_{1i}, \xi_{2i}, \xi_{3i}), \quad \text{where} \quad F(\xi_{1i}, \xi_{2i}, \xi_{3i}) = h\boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}\frac{1}{2}J. \tag{4.1}$$

   where p denotes the number of sample points of the Gauss rule being used, w_i is the integration weight for the i^{th} sample point and ξ_{1i}, ξ_{2i}, ξ_{3i} are the sample point triangular coordinates.

   Every local stiffness matrix was then added to a global stiffness matrix K in the following way:

```
for i in range(nPts) :
    for m in range(nPts) :
        for j in range(dim) :
            for n in range(dim) :
                ij = i*dim + j
                mn = m*dim + n
```

[1] System of linear equations that needs to be solved to get an approximation to a differential equation.

```
IJ = elConnect[i]*dim + j
MN = elConnect[m]*dim + n
K[IJ,MN] += Ke[ij,mn]
```

where `elConnect` is a vector containing the ids of an element (i.e [0,1,2,5,6,10] in Figure 4.6 would be an `elConnect`), `nPts` is the length of the element connections and `dim` is the number of nodes (or coordinates) of an element.

5. Getting the coordinates of each point.

   To do so, we calculated the proportion of the minimum and the maximum of each axis, and the number of elements to be represented.

6. Constructing every triangle with its correct connections.

   For that, we had to get the connecting points for each triangle from the `elConnect` previously mentioned, which relationship can be seen in the following figure.

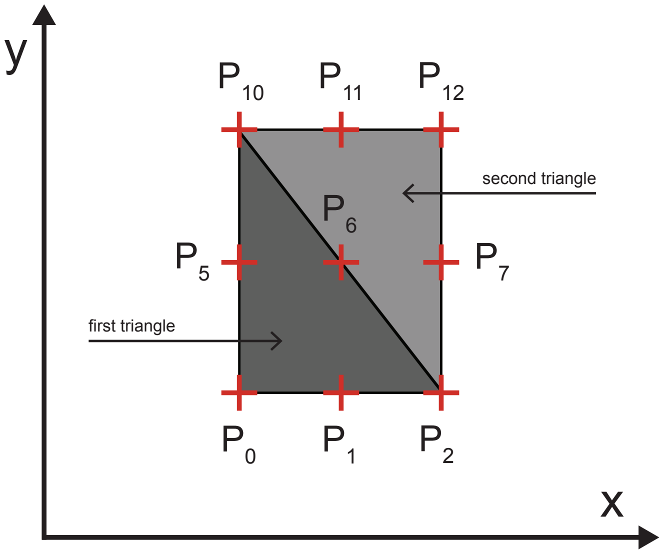


Fig. 4.6: Example of connection of points in a beam of 4 triangles.

7. Plotting the resulting mesh.

The Python program input is an XML file with the following parameters:

- Young's modulus
- Poisson's ratio
- Displacement boundary conditions
- A restricting plane to specify the regions of interest of the object (defined by a point and the normal)

In Figure 4.7, we can see the result of an example of the described implementation with the following input:

DOFs	***ν***	***E***	**Restricting points**	**Displaced point**
480	0.45	30000	[(0,0),(0.5,0),(1,0)]	(0,10)

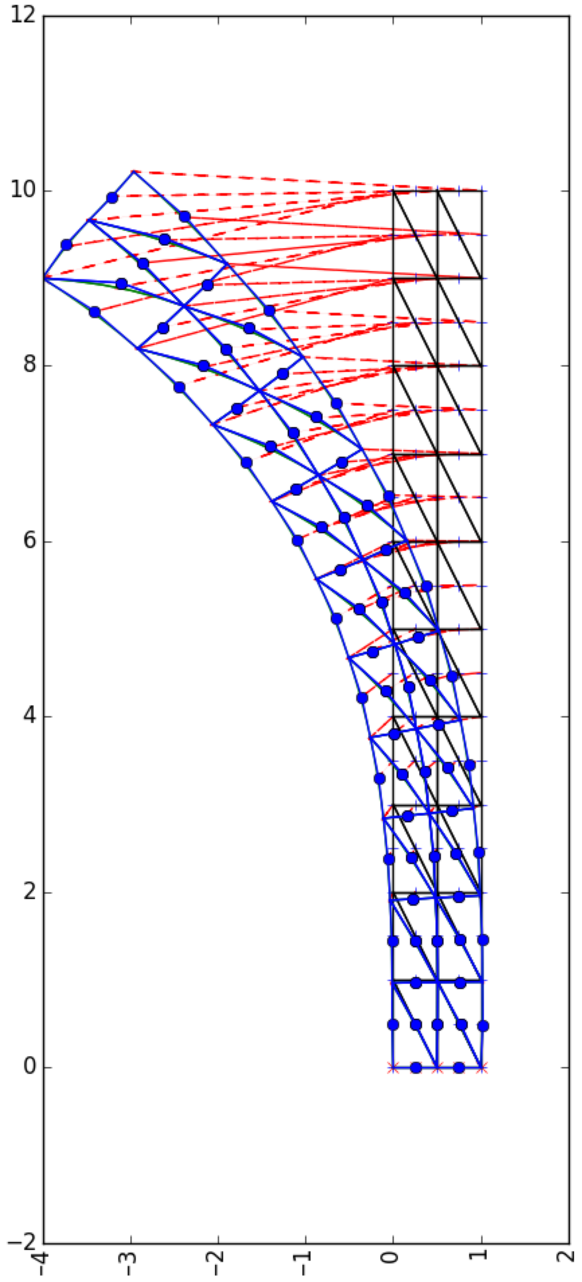


Fig. 4.7: Bending of a quadratic beam in Python.

The black beam represents the initial configuration of the beam. The blue beam has the same configuration as the black one except for the displaced point that causes a deformation. The red dashed lines represent the beam's displacement and the red crosses the restricting points.

Figure 4.7 shows how the beam is curved because of the intermediate nodes between each axis, provided by its quadratic representation.

In the following section, we present an approach of the quadratic tetrahedra implementation made in SOFA.

4.2 3-D Implementation - SOFA

In the following section, we explain the adaptations made in order to include the quadratic tetrahedra plugin implemented by Stefan Suwelack [11] to the most recent version of SOFA. The task was based on performing all the necessary processes to put the plugin back into operation.

First of all, it is important to consider that the elements represented in SOFA are stored by their ids in separate data arrays. That means that all the elements have their respective arrays defined: `seqPoints`, `seqEdges`, `seqTetrahedra`, `seqQuads`, `seqHexahedra`, among others. In this context, `seqQuadTetrahedra` was included in order to store the quadratic tetrahedra into the model.

Then, all the respective methods had to be adapted, including

- `addQuadTetra`
- `getNbQuadTetrahedra`
- `getQuadTetrahedra`
- `createQuadTetrahedra`
- `createQuadTetrahedraAroundNode`
- `createQuadTetrahedraAroundEdge`
- `createQuadTetrahedraAroundFace`

The transformation consisted in updating the deprecated methods used to read positions, to get elements coordinates, to include forces to the objects and to get the potential energy of the elements, among many others. Also, it was necessary to include new SOFA tools (BoxROI for quadratic tetrahedra and Dirichlet Boundary Constraints) and an updated version of the Mumps Solver[2].

The main improvements were made in the `Quadratic Mesh Topology` class, where the topology methods are implemented.

Considering the Quadratic Tetrahedra deformation problem, the steps used to simulate objects with quadratic tetrahedra were very similar to the ones implemented in Python:

[2] MUltifrontal Massively Parallel sparse direct Solver

- Obtaining the mesh (points placement, cells and cell types) from a VTK loader.
- Getting the Young's modulus and Poisson's ratio.
- Calculating every element stiffness matrix with the mesh, Young's modulus and Poisson's ratio, and then calculating the global stiffness matrix.
- Getting the Dirichlet condition; and getting points and grid.
- Combining the boundary conditions with the stiffness matrix to get a linear equation system.

After the code adaptation, we obtained a functional Quadratic Tetrahedra Plugin that allowed us to simulate objects represented by quadratic tetrahedra and beginning to analyze their behavior subjected to deformations.

In the following figure, the simulation of a quadratic tetrahedral torus with 680 elements, a Poisson's ratio of 0.4 and a Young modulus of 3000 is presented:

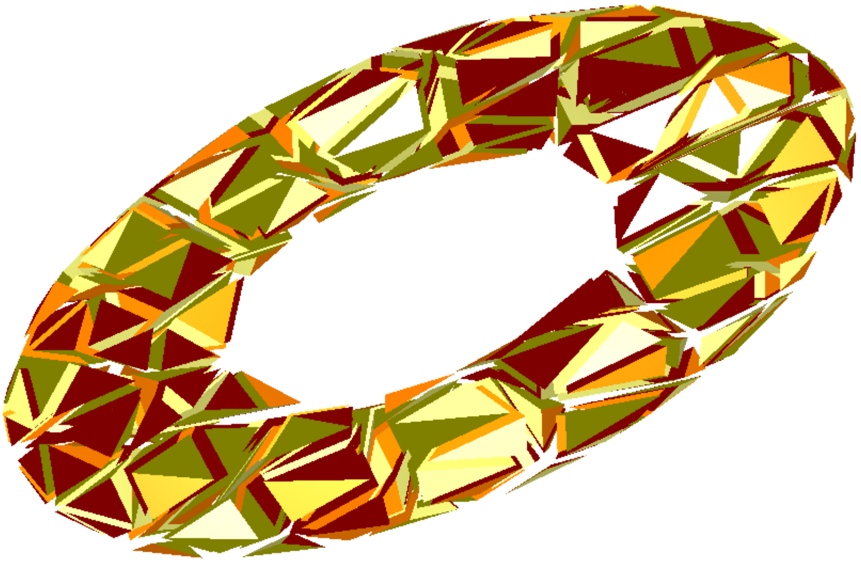


Fig. 4.8: Torus represented by quadratic tetrahedral elements.

The correctness of the implementation was verified by means of different simulations that were the same as those obtained in the old version of SOFA with the respective plugin.

After the code adaptation, quadratic tetrahedra were included in the CGoGN Plugin. In the following section, the process carried out to achieve it is explained.

4.3 3-D Implementation - CGoGN

As mentioned in section 3.2, the CGoGN library provides a very efficient implementation of combinatorial maps [15] due to the fast processing times and versatility of implemented operations. Data structures are saved as attributes related to each cell and updated efficiently, e.g. when a cell is deleted, the entry in the attributes is skipped and when a new cell of the same dimension is added, this entry is reused.

Maps rely on darts as their basic entity. According to the model and dimension, each dart has to store a variable number of topological relations (links to other darts) and indices of the embedded cells. For each embedded dimension, a variable number of custom attributes have to be associated to the indices of the cells.

Attributes are stored in chunk arrays [15] which are sets of chained arrays because it allows to allocate additional memory while leaving all existing elements in place.

In order to incorporate quadratic tetrahedra in CGoGN, some Quadratic Tetrahedra Plugin classes and methods were integrated. These included the handling of mass and force for quadratic tetrahedra, and a mesh loader, among others. These had to be adapted accordingly for proper operation.

Then, in order to be able to use the existing topology operations of CGoGN and its respective methods, some modifications were necessary. In the following section, these are presented.

4.3.1 Placement of points

In this section, we present the implementation carried out to obtain a correct representation of quadratic elements in CGoGN.

The main difference between linear and quadratic tetrahedra are the middle points so, instead of including an entire new topology to the plugin with all its methods and additional complications, we adapted the linear tetrahedra topology to save implementation time. In order to have an additional node on each edge of a quadratic tetrahedra, we used attributes to save the ids of the nodes on the edges, see figure 4.9 for a two dimensional example.

First of all, instead of defining the quadratic tetrahedra as a class like the other elements (triangles, linear tetrahedra, linear hexahedra, pyramids, etc.), we defined it as an array of size 10 with `PointID` as element type in the Combinatorial Map Topology of CGoGN.

Then, we declared the container of all quadratic tetrahedra as a vector of quadratic tetrahedra.

Subsequently, we created a method to initialize the quadratic tetrahedra. To do so, we first declared an *unsigned edge attribute* as a middle point for each volume edge of the linear tetrahedra topology. The `middlePoint` attribute incorporates the ID of the middle point as an `unsigned integer`, allowing then to go from 4 natural coordinates to 10 natural coordinates.

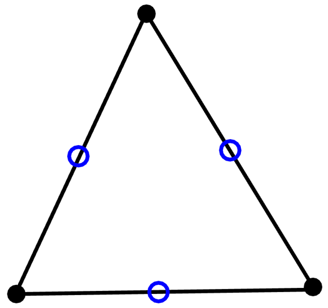


Fig. 4.9: Example of triangle with middle points.

In the figure above, we can see a triangle with blue nodes on the edges representing the attributes that save the middle points ids and black nodes representing the original triangle nodes.

Considering the following functions defined in CGoGN

- `phi1`: Traverse from one vertex to its right vertex.
- `phi2`: Traverse from one face to its right face.
- `phi_1`: Traverse from one vertex to its left vertex.

the quadratic tetrahedra w traversal is the following one:

```
vertices[0] = vertexID(phi2(w))
vertices[1] = vertexID(w)
vertices[2] = vertexID(phi1(phi1(phi2(w))))
vertices[3] = vertexID(phi_1(w))
vertices[4] = middlePoint[phi2(w)]
vertices[5] = middlePoint[phi1(phi2(w))]
vertices[6] = middlePoint[phi_1(phi2(w))]
vertices[7] = middlePoint[phi1(w)]
vertices[8] = middlePoint[phi_1(w)]
vertices[9] = middlePoint[phi_1(phi2(phi1(w)))]
```

Where `vertexID` is a function that returns the vertex id of a dart and `middlePoint` is the edge attribute containing its middle point id.

In order to keep track of the middle points, we created a Middle Point Map to store the middle point of each edge of each Tetrahedra.

Finally, we adapted all the affected methods to achieve a correct representation of quadratic tetrahedra, which included visualization functions, barycentric mapping functions, mass functions, among others.

In order to get the correct deformation of the Quadratic Tetrahedra, the steps described in section 4.2 were implemented using the modified topology.

In the next chapter, we explain the necessary adaptation of the linear remeshing methods in order to achieve a correct simulation of the cuts of objects represented by quadratic tetrahedra.

5. CUTTING

Virtual cutting of deformable objects is at the core of many applications in interactive simulation and especially in computational medicine.

Virtual cutting essentially involves three steps:

1. The update of the geometrical and topological representation of the simulation domain
2. The numerical discretization of the governing equations
3. The simulation of the deformable body being cut

The virtual cutting operations should guarantee a minimal quality of the FEM mesh at any time of the simulation since the numerical stability of the simulation is directly impacted by the quality of the mesh, at least in the context of finite element methods. Moreover, the computation time is directly impacted by the number of degrees of freedom in the discretized domain. Therefore, limiting the introduction of new elements or nodes during topological changes is an important requirement.

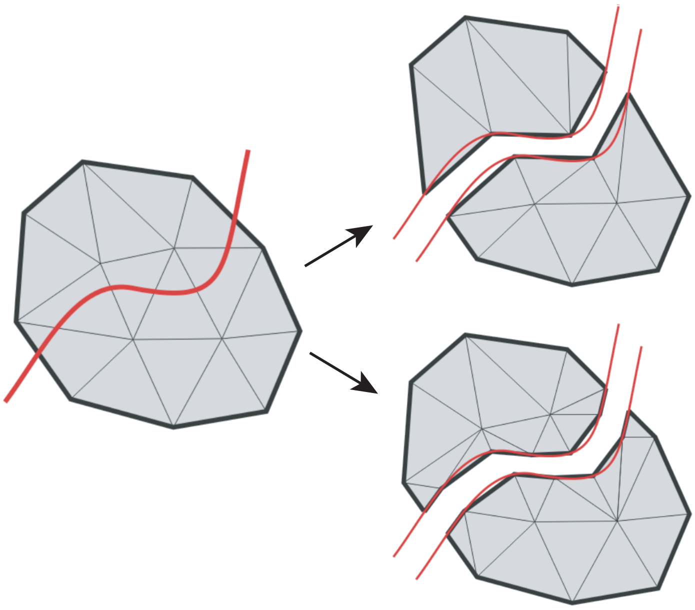


Fig. 5.1: Cutting with non-remeshing (top) vs remeshing (bottom).

As cuts occur in the simulation, the mesh that supports the FEM model has to be adapted so that the simulation takes account of the expected phenomena and produces the *mesh separation*. Figure 5.1 shows the result of a remeshed object and a non-remeshed object after a cut.

The *separating surface* S (red line in Figure 5.1) may be defined as the trajectory of a cutting tool or computed by a fracture algorithm describing the occurrences of tearing or shearing in the material. The tetrahedral elements traversed by this surface have to be cut, refined or rearranged to reflect the new physical state.

In order to approximate the cut, usual remeshing algorithms sample a cutting surface using the intersections with the edges (and in three dimensions potentially with faces) of the mesh, i.e. the boundary of the elements.

For our remeshing approach, we combine the Cutting remeshing presented in [20] with the CGoGN adaptation described in section 4.3 to provide the simulation of cuts with quadratic elements. To achieve it, points are not inserted on the boundary of the elements, but inside the volume. Thus, a cut is sampled by points in the volume, resulting in a higher freedom of the choice for their positions. Currently these positions are calculated based on geometric constraints, that are detailed and discussed in the following sections.

In the following section, we describe the steps implemented to achieve a correct quadratic element cutting procedure.

5.1 Remeshing

During the cut procedure, the mesh should closely approximate the intended separation surface. In order to achieve it, we adapted the remeshing method presented in [20] that introduces a separation surface inside a FEM mesh with two main benefits: the number of inserted vertices (nodes) and elements is kept as low as possible and the quality of the generated mesh is controlled.

When a surface S cuts the volume mesh, we first introduce vertices positioned on the surface S in the volume incident to crossed edges. Then, edges between the new vertices are inserted and finally new elements are introduced connecting the vertices previously inserted on the cut surface. In this work, we adapted those methods used to prepare the quadratic elements to be cut to the mentioned combinatorial map.

The position of inserted points on the separation surface is crucial for our method for the correct approximation of the cut surface and to obtain well-shaped elements.

Our algorithm inserts the middle point at the barycenter of the intersections between the cut surface S and the edges. That means, the cut is sampled in the volume of the element using positions of the cut on the element boundary. This choice potentially leads to an inaccurate approximation of the cut, so we decided to set the middle point positions on the cut inside of the element. However, this improvement only shows relevance for cuts that have strong changes in direction inside an element.

In what follows, we describe the technique used to reach a good remeshing in 2-D in the first place and in 3-D in the second place.

5.1.1 Remeshing in 2-D

In order to facilitate the comprehension of the three-dimensional remeshing methods, we begin by explaining the process in two dimensions.

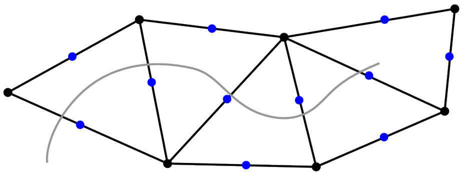


Fig. 5.2: Object being cut.

The first step is the sampling or detection of the separation surface at the level of the edges of the FEM mesh. For each edge e of the mesh, we compute or estimate a cutting or breaking point and the normal of the separation surface at this point, which we use for the placement of the points inserted to represent the cut in the mesh.

An edge that crosses the separation surface indicates that incident volumes are cut by the surface S. To take account of that, we introduce vertices in the triangle incident to crossed edges. The newly introduced vertices, denoted p_k, are positioned on the surface S, as shown in Figure 5.3.

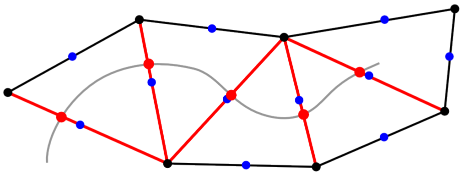


Fig. 5.3: Edges being detected.

The second step is the insertion of the triangles connecting the newly introduced vertices, as presented in Figure 5.4 for one volume and in Figure 5.5 for all the affected volumes.

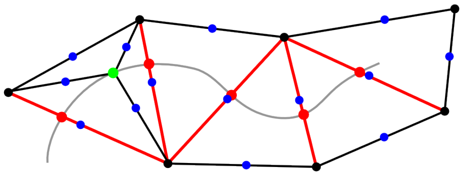


Fig. 5.4: One split 1 to 3.

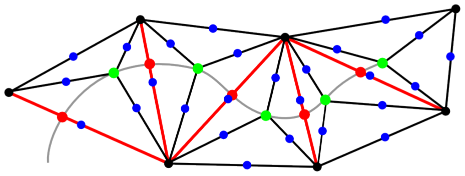


Fig. 5.5: Split 1 to 3.

Then, the affected triangles are flipped as can be seen in Figure 5.6 for one volume and in Figure 5.7 for all the affected volumes.

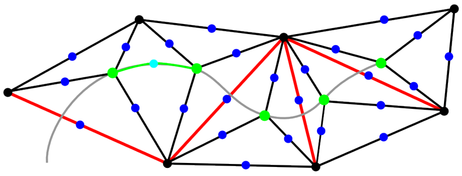


Fig. 5.6: One Flip 2-2.

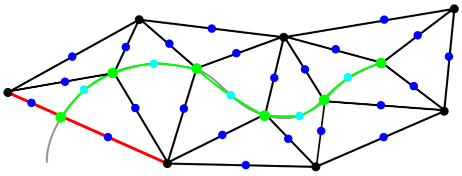


Fig. 5.7: Flip 2-2.

It is important to note that the described methods add a limited amount of new elements: 3 per cut triangle.

We just presented how triangles emerge from the initial mesh after a series of splits and flips on the quadratic elements, in order to get a local remeshing of the model and to approximate the separation surface in 2-D. In the following section, the 3-D linear case and its quadratic transformation are analyzed.

5.1.2 Remeshing in 3-D

In the CGoGN Plugin, every topology has its own methods and ways of being managed. In section 4.3, we presented a way to include quadratic tetrahedra without adding a new topology. In this section, we explain how this adaptation is used to cut objects represented by quadratic tetrahedra.

To achieve an accurate cutting representation of quadratic elements, we adapted the 3-D linear remeshing methods by including a middle point to every affected edge in order to maintain the essence of the mentioned topology.

To include the middle points in the remeshing methods, we considered the following:

1. Since every point is identified by an index, we created a method to keep track of every unused index to avoid duplicates. This method is used to get a free index every time a point is created.
2. Every edge incident to an affected dart needs a middle point, so these were added to their combinatorial map topology.

In what follows, we explain the remeshing algorithms adapted in order to achieve an accurate cut simulation.

- **Split 1 to 4:** The first step is to subdivide every tetrahedron that is crossed, even partially, by the separation surface S. The 1-4 split replaces the initial tetrahedron by four new tetrahedra sharing the inserted vertex p_k, as shown in Figure 5.8.

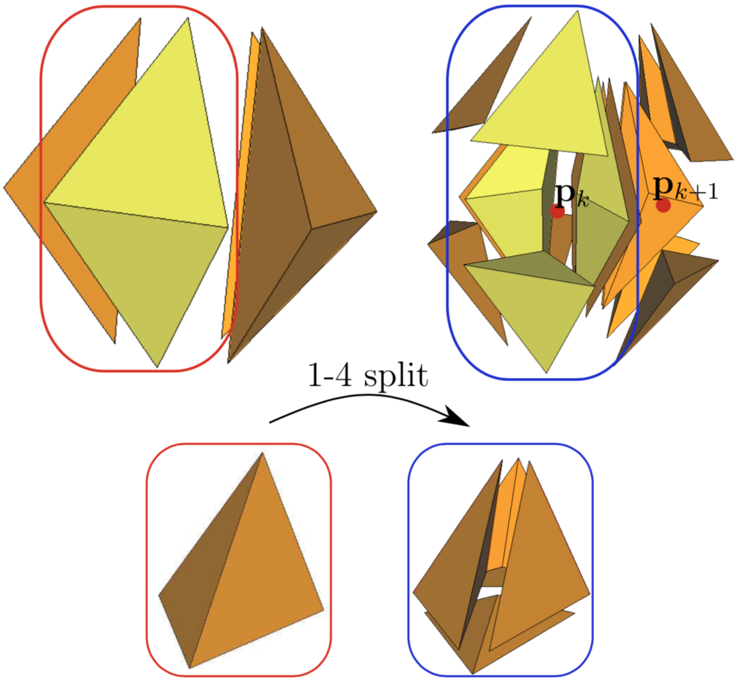


Fig. 5.8: A set of tetrahedra incident to an edge before (top left) and after (top right) the 1-4 split; 1-4 split of a tetrahedron (bottom). Figure extracted from [22].

The position of the middle point inserted in the 1-4 split operation in partially cut elements is not restricted to intersections of the tetrahedral element boundary and the cut front, but can be displaced along the cutting front inside of the element.

As the remeshing of partially cut elements is prone to yield illshaped elements, we choose to place the point at the intersection with the edges between a point in the barycenter and the corners of the element. While this improves the condition number, it has a negative impact on the approximation of the cut surface, as the partial cut is performed either too far or not far enough. This could be improved while preventing flat elements with a 1-4 split of neighboring elements followed by a flip 2 to 3 as shown in Figure 5.10.

In what follows, the middle points in volumes are noted p_m, the middle points in the new edges are noted $p_{m,i}$ (i is the point id) and the coordinates are noted φ.

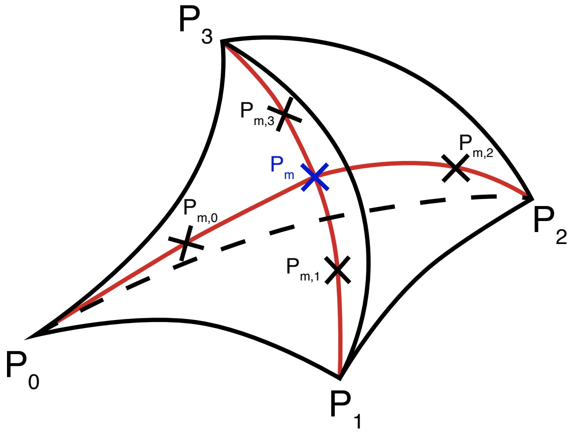


Fig. 5.9: Tetrahedron with its barycenter point (blue) and intersection edges (red). Midpoints on original edges are not shown for visual clarity.

For example, in the tetrahedron of Figure 5.9 we consider

$$\varphi_0 = (1,0,0,0) \qquad \varphi_1 = (0,1,0,0) \qquad \varphi_2 = (0,0,1,0) \qquad \varphi_3 = (0,0,0,1)$$

Then, $\varphi_m = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ and $\varphi_{m,j} = (\varphi_m + \varphi_j)/2$ for $j \in \{0,1,2,3\}$.

We define $(\varphi_m)_e = \frac{1}{4}$ for $e \in \{0,1,2,3\}$ and, in order to obtain every new middle point coordinate, the following process must be performed:

$$\begin{aligned} (\varphi_{m,1})_1 &= ([\varphi_m + \varphi_1)/2]_1 \\ &= ([(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}) + (0,1,0,0)]/2)_1 \\ &= (\frac{1}{4} + 1)/2 \\ &= \frac{1}{8} \end{aligned}$$

Generalizing,

$$(\varphi_{m,j})_e = \begin{cases} \frac{1}{8} & \text{if } e \neq j \\ \frac{5}{8} & \text{otherwise} \end{cases} \tag{5.1}$$

- **Swap 2 to 3:** The second step aims at creating edges between the inserted vertices. Let us consider the tetrahedra generated by the 1-4 splits around the initial edges e. They form a sequence of pairwise adjacent tetrahedra. Each pair of tetrahedra t_k, t_{k+1} share a face f, incident to e, and thus three vertices. Their fourth vertices are respectively p_k and p_{k+1}.

  We perform a Swap 2-3 (or flip) around the face f, replacing two adjacent tetrahedra by three tetrahedra with the same vertices. The shared face is deleted, but the three new tetrahedra share the edge $\{p_k, p_{k+1}\}$. This way, the faces initially incident to e are replaced by a sequence of edges. These edges form a closed polygon lying on the surface S (see figure 5.10). Finally, new edges middle points are inserted at the barycenter between the tetrahedral nodes and adjusted to the surface.

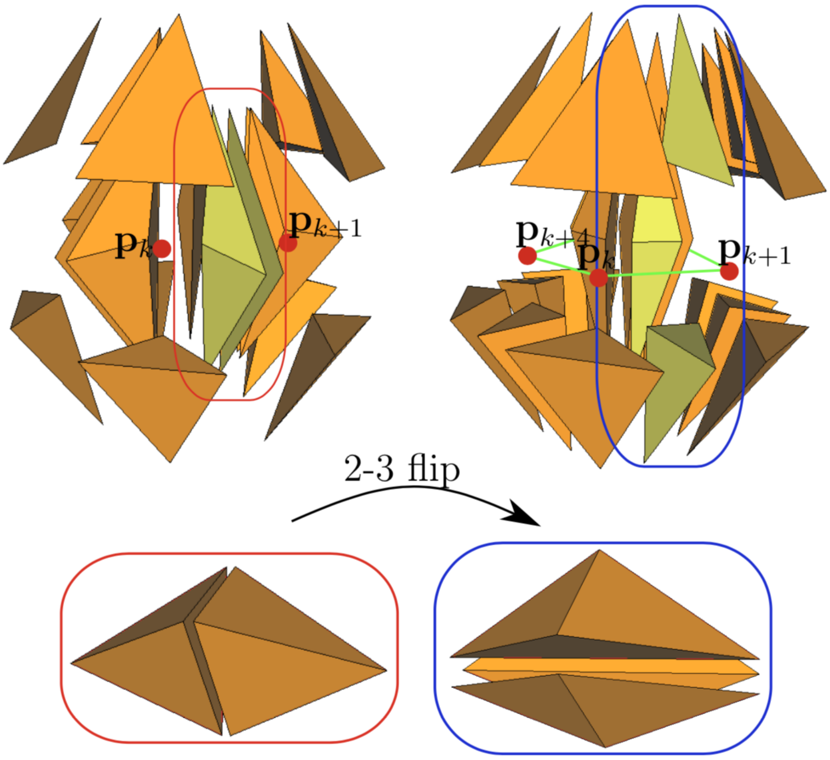


Fig. 5.10: A set of tetrahedra before (top left) and after (top right) the 2-3 swaps. Flipping the faces incident to e creates a closed polygon $\{p_1, ..., p_n\}$ (draw in green); (bottom) 2-3 swap of two tetrahedra. Figure extracted from [22].

In order to perform the 2-3 swap, the connection between $\{p_k, p_{k+1}\}$ has to be completely inside the two neighboring tetrahedra. The vertices $\{p_k, p_{k+1}\}$ inserted by the 1-4 splits both depend on the intersection of the separation surface with the tetrahedras' edges. This choice helps to ensure that the 2-3 swap can be performed.

- **Remove Edge:** Each crossed edge e is now surrounded by a set of n tetrahedra that contain the vertices of the crossed edge and two points of the polygon $\{p_1, ..., p_n\}$. The last step builds a set of n-2 triangles by triangulating the polygon $\{p_1, ..., p_n\}$.

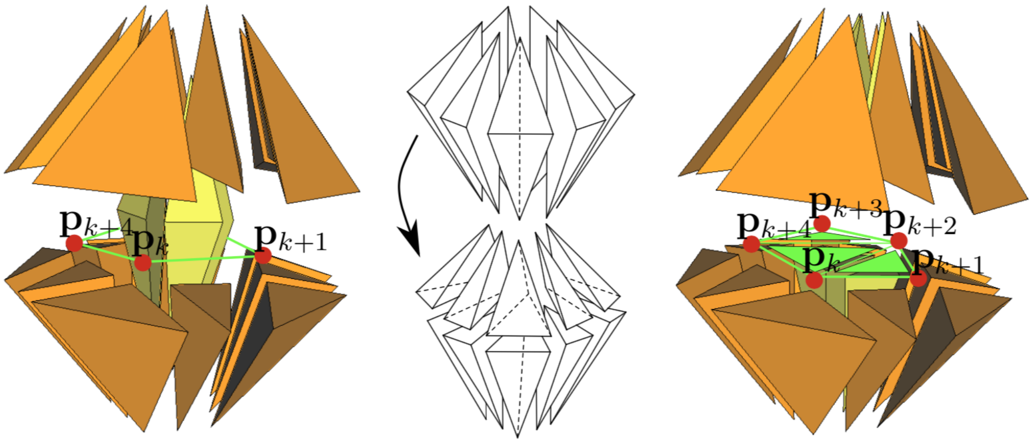


Fig. 5.11: To be cut edge surrounded by tetrahedra in yellow (left); edge removal (middle); triangulation of the separation surface (right), showing the tetrahedra below the cut. Figure extracted from [22].

As the vertices $\{p_k\}$ are placed to sample the separation surface, the triangles approximate the separation surface by construction. The n tetrahedra surrounding the edge are replaced by $2(n$ - $2)$ tetrahedra defined by the three vertices of each triangle and by one of the vertices of e (figure 5.11). This operation is called a general flip or an edge removal. For newly inserted edges inside of the object, middle points are inserted on the straight barycenter of the two edge nodes.

When the separation surface crosses the boundary of the simulated object, a specific refinement of the boundary tetrahedra is needed to build this *cut line*. The additional refinement combines the previous tetrahedral remeshing with an extended interpolatory refinement (called $\sqrt{3}$-subdivision [4]) of the boundary triangles.

First, the triangles of the boundary surface are remeshed this way: The edges that are crossed by the cut line are first selected. New vertices are inserted on the cut line in the adjacent triangles that are split into three. Then the selected edges, except the boundary edges, are flipped to link the new vertices. The flipped edges define a polygonal line that smartly approximates the cut line. The boundary edges are finally split at their intersections with the cut line.

Then, the boundary tetrahedra's remeshing consists of the following two steps:

- **Split 1 to 3:** This operation is similar to the *Split 1 to 4* but with three tetrahedra. We obtain a polygon of p_k with vertices on the separation but, in contrast to the inner edges, that polygon is opened, between the points inserted by the 1-3 split.

For quadratic shape functions, the position of the middle point is as essential as the edge points themselves. When displacing the middle point along the curve of the edge, the continuous representation changes greatly [22]. Thus when cutting a boundary edge, special attention has to be paid to set the points in the interpolated edge barycenter between the cut point and the edge points.

The middle points on the edges of a quadratic triangles and tetrahedra have the same importance and change the elements shape if not correctly placed. Therefore cut boundary triangles need a special treatment for the split 1 to 3 operation.

The split 1 to 3 point P_{13} (see Figure 5.14) is inserted at the intersection of the cut with the curve connecting the interpolated triangle barycenter with the nodes. Then, edge points are inserted as the interpolated triangle barycenter between P_{13} and the triangle corner points.

An example of this procedure in 2-D can be observed in the following figures:

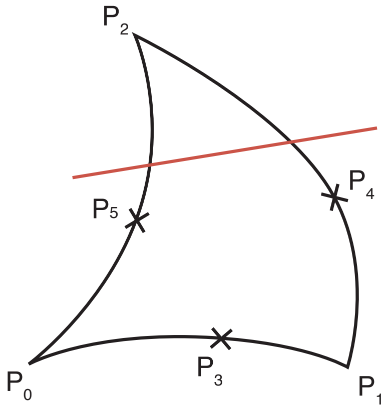


Fig. 5.12: Cut scenario.

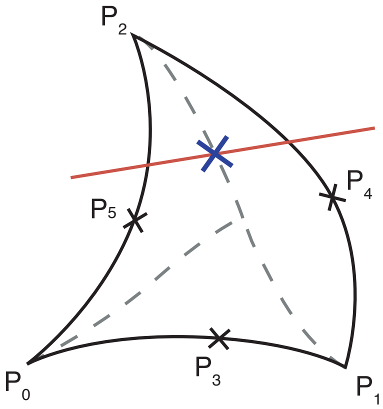


Fig. 5.13: Insertion of the split 1 to 3 point.

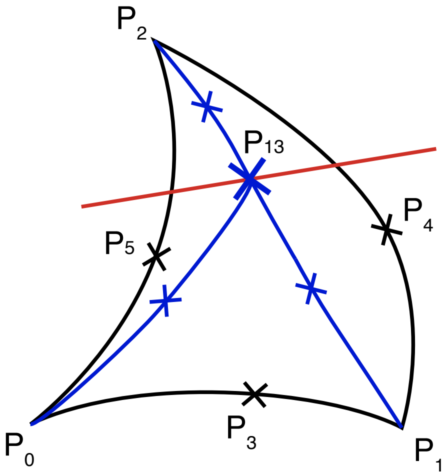


Fig. 5.14: Split 1 to 3 on surface, new edges and points in blue. Midpoints ids of internal edges are not shown for visual clarity.

In the first place (Figure 5.12), we can observe a cut triangle. Then, in Figure 5.13, we can see how the split 1 to 3 point is inserted. Finally, Figure 5.14 shows the split 1 to 3 result, including the new edges and points.

The split 1 to 3 is followed by a bisection of the boundary edges intersecting with the cut, explained in the next point.

- **Bisect Edge:** In this step, an intersection point p is inserted on the edge. Then the tetrahedron is split into two by inserting a node at the interpolation edge barycenter between p and the nodes of the cut edge. New edges are inserted between the other nodes of the tetrahedron and the intersection point p. These new edges use the intersection between the curve between the two existing edge middle points on the surfaces, that are adjacent to the cut edge. The new vertex p is used to close the polygon opened by the split 1 to 3.

  The following figure illustrates the process:

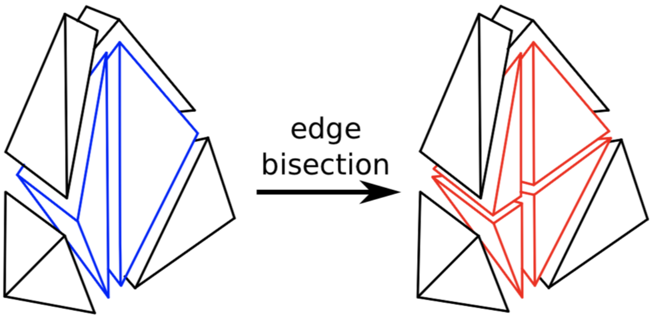


Fig. 5.15: Bisection of a boundary edge: the incident tetrahedra are split. Figure extrated from [20].

In this case, as most newly inserted edges lie inside of the object, we insert the edge points on the straight barycenter of the two edge points.

Note: This method does not support partial cuts because while the integration in completely cut elements can be handled with algorithms accurately, the integrals in partially cut elements can only be approximated, which means that we can't make a difference between these two cases:

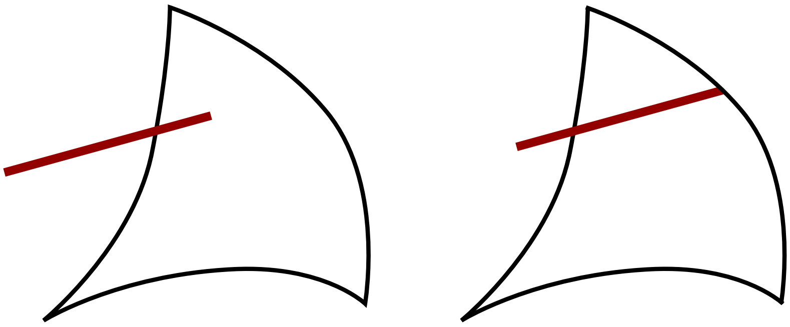


Fig. 5.16: Triangle with a partial cut (left) and a complete cut (right).

Besides, our expansion of the Gauss integration has high computational cost and we do not expect such an implementation to run in real-time. Moreover, the cut surfaces inside a cut element do not maintain the δ-property, and thus a treatment of boundary conditions on these surfaces needs particular methods.

5.2 Cutting

Cutting can be described as a controlled fracture process performed through a precisely directed path, exerted through sharp-edged devices or laser beams. This process requires topological updates and computation of data that will influence the physics of the object.

After remeshing the elements, the nodes of the cutting separation are duplicated in order to allow the surface opening and separating in the following way: As soon as a cycle of adjacent faces, incident to a same vertex, are disconnected, the vertex is separated into two vertices (i.e. it is duplicated) triggering an update of the FEM part. This last condition is automatically checked by the CGoGN library [22].

The methods to achieve it [20] were adapted to work on a quadratic element's representation.

The following is an example of a quadratic beam being remeshed and cut:

Beam properties:

Nb of elements	Solver	Iterations	Threshold	ν	E	Force
24	CGLinearSolver	100	1.0e-9	0.4	3000	(0, 0, 2000)

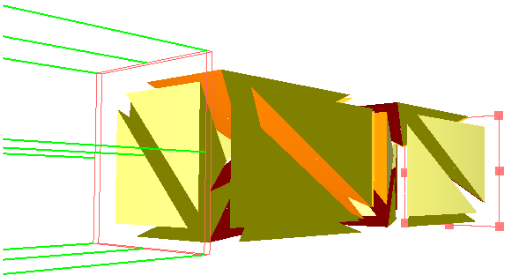


Fig. 5.17: Beam before cut.

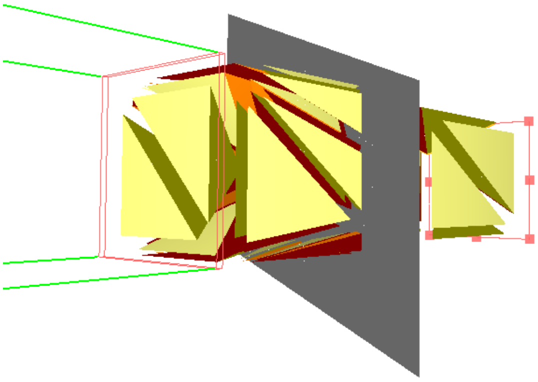


Fig. 5.18: Beam during cut.

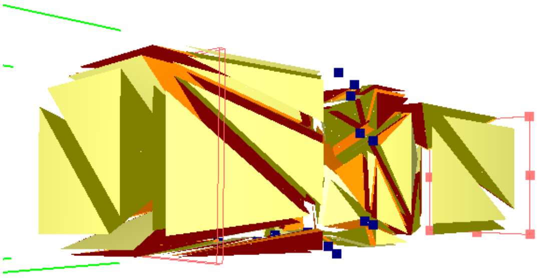


Fig. 5.19: Beam after cut.

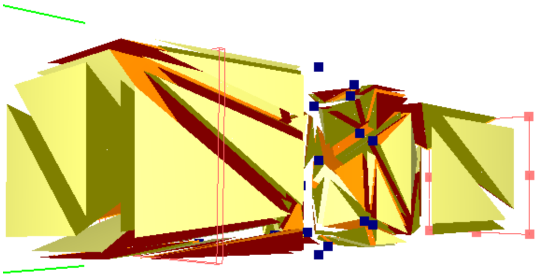


Fig. 5.20: Beam after 0.5s from the cut.

We can observe how the number of tetrahedra is multiplied after the cut due to the remeshing.

In the following chapter, the results of the exhibited implementations are presented and analyzed.

6. RESULTS

In this section, we present the results obtained after some experimentation.

When necessary, the meshs for the experimentation were created with Gmsh 3.0[1], set in *3D* and with *set order 2* to get quadratic elements.

6.1 Convergence Analysis

The convergence of the quadratic tetrahedra implemented was studied with the numerical test presented by Kwon et al [6].

This consists of simulating beam bendings subjected to different configurations and calculating their vertical displacement in order to compare it to a numerical approximation.

Beams with different amounts of quadratic tetrahedra were evaluated in order to study if convergence was reached for any of them. Those were:

Nb of elements	24	97	371	712	1810	4621

Also, two kinds of beams were created: a thin and a thick beam:

- The dimensions of the thin beam were:

$$L = 1m \qquad h = 0.01m \qquad b = 0.01m$$

The following figure illustrates it:

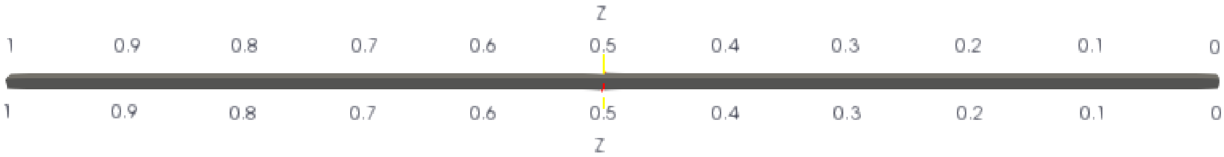


Fig. 6.1: Thin beam representation.

- The dimensions of the thick beam were:

$$L = 1m \qquad h = 0.1m \qquad b = 0.1m$$

The following figure illustrates it:

[1] http://gmsh.info/

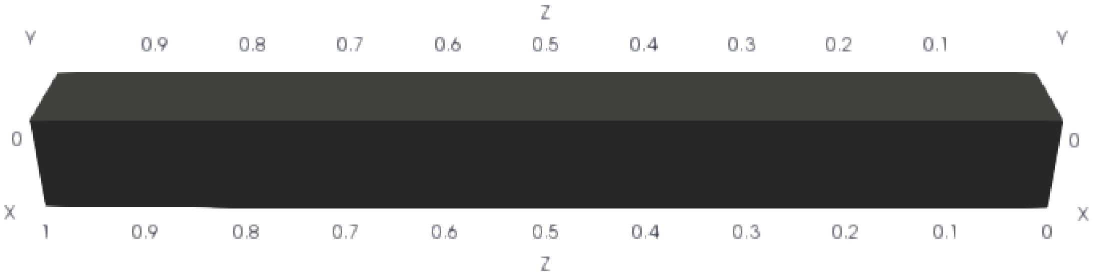


Fig. 6.2: Thick beam representation.

For this analysis, two different configurations (or scenarios) were considered. In what follows, the study carried out is detailed.

6.1.1 First configuration

The first test consisted of analyzing the following configuration:

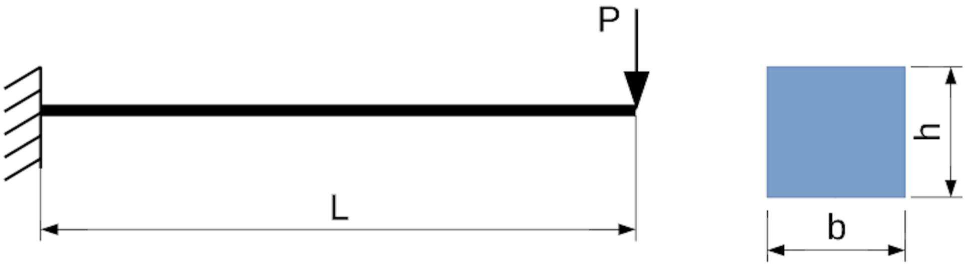


Fig. 6.3: Schematic representation of a bending test with a punctual load.

where P is the punctual load in Newton (N), L is the beam length in meters (m), h is the beam height in m and b is the beam base length in m.

Then, the obtained vertical displacements were compared to the results of the evaluation of the following equations:

$$\text{Thin beam: } w_{max} = \frac{PL^3}{3EI} \tag{6.1}$$

$$\text{Thick beam: } w_{max} = \frac{PL^3}{3EI} + \frac{PLh^2}{10GI} \tag{6.2}$$

where w_{max} is the maximum deflection in m, G is the shear modulus in $\frac{N}{m^2}$ and I is the moment of inertia in m^4, defined by:

$$G = \frac{E}{2(1+\nu)} \quad \text{and} \quad I = \frac{bh^3}{12} \tag{6.3}$$

The parameters chosen for both beam deflections were:

E	ν	Force (f)	Gravity	Solver	Tolerance	Threshold
209e9	0.3	1e-3	10	CGLinearSolver	1e-7	1e-7

Mass Density	Constant Force Field	Fixed Constraint
0.0	(0, 10, 0)	[-0.1, -0.1, -0.1]×[0.5, 0.5, 0.001]

These are standard and do not present any particularity. We decided not to modify them in order to focus the analysis on the beam resolution and on the precision this alteration may represent.

In what follows, the evaluated beams are presented.

1. **Thin beam**

   First, we define $f = \frac{P}{A}$, where f is the load in $\frac{N}{m^2}$ and A is the volume area in m^2. Then, $P = 10 * 0.01^2 = 1\text{e-}3N$.

   Also, $I = \frac{0.01^4}{12} = 8.333\text{e-}10m^4$ and, from 6.1, we get

   $$w_{max} = \frac{1\text{e-}3 * 1^4}{3 * 209\text{e}9 * 8.333\text{e-}10} = 1.914\text{e-}6m$$

   The thin beam vertical displacements obtained with the mentioned parameters were:

Nb of elements	24	97	371	712	1810	4621
Vertical displacement (in m)	6.583e-3	1.604e-7	1.464e-8	8.976e-9	8.885e-9	8.920e-9

   In the following figure, we can observe the relationship between the beam vertical displacement and the number of elements of the simulated beams with a logarithmic scale:

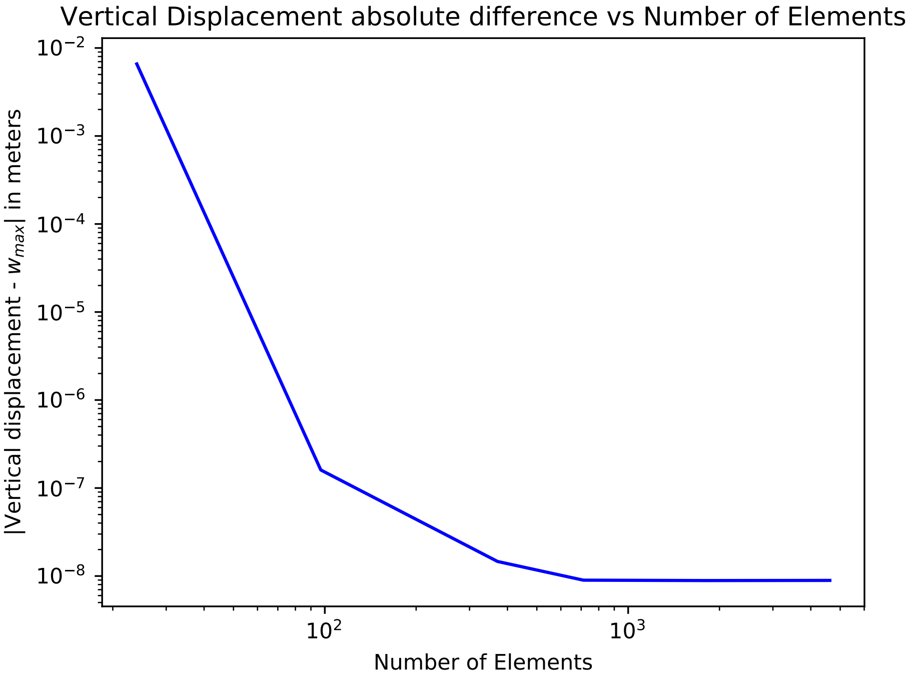


Fig. 6.4: Vertical displacement obtained with a punctual load on a thin beam.

We can observe that the vertical displacement decreases significantly as elements are added to the beam. In addition, the maximum analytical deflection is achieved with less than 100 tetrahedra. As convergence is reached with few elements, we can conclude that the algorithm is efficient in this scenario.

In what follows, we evaluate the same configuration for a thick beam.

2. **Thick beam**

In this case, we get $I = \frac{0.1^4}{12} = 8.333\text{e-}6m^4$, $G = \frac{209\text{e}9}{2(1+0.3)} = 8.038\text{e}10\frac{N}{m^2}$, $P = 10 * 0.1^2 = 0.1N$ and, from 6.2, we get

$$w_{max} = \frac{0.1 * 1^2}{3 * 209\text{e}9 * 8.333\text{e-}6} + \frac{1 * 0.1 * 0.1^2}{10 * 8.333\text{e-}6 * 8.038\text{e}10} = 1.929\text{e-}8m$$

The vertical displacements obtained with the mentioned parameters were:

Nb of elements	24	97	371	712	1810	4621
Vertical displacement (in *m*)	4.834e-9	3.019e-9	2.992e-9	3.008e-9	1.950e-9	1.032e-9

In the following figure, we can observe the relationship between the beam vertical displacement and the number of elements of the simulated beams with a logarithmic scale:

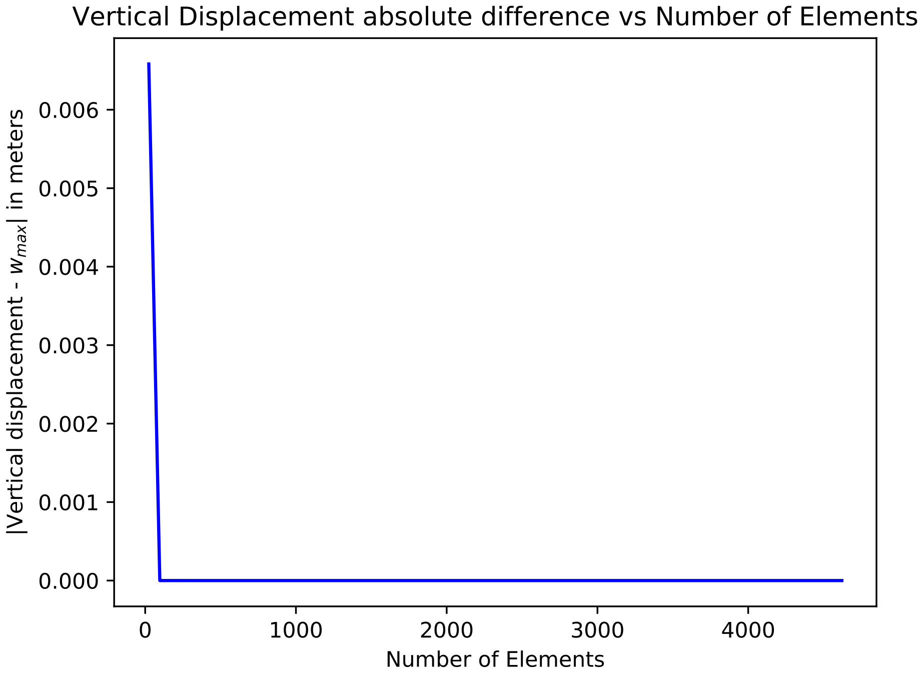


Fig. 6.5: Vertical displacement obtained with a punctual load on a thick beam.

As in the previous case, we can see that the vertical displacement decreases significantly as elements are added to the beam. In this case, we can observe that the maximum analytical deflection is achieved with less than 500 tetrahedra. Although the amount of elements necessary to reach convergence is greater than in the previous case, it remains low with respect to the number of elements that are used for medical simulation.

The next subsection presents the convergence analysis for a different scenario.

6.1.2 Second configuration

This test consisted in analyzing the behavior of a quadratic tetrahedral beam in the following scenario:

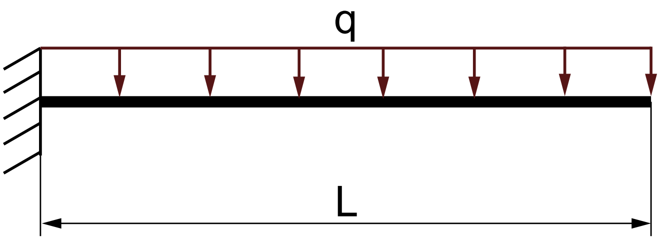


Fig. 6.6: Schematic representation of bending test.

where q is the distributed load in $\frac{N}{m}$.

The obtained vertical beam displacements were then compared to the results of evaluating the following equations:

$$\text{Thin beam: } w_{max} = \frac{qL^4}{8EI} \tag{6.4}$$

$$\text{Thick beam: } w_{max} = \frac{qL^4}{8EI} + \frac{L^2h^2}{20GI} \tag{6.5}$$

where $q = \rho g A$, ρ is the density mass in $\frac{kg}{m^3}$, g is the gravity in $N\frac{m}{s^2}$[2] and A is the volume area in m^2.

The parameters chosen for the tests of thin and thick beams were:

E	***ν***	**Force (f)**	**Gravity**	**Solver**	**Tolerance**	**Threshold**
209e9	0.3	1e-3	10	CGLinearSolver	1e-7	1e-7

Mass Density	**Fixed Constraint**
1000	[-0.1, -0.1, -0.1]×[0.5, 0.5, 0.001]

These are standard and do not present any particularity. We decided not to modify them in order to focus the analysis on the beam resolution and on the precision this alteration may represent.

1. **Thin beam**

   In this case, we obtained I = 8.333e-10m^4, $q = 1000 * 10 * 0.01^2 = 1\frac{N}{m}$ and, from 6.4,

$$w_{max} = \frac{1 * 1^4}{8 * 209\text{e}9 * 8.333\text{e-}10} = 7.177\text{e-}4m$$

   The vertical deformations obtained with the mentioned parameters were:

Nb of elements	24	97	371	712	1810	4621
Vertical displacement (in *m*)	8.869e-4	3.556e-8	1.504e-9	7.238e-10	2.000e-10	8.988e-11

[2] $1N = 1\frac{kg*m}{s^2}$

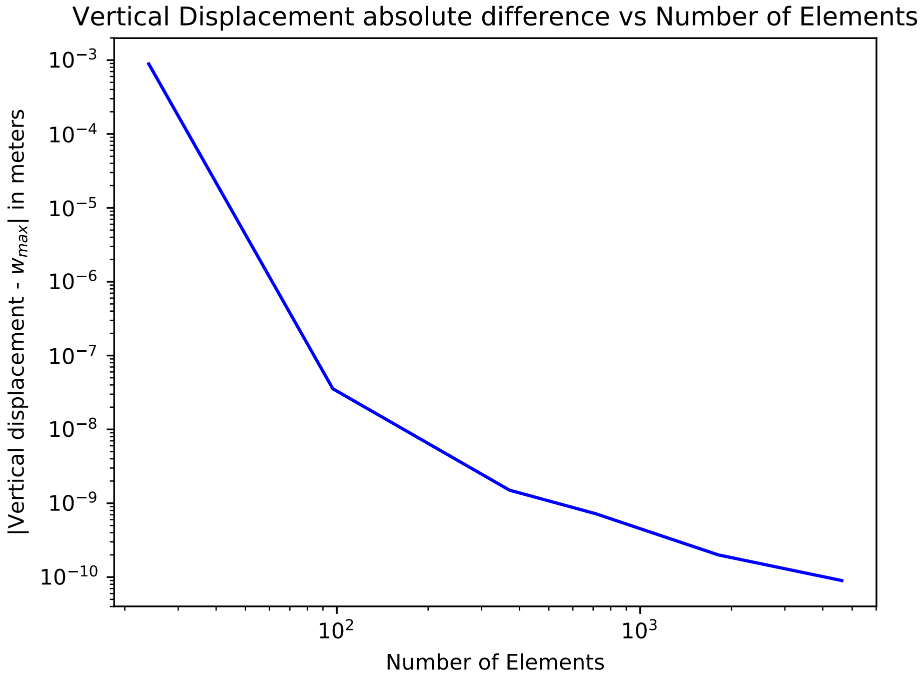


Fig. 6.7: Vertical displacement obtained with a distributed load on a thin beam.

We can observe that the vertical displacement decreases notably as the amount of elements of the evaluated beams increases. In this case, we can observe that the maximum analytical deflection is achieved with less than 50 tetrahedra, which is close to the lowest amount of elements tested. Then, convergence is reached more quickly than in the previous configuration.

In what follows, we evaluate the same configuration but for a thick beam.

2. **Thick beam**

In this case, we obtained I = 8.333e-6m^4, $q = 1000{*}10{*}0.1^2 = 100\frac{N}{m}$, $G = 8.038\text{e}10\frac{N}{m^2}$ and, from 6.5,

$$w_{max} = \frac{100 * 1^4}{8 * 209\text{e}9 * 8.333\text{e-}6} + \frac{1^2 * 0.1^2}{20 * 8.038\text{e}10 * 8.333\text{e-}6} = 7.178\text{e-}6m$$

The vertical deformations obtained with the mentioned parameters were:

Nb of elements	24	97	371	712	1810	4621
Vertical displacement (in m)	1.563e-7	1.042e-7	6.598e-8	7.786e-8	5.223e-9	2.000e-10

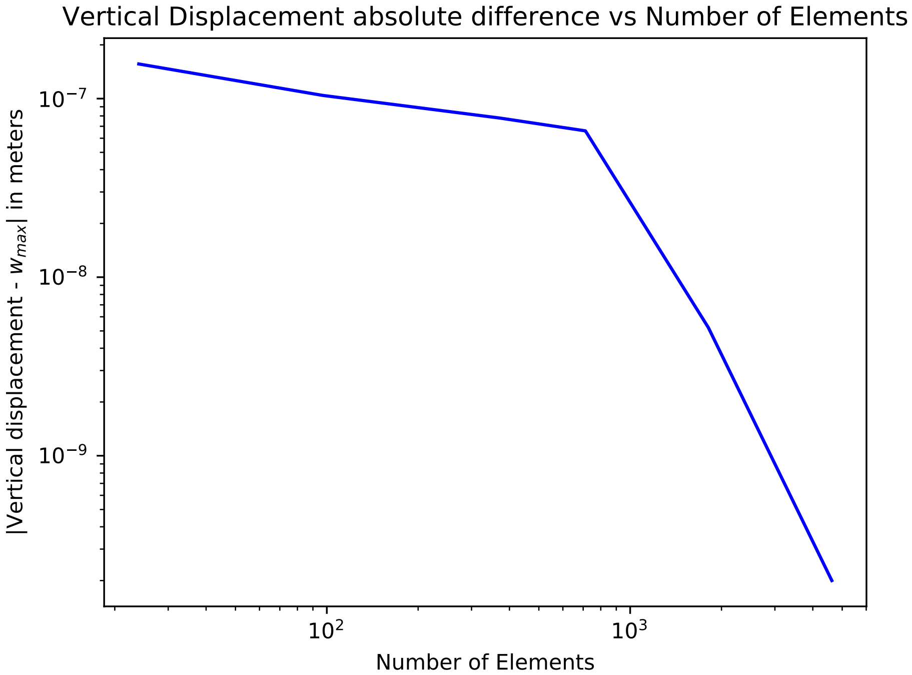


Fig. 6.8: Vertical displacement obtained with a distributed load on a thick beam.

> In this case, we can see that the vertical displacement slowly decreases at the beginning and when a sample of 100 elements is reached the decrease accelerates. On the other hand, we can observe that the vertical displacement is always above the maximum analytical deflection for all the simulated beams. Then, convergence is reached for any amount of quadratic tetrahedra over 24 under this configuration and this parameters.

In conclusion, we could observe that there is no convergence for all the quantities of quadratic tetrahedra evaluated (from 100 for the case of thin beams and 500 for the case of thick beams) in the first configuration. It begins to converge where the values are low with respect to the amount of elements that are used for medical simulations.

On the other hand, in the second configuration there is a convergence for almost all the quantities of elements evaluated.

In both cases, convergence is quickly achieved.

From this, we can conclude that the implementation is valid and efficient.

In the following section, we present an analytic comparison between the quadratic and linear tetrahedra implementations.

6.2 Comparison linear vs quadratic

6.2.1 Analytic comparison

In this subsection, a reference solution on a high resolution[3] quadratic mesh (100k elements) is computed in order to accomplish a quantitative analysis of the discretization error for each model (linear and quadratic). The reference solution is built by the api of a sofa simulation runner created by Stefan Suwelack[4] in MSML (Media Server Markup Language).

Then, test models of different resolutions are subsequently compared to this reference model. We chose the root mean squared (RMS[5]) error at the nodes of the reference solution as the error measures.

In this case, we consider the beam deformation under gravity shown in Figure 6.9 with the configuration described in 6.1.

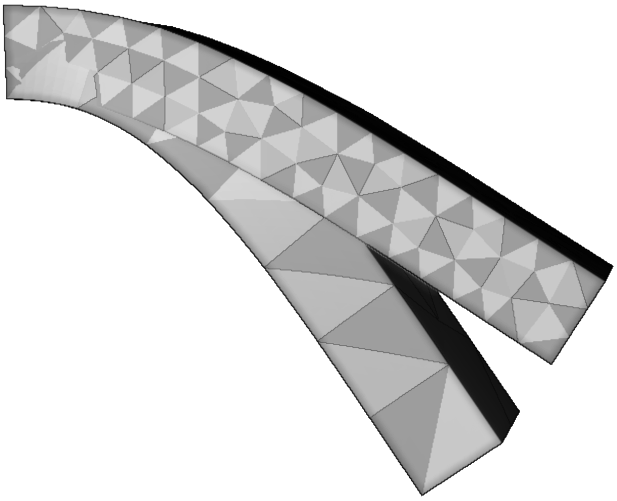


Fig. 6.9: Linear and Quadratic beams deformation under gravity. Both beams present the same number of DOFs.

E	ν	**Iterations**	**Gravity**	**dt**	**Solver**
300000	0.49	5	(-9.81, 0, 0)	0.5	Direct Linear Solver

Time Integration	**Processing Unit**	**Fixed Constraint**
Dynamic Implicit Euler	CPU	[-0.1, -0.1, -0.1]×[0.04, 0.07, 0.21]

Tab. 6.1: Parameters used to compare beam convergence.

[3] Numerically study consisting in doubling the resolution a couple of times in order to see when the results are only changing in an insignificant way.

[4] https://github.com/ssuwelack/msml-docker-runtime

[5] $RMSError = \sqrt{\frac{\sum_{i=1}^{n}(\bar{y_i}-y_i)^2}{n}}$ where y_i is the observed value for the ith observation and $\bar{y_i}$ is the predicted value.

The number of tetrahedral elements evaluated are:

Nb of elements	371	712	1810	4621	8990	19128	45418

The RMS errors with respect to the number of elements are depicted in Figure 6.10.

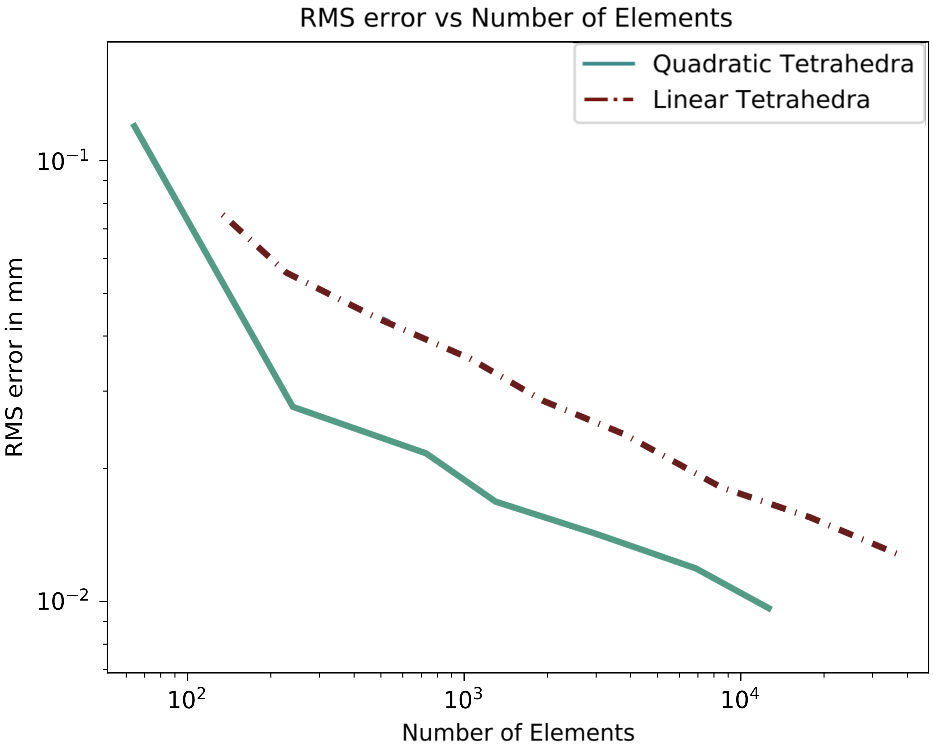


Fig. 6.10: RMS error over number of elements for gravity induced deformation.

The quadratic elements show superior accuracy for the same tetrahedra than the linear elements.

In the case of the gravity induced deformation, linear elements need much more tetrahedra (up to 40x) than quadratic elements in order to achieve the same accuracy.

6.2.2 Time comparison

In this subsection, we study the runtime difference between quadratic and linear beams deformations subjected to gravity force.

To do so, we used the following configuration:

E	ν	**Iterations**	**Gravity**	**dt**	**Solver**
300000	0.49	5	(-9.81, 0, 0)	0.5	MumpsSolver

Time Integration	**Processing Unit**	**Fixed Constraint**
Dynamic Implicit Euler	CPU	[-0.1, -0.1, -0.1]×[0.04, 0.07, 0.21]

Tab. 6.2: Parameters used for time comparison.

The Number of elements considered were:

Nb of elements	24	97	371	712	1810	4621	8990	19128	45418

In the case of quadratic beams, more than 4621 tetrahedra are difficult to compute, then they are not considered.

The result of evaluating the deformation of beams subjected to gravity with different number of elements with respect to the time of the simulation is the following:

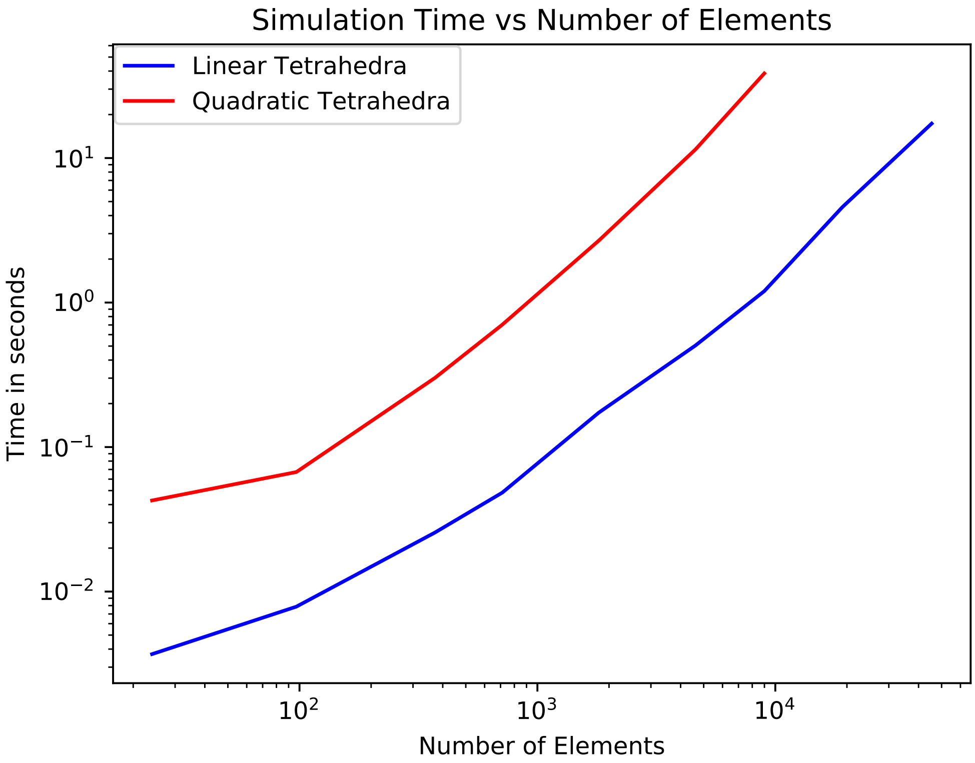


Fig. 6.11: Time comparison.

We can observe that the simulation time of beams under gravity is smaller in the linear case than in the quadratic one for any number of elements. This occurs because, for the same number of elements, the number of nodes is greater in the quadratic case, as shown in Figure 6.12. Then, the number of operations to be performed is greatly increased, delaying the simulation process.

The following figure shows the proportion of nodes of a linear and a quadratic tetrahedral beam depending on the number of elements:

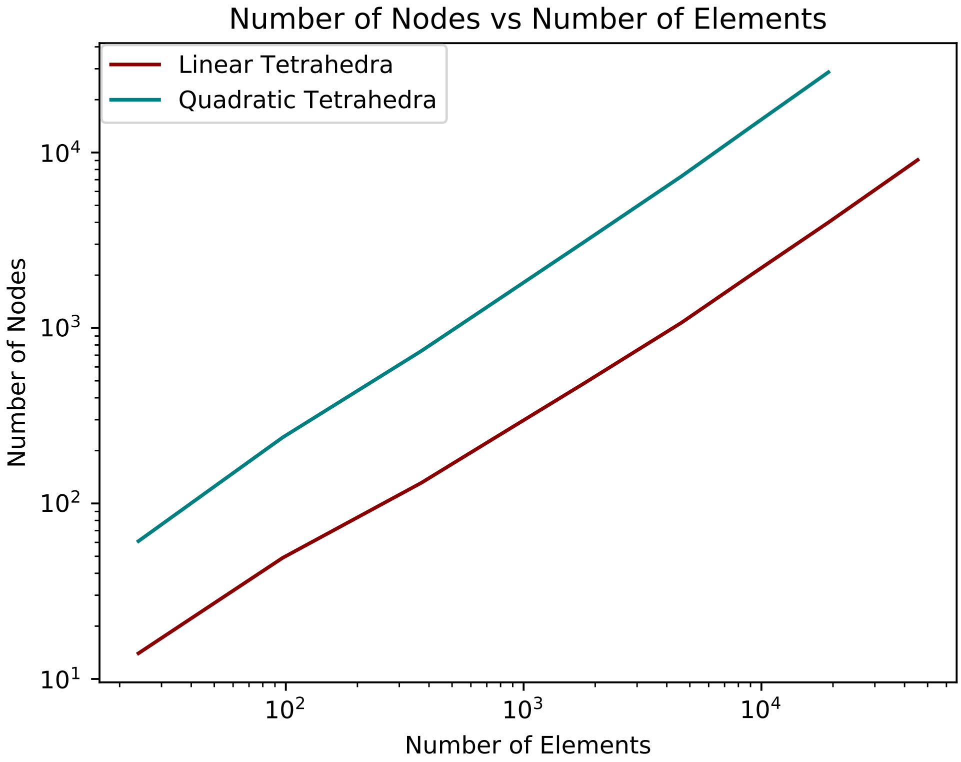


Fig. 6.12: Number of nodes vs Number of Elements.

By comparing Figure 6.11 and Figure 6.12, we observe that the simulation time is almost directly related to the number of nodes of the object, confirming the previous statement.

6.3 Visual comparison

In order to have a visual idea of the deformation convergence, we compared linear and quadratic beam simulation results.

To do so, we defined quadratic (in yellow) and linear (in blue) beams with different numbers of elements and checked their similarity while subjected to the same force.

The configuration used for the beams deformations is:

Solver	**Iterations**	**Tolerance**	**Threshold**	ν	E	**Force**
CGLinearSolver	5000	1e-7	1e-7	0.0	3000	(0, -9, 0)

And the Boundary Conditions are:

Force Boundary Conditions Box	[-0.1, -0.1. 1] × [0.5, 0.5. 1.001]
Displacement Boundary Conditions Box	[-0.1, -0.1. -0.1] × [0.5, 0.5. 0.01]

- With 24 elements:

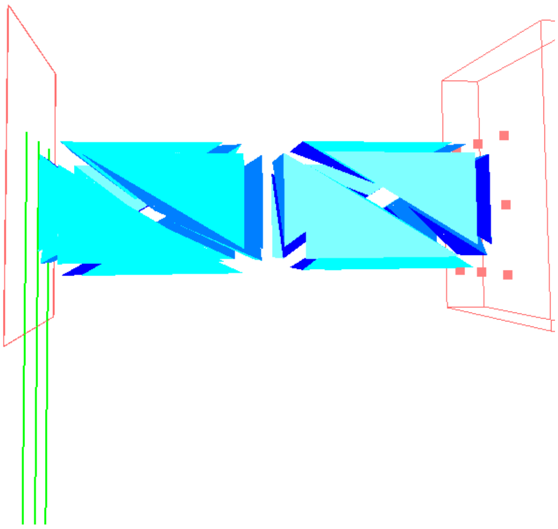


Fig. 6.13: Before force application.

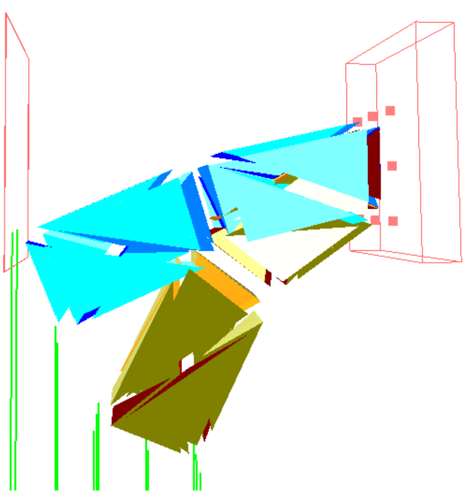


Fig. 6.14: After force application.

- With 97 (left) and 712 elements (right)

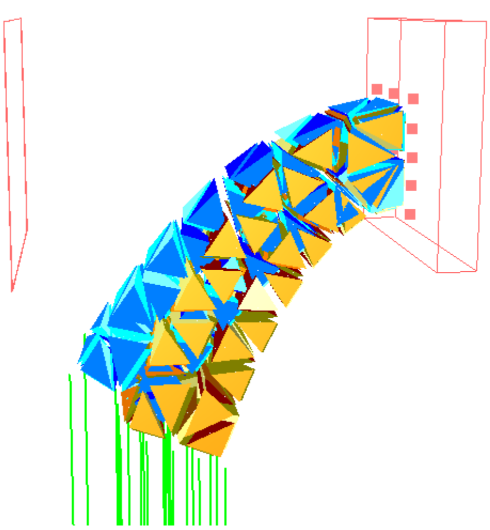


Fig. 6.15: Linear and quadratic beams deformations with 97 elements.

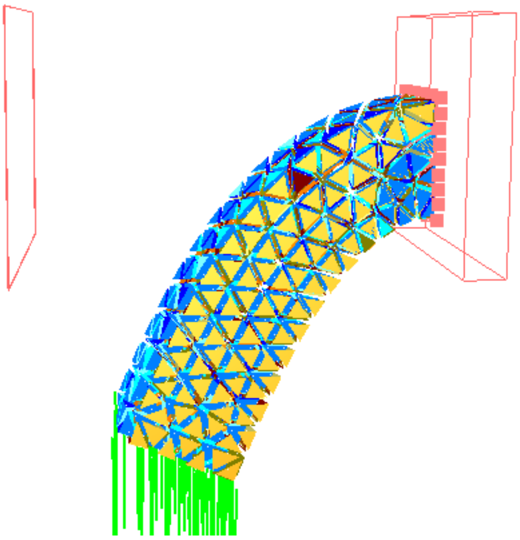


Fig. 6.16: Linear and quadratic beams deformations with 712 elements.

- With 1810 DOFs

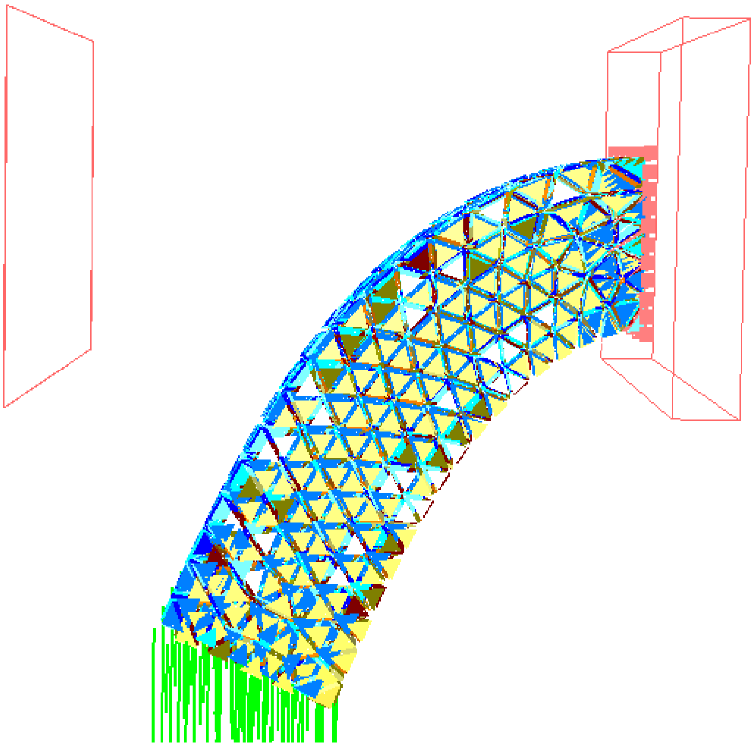


Fig. 6.17: Linear and quadratic beams deformations with 1810 elements.

The previous figures show the comparison between deformations of linear and quadratic beams subjected to forces for different number of elements, starting from the same initial configuration.

In Figure 6.14, we can see that the quadratic beam deformation is much more pronounced than the linear one for 24 elements.

In the case of 97 elements (Figure 6.15), the difference is still remarkable but not as pronounced as in Figure 6.13.

In Figure 6.16, we can perceive that the difference between the deformations of linear and quadratic beams is less important than for the previous cases. This can also be observed in Figure 6.17, for 1810 elements.

We can see that the difference between the deformations with 712 elements and 1810 elements is not significant.

From this, we can conclude that despite having a low number of elements, the beam deformation is outstanding in the case of quadratic tetrahedra, but not in the case of linear tetrahedra. Only when a sample of 712 elements is reached, the beam deflection simulation is not so different for one type of elements than for the other, even though the quadratic is still better in all cases.

6.3.1 Comparison between elements of the same type

In the following figures, we can see the visual comparison between beams deformations with quadratic and linear elements with the following configuration:

Nb of elements	Solver	Iterations	Tolerance	Threshold	ν	E	Force
24 vs 712	CGLinearSolver	5000	1e-7	1e-7	0.49	3000	(0, -9, 0)

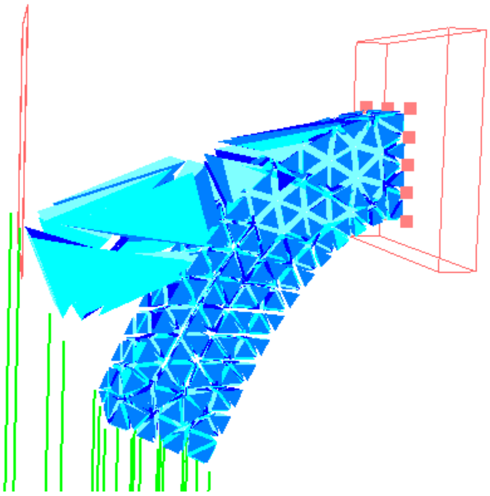


Fig. 6.18: Linear beam comparison.

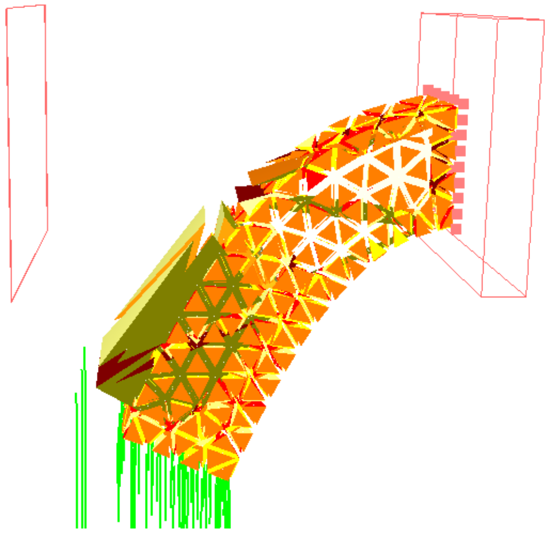


Fig. 6.19: Quadratic beam comparison.

In this case, we can see that the linear beam deformation with 24 tetrahedra is noticeably different from the one with 712 elements. On the other hand, the difference between the quadratic beam deformation with 24 elements and the one with 712 elements is not that substantial. Then, we could visualize that the quadratic representation converges much faster than the linear one.

6.4 Cutting results

This section contains a theoretical analysis of the number of nodes added during the cut process and the experiment results obtained.

6.4.1 Theoretical analysis

The following theoretical analysis presented by Paulus et al [22] helps to determine the impact of the remeshing algorithm on the FEM simulation, as solving the resulting linear system depends (at best) linearly on the number of nodes in the new mesh [22]. Then, it is compared to the linear case.

We consider an object represented by quadratic tetrahedra where the cut traverses $n(v)$ tetrahedra. The number of added nodes on the cutting surface are noted $n(k_S)$ and the number of added nodes that are not on the cutting surface are noted $n(k)$. This distinction is necessary to allow for a derivation of the number of nodes, when the two parts above and below the cut are separated. We assume that an average of 5 tetrahedra are adjacent to each edge and that the cutting plane intersects all tetrahedra at three (case A) or four edges (case B). The reality lies between these two cases.

The number of cut edges are noted $n(e)$ and the number of cut faces $n(f)$. In case A, 3 edges are cut for each tetrahedron. Then, $n(e) = \frac{3}{5} \times n(v)$ and $n(f) = \frac{3}{2} \times n(v)$. In case B, 4 edges are cut for each tetrahedron resulting in $n(e) = \frac{4}{5} \times n(v)$ and $n(f) = \frac{4}{2} \times n(v)$.

In 6.3, we analyze the number of added nodes for the remeshing $n(k_S) + n(k)$ and for the remeshing with separation $2n(k_S) + n(k)$.

Based on the observations in the resulting objects after a cut, the number of newly inserted nodes calculated is:

Case	$\boldsymbol{n(k_S)}$	$\boldsymbol{n(k)}$	$\boldsymbol{n(k_S) + n(k)}$	$\boldsymbol{2n(k_S)+n(k)}$
A	$n(v)+n(f)+n(e)$	$[4n(v)]$	$7.1n(v)$	$10.1n(v)$
B	$\underbrace{n(v)}_{\text{split14}}+\underbrace{n(f)}_{\text{flip23}}+\underbrace{n(e)}_{\text{edge removal}}$	$\underbrace{[4n(v)]}_{\text{split14}}$	$7.8n(v)$	$11.6n(v)$

Tab. 6.3: Remeshing quadratic tetrahedra: the number of added nodes for the remeshing $n(k_S) + n(k)$ and for the remeshing with separation $2n(k_S) + n(k)$; nodes added on the edges, are surrounded by [].

Then, we compare it with the linear tetrahedra:

	Linear		**Quadratic**		**Quadratic/Linear**	
Case	$n(k_S) + n(k)$	$2n(k_S) + n(k)$	$n(k_S) + n(k)$	$2n(k_S) + n(k)$	$n(k_S) + n(k)$	$2n(k_S) + n(k)$
A	$n(v)$	$2n(v)$	$7.1n(v)$	$10.1n(v)$	7.1	5.1
B	$n(v)$	$2n(v)$	$7.8n(v)$	$11.6n(v)$	7.8	5.8

Tab. 6.4: Remeshing tetrahedra: comparison between linear and quadratic case.

The theoretical analysis on quadratic tetrahedra of Table 6.4 shows that there are at least 5 times more nodes inserted than in the linear tetrahedra. For the same accuracy, the number of elements is greatly reduced when using quadratic instead of linear shape functions. Similarly, the number of cut edges is greatly reduced. If we consider for example the subdivision of a quadratic tetrahedral element into eight linear tetrahedral elements, then there is potentially one additional edge cut per tetrahedra and two per face. That means, when cutting $n(e)$ quadratic edges we can expect $2n(f) + n(v)$ cut linear edges. Thus, for the first case we get $2 * \frac{3}{2}n(v) + n(v) = 4n(v)$ and for the second case $2*2n(v)+n(v) = 5n(v)$ cut linear edges. This factor is below the factors of inserted nodes, but it has to be underlined that a quadratic tetrahedron, which has the same number of degrees of freedom like the eight linear tetrahedra might have a higher accuracy.

6.4.2 Experimental Results

In order to confirm the theoretic results of the last subsection, we conducted the following example:

We cut a beam with 371 tetrahedral elements and 124 nodes as shown in Figure 6.20. The cut advances step by step, separating the beam in two pieces with threshold $\varepsilon = 1e{-}1$.

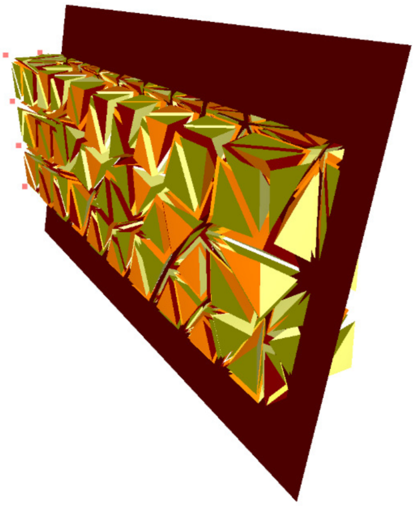


Fig. 6.20: Cut beam constructed with 371 tetrahedra. Figure extracted from [20].

The result is given in the following table:

Beam Elements	$\boldsymbol{n(v)}$	$\boldsymbol{2n(k_S) + n(k)}$
371	124	$2050 - 739 = 1311 \simeq 10.6n(v)$
712	215	$3033 - 1334 = 2199 \simeq 10.2n(v)$
4621	982	$16786 - 7334 = 9452 \simeq 9.6n(v)$

Tab. 6.5: Evaluation of beams with different resolutions.

For three different beam resolutions, table 6.5 displays the number of tetrahedral elements $n(v)$ intersecting with the cut and the number of added nodes using our remeshing algorithm to allow for a separation. For example, in the case of 371 beam elements, the number of nodes before the cut is 739 and 2050 after it. Then, the number of inserted nodes is 1311.

In this example, boundary elements are cut and the split 1 to 3 and the bisect edge add nodes for cut boundary elements. Those particularities are not considered in the theoretical analysis. Beams with a low resolutions have a higher ratio of cut boundary elements when compared to cut elements inside the volume. Thus, we expect more nodes added per cut element for cuts of low resolution beams. Table 6.5 confirms this expectation, but also reveals that the theoretical values are above our results in the experiment. This behaviour can be explained by a lower number of tetrahedra adjacent to an edge than in the theoretical analysis.

Results on a beam with different resolutions 6.5 show that while the number of nodes per element increases by a factor of $10/4 = 2.5$, the number of added nodes increases by a factor around 5. This is reasonable, due to the higher accuracy per degree of freedom of and the lower number of cut edges for quadratic elements.

6.5 Organ example: a liver

For the curved surfaces of organs or other structures like eye membrane, many linear tetrahedra need to be used to represent the smooth surface. In this particular context, cutting and cauterizing movement trajectories are smooth and thus difficult to represent by the straight lines and planar surfaces of the linear tetrahedra. In this section, we present the result of a liver represented by quadratic tetrahedra being remeshed and cut, as shown in Figure 6.21.

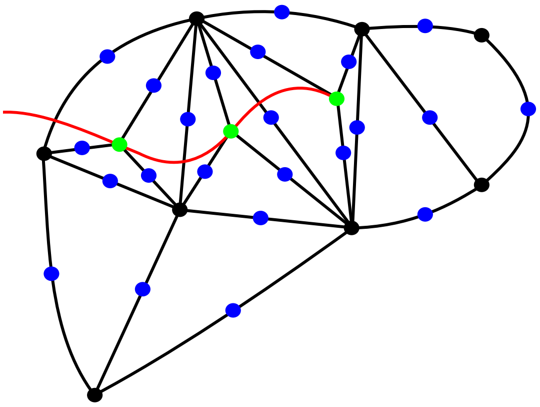


Fig. 6.21: Remeshing of a cut liver.

In what follows, we analyze the result of cutting a quadratic liver with a low resolution mesh created with the following parameters:

Solver	Iterations	Tolerance	Threshold	ν	E	Force
CGLinearSolver	1000	1e-5	1e-7	0.4	3000	(0, 0, 0)

About the cut:

Threshold Cut Node	Cut Left	Cut Right	Cut Direction	Boundary Condition
0.001	(15, -50, 0)	(15, 50, 0)	(0, 0, 5)	[0, 0, 0]×[10,30,30]

The process of cutting and remeshing of the liver configured as presented above can be seen in the following figures:

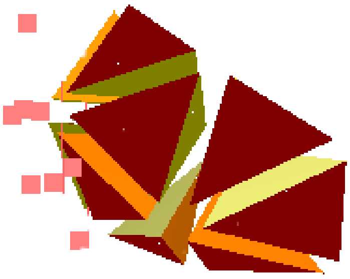


Fig. 6.22: Liver before the cut.

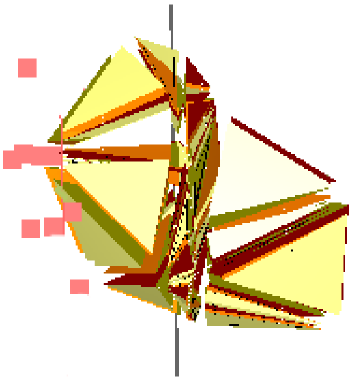


Fig. 6.23: Liver during the cut.

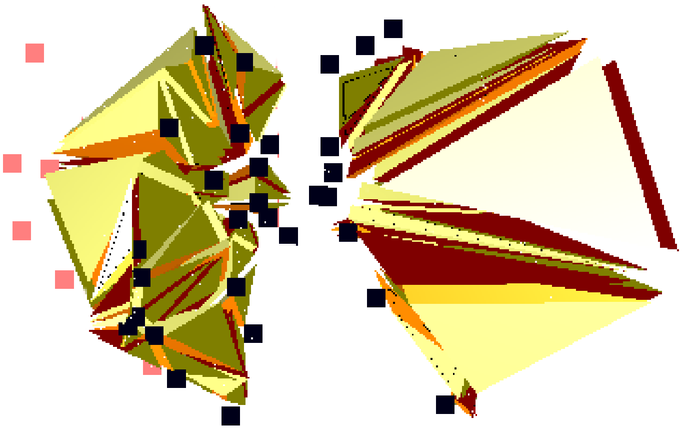


Fig. 6.24: Liver separated after the cut.

The pink dots represent the points of contact between the boundary conditions and the liver. The black dots represent the new created nodes.

We can observe that before the cut, the liver is made of 9 quadratic tetrahedra, during the cut it is remeshed and, after it, it is made of 34 elements.

Every split 1 to 4 inserts five new nodes: one on the cut and four on the edges, see table 6.3. Thus for the cut of the liver we have $n(v) = 45/9 = 5$ tetrahedral cut elements and we get for the number of added nodes $2n(k_S) + n(k) = (278 - 36)/5 = 48.4n(v)$. This is equivalent to a factor close to five, when comparing to theoretic results. Note that the presented case could be considered as worst case scenario: due to the low resolution most cut elements are boundary elements and since their surface is curved, the general flip can not get applied.

Every step adds the following number of nodes:

	Number of Nodes
Initial mesh	36
Split 1 to 4	81
Split 1 to 3	126
Flip 2 to 3	136
Bisect edge	198
Duplicate nodes	278

Tab. 6.6: Added nodes after the subsequent operations of the cut of a quadratic liver.

Then, the following result can be exposed:

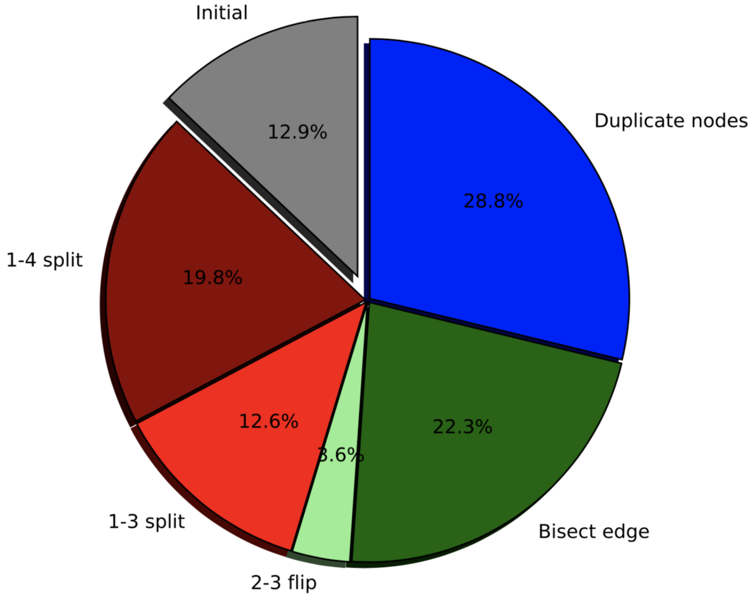


Fig. 6.25: Cut of quadratic liver, impact of the different topological operations on the number of nodes: pie chart. Figure extracted from [22].

Finally, we tested an example with more elements in order to observe its behavior. The configuration used in this case was the following one:

Number of elements	Solver	Iterations	Tolerance	Threshold	ν	E	Mass	Force
297	CGLinearSolver	1000	1e-5	1e-7	0.4	3000	1.0	(0, 0, 0)

In this case, the liver is not subjected to any force. The following figures present the result obtained:

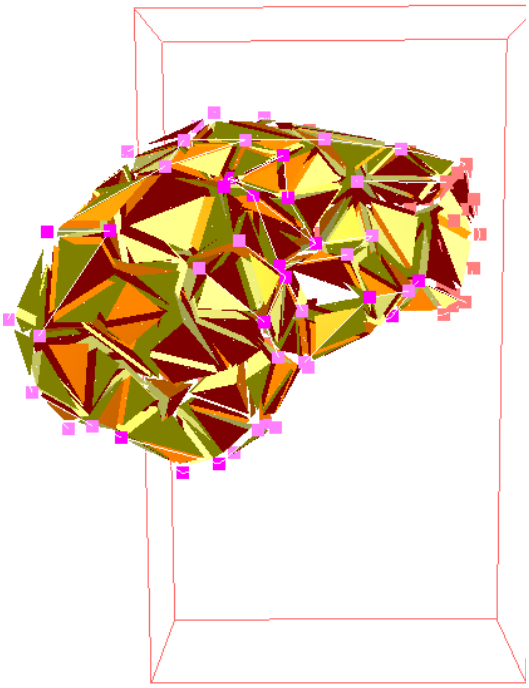


Fig. 6.26: Before cut.

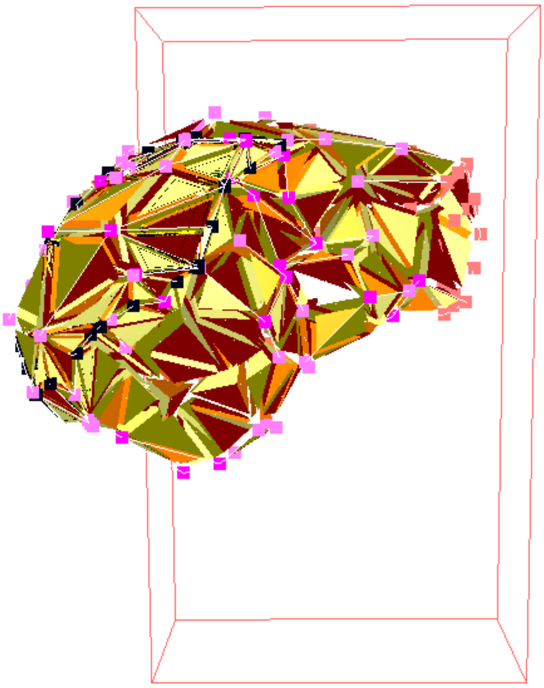


Fig. 6.27: After cut.

In this case, $n(v) = 124$ and the number of elements goes from 297 to 1787 after the cut. Then, $2n(k_s) + n(k) \simeq 12.0n(v)$, which is closer to the factor presented in 6.4.1 than the liver with few elements exhibited above. This confirms that by increasing the number of elements in the simulated object, the number of nodes inserted by the cut approaches the value exposed in the theoretical analysis of 6.4.1.

7. CONCLUSION

Interactive numerical simulations of surgical procedures, aimed at training, rehearsal or per-operative guidance are now considered important avenues to improve patient care and reduce risks [5]. Such interactions involve deformations, cutting, or tearing of the modeled soft tissues. Being able to handle all these interactions in real-time is of major importance and several conditions need to be met: the deformations should be computed accurately, computations must remain fast enough to allow interactivity, and topological changes need to cover a wide range of cases, yet not hinder computational efficiency.

A geometrical representation that uses linear tetrahedra is sufficient for applications in engineering, as the objects often have sharp corners. But for the curved surfaces of organs, many linear tetrahedra need to be used to represent the smooth surface. In this particular context, cut is smooth and difficult to represent by the straight lines and planar surfaces of the linear tetrahedra.

Suwelack et al [14] proposes to use quadratic tetrahedra as a robust and accurate model for real-time soft tissue simulations. The paper gives a detailed numerical analysis on the improved accuracy of quadratic shape functions and even shows an improved efficiency for the same accuracy.

The objetive of this thesis was to implement quadratic tetrahedra and to be able to cut objects represented by them in real-time in SOFA: a simulation framework dedicated to research in medical simulation 3. We mainly considered the deformation and cutting of deformable objects, based on elasticity theory and computed using the finite element method.

We began this thesis by introducing the theory of elasticity applied to our problem in 1 and the procedure of the finite element method followed to discretize it in 2. Then, in 3, we presented the structure of the SOFA framework and the CGoGN Plugin, where the implementation was included. After that, in 4, we detailed the implementation process applied, from 2-D in Python to 3-D in SOFA CGoGN, including the 3-D SOFA implementation without the CGoGN combinatorial map. Later, in 5, we explained the cutting process including the necessary methods to prepare the object to be cut (remeshing). Also, we described how the middle points were inserted in the introduced edges and we presented cutting results with an organ structure view, considering the remeshing algorithm expansion. Finally, we exhibited the results in 6, focusing the analysis in the quadratic tetrahedra implementation and in the remeshing with quadratic tetrahedra.

The results exposed in 6 allow us to conclude that quadratic tetrahedra have a better convergence 6.10, better shape approximation 6.14 and need less operations to converge 6.11 than linear elements, which confirms that quadratic elements are superior.

On the other hand, the theorical analysis 6.4.1 shows that in the case of quadratic tetrahedra, the remeshing inserts at least 5 times more nodes than in the linear tetrahedra and, for the same accuracy, the number of elements is greatly reduced when using quadratic instead of linear shape functions.

Finally, experimental results show that while the number of nodes per element increases by a factor of 2.5, the number of added nodes expands by a factor of around 5, which is reasonable due to the higher accuracy per degree of freedom and the lower number of cut edges for quadratic elements.

8. FUTURE WORK

The research presented in this thesis raised many questions. There are several lines of research arising from this work which should be pursued.

Among them, diverse studies could be done in order to compare more in detail the deformation differences between linear and quadratic tetrahedra by varying different parameters. In the case of the convergence analysis, for example, different Solvers (ODESolvers, MumpsSolver, EulerSolver, etc) could be evaluated in order to try to find distinct results. Besides, the number of iterations, the mass density and the tolerance of the solver could be varied.

In the context of the cut, inserting the middle point at the barycenter of the intersections between the cut surface S and the edges in the remeshing process leads to an inaccurate approximation of the cut. It is more natural, to set the middle point positions on the cut inside of the element [22], by averaging all the barycentric coordinates of the points and then inserting this averaged barycentric coordinate into the interpolation inside the element (an edge, a triangle or a tetrahedron). Even if this improvement only shows relevance for cuts that have strong changes in direction inside an element, it could be implemented in the future in order to have a more accurate cut simulation.

Finally, an important improvement would be to work with augmented-reality [18] while simulating cuts. Current methods dealing with non-rigid augmented reality only provide an augmented view when the topology of the tracked object is not modified, which is an important limitation. Paulus et al [19] introduces a method for physics-based non-rigid augmented reality, where singularities caused by topological changes are detected by analyzing the displacement field of the underlying deformable model. These topological changes are used to approximate the real cut and all its steps, from deformation to cutting simulation, and are performed in real-time. In the future, this could be included in the implementation to improve the coherence between the actual view and the model, and provide added value.

BIBLIOGRAPHY

[1] Lipshitz EM. Landau LD. *Theory of Elasticity.* Pergamon Press Ltd., 1970. Chap. 1.

[2] Johnston Jr. E.R. Beer F.P. *Mechanics of Materials.* McGraw-Hill, Inc., 1981. ISBN: 978-0073398235.

[3] K.J. Bathe. *Finite Element Procedures.* Prentice Hall, 1996.

[4] Leif Kobbelt. "&#8730;3subdivisionn". In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 103–112. ISBN: 1-58113-208-5. DOI: `10.1145/344779.344835`. URL: `http://dx.doi.org/10.1145/344779.344835`.

[5] L.T. Kohn et al. *Toerr is human: building a safer health system.* Vol. 627. National Academies Press, 2000.

[6] Young W. Kwon and Hyochoong Bang. *The Finite Element Method Using MATLAB.* 2nd. Boca Raton, FL, USA: CRC Press, Inc., 2000. ISBN: 0849309182.

[7] David V. Hutton. *Fundamentals of Finite Element Analysis.* McGraw-Hill, 2003.

[8] Jérémie Allard et al. "SOFA - an Open Source Framework for Medical Simulation". In: *MMVR 15 - Medicine Meets Virtual Reality.* Vol. 125. Studies in Health Technology and Informatics. Palm Beach, United States: IOP Press, Feb. 2007, pp. 13–18. URL: `https://hal.inria.fr/inria-00319416`.

[9] J. Fish and T. Belytschoko. *A First Course in Finite Elements.* John Wiley & Sons, Ltd, 2007. Chap. 3.

[10] Johannes Mezger et al. "Interactive Physically-Based Shape Editing". In: *ACM Solid and Physical Modeling Symposium (SPM).* 2008.

[11] Stephan Suwelack et al. "Quadratic Corotated Finite Elements for Real-Time Soft Tissue Registration". In: (2010).

[12] François Faure et al. "SOFA: A Multi-Model Framework for Interactive Physical Simulation". In: *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery.* Ed. by Yohan Payan. Vol. 11. Studies in Mechanobiology, Tissue Engineering and Biomaterials. Springer, June 2012, pp. 283–321. DOI: `10.1007/8415\_2012\_125`. URL: `https://hal.inria.fr/hal-00681539`.

[13] Christoph J. Paulus. "Simulation of any section of soft-tissues with the Extended Finite Element Method (German)". MA thesis. Karlsruhe Institute of Technology, 2012.

[14] Stefan Suwelack et al. "Quadratic Corotated Finite Elements for Real-Time Soft Tissue Registration". In: *Computational Biomechanics for Medicine.* Ed. by Poul M.F. Nielsen, Adam Wittek, and Karol Miller. New York, NY: Springer New York, 2012, pp. 39–50. ISBN: 978-1-4614-3172-5.

[15] Pierre Kraemer et al. "CGoGN: N-dimensional Meshes with Combinatorial Maps". In: *22nd International Meshing Roundtable*. Ed. by Josep Sarrate and Matthew Staten. Strasbourg, France: Springer International Publishing, 2013, pp. 485–503. DOI: `10.1007/978-3-319-02335-9\_27`. URL: `https://hal.archives-ouvertes.fr/hal-01162100`.

[16] Christoph J. Paulus et al. "Simulation of Complex Cuts in Soft Tissue with the Extended Finite Element Method (X-FEM)". In: *Preprint Series of the Engineering Mathematics and Computing Lab* 0.02 (2014). ISSN: 2191-0693. URL: `http://heiup.uni-heidelberg.de/journals/index.php/emcl-pp/article/view/17635`.

[17] *Introduction to Finite Element Methods [(ASEN 5007)]*. Department of Aerospace Engineering Sciences, University of Colorado at Boulder. 2015. URL: `http://www.colorado.edu/engineering/CAS/courses.d/IFEM.d/Home.html`.

[18] Christoph J. Paulus et al. "Augmented Reality during Cutting and Tearing of Deformable Objects". In: *The 14th IEEE International Symposium on Mixed and Augmented Reality*. Fukuoka, Japan, Sept. 2015, p. 6. URL: `https://hal.inria.fr/hal-01184495`.

[19] Christoph J. Paulus et al. "Surgical Augmented Reality with Topological Changes". In: *Medical Image Computing and Computer Assisted Interventions*. München, Germany, Oct. 2015. URL: `https://hal.inria.fr/hal-01184498`.

[20] Christoph J. Paulus et al. "Virtual Cutting of Deformable Objects based on Efficient Topological Operations". In: *Computer Graphics International*. Strasbourg, France, 2015. URL: `https://hal.archives-ouvertes.fr/hal-01208546`.

[21] Stefan Suwelack. *A short introduction to soft tissue simulation*. 2015.

[22] Christoph J. Paulus. "Topological Changes in Simulations of Deformable Objects". PhD thesis. University of Strasbourg, 2017.

[23] *Dissemination of IT for the Promotion of Materials Science Kernel Description*. `https://www.doitpoms.ac.uk/tlplib/fem/node.php`. University of Cambridge.

[24] *SOFA Documentation*. `www.sofa-framework.org`.

[25] *The Finite Element Method*. `https://www.comsol.com/multiphysics/finite-element-method`. Multiphysics Cyclopedia.