



---

# Development of an Automatic TET10 Meshing program for rotating components

---

*Author:*

Chhavi KUSUM

Immatrikulation Number:

10017164

*Supervisor:*

Nicolai FORSTHOFER

Deutsche Zentrum für

Luft-und Raumfahrt

*Examiner:*

Prof. Dr.-Ing. Udo

Nackenhorst,

Prof. Dr.-Ing. Dominik

Schillinger

*A thesis submitted in fulfillment of the requirements  
for the degree of  
Master of Science*

*in*

Computational Methods in Engineering

May 22, 2020

## Declaration of Authorship

I, Chhavi KUSUM, declare that this thesis titled, "Development of an Automatic TET10 Meshing program for rotating components" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly by me without any foreign help.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

---

Date: 22.05.2020

---

## *Abstract*

This thesis addresses mesh generation with the focus on automation. It targets the meshing of a complex turbine blade and disk assembly. Firstly, it evaluates element types, studying them mathematically and practically. Subsequently, the algorithms and methods used to generate such element types are discussed. Existing all-purpose mesh generation programs are also deliberated. They are tested to determine *SALOME* can be developed further to accomplish the task at hand. Based on this a strategy is evolved to develop the tool with *SALOME*. Finally, the tool is developed which generates the mesh alongside nodal groups/sets. This meshing tool is part of a bigger automated design tool, which aims to automate the whole process of designing a new component...

**Keywords:** automatic mesh generation, FEM, shape functions, higher order elements, tetrahedron generation algorithm, *SALOME*

Die vorliegende Arbeit befasst sich mit der Netzgenerierung, die Automatisiert ist. Es zielt auf die Vernetzung einer komplexen Turbinenschaufel- und Scheibenbaugruppe ab. Zunächst werden Elementtypen bewertet, die mathematisch und praktisch untersucht werden. Anschließend werden die Algorithmen und Methoden zur Erzeugung solcher Elementtypen besprochen. Bestehende Allzweckprogramme zur Netzgenerierung werden ebenfalls in Betracht gezogen. Sie werden getestet und festgestellt, dass *SALOME* weiterentwickelt werden kann, um die jeweilige Aufgabe zu erfüllen. Basierend darauf wird eine Strategie entwickelt, um das Tool mit *SALOME* zu entwickeln. Schließlich wird das Werkzeug entwickelt, das das Netz mit Knotengruppen erzeugt. Dieses Vernetzungswerkzeug ist Teil eines größeren automatisierten Entwurfswerkzeugs, mit dem der gesamte Prozess des Entwurfs einer neuen Komponente automatisiert werden soll...

**Stichwörter:** automatisierte Netzgenerierung, FEM, Ansatzfunktion, lineare Ansatzfunktionen, quadratische Ansatzfunktionen, Tetrahedron generieren algorithm, *SALOME*

## *Acknowledgements*

I would first like to thank my thesis advisor, Prof. Dr.-Ing. Udo Nackenhorst of the Institut für Baumechanik und Numerische Mechanik at Leibniz Universität Hannover. He was always available whenever I ran into a trouble spot or had a question about my research or writing.

I would also like to thank my supervisor, Mr. Nicolai Forsthofer, who was always patient, enthusiastic and supportive. The door to Mr. Forsthofer was always open to discuss or solve a problem. He was specially very patient with my limited German language skills.

I would also like to acknowledge Prof. Dr.-Ing. Dominik Schillinger of the Institut für Baumechanik und Numerische Mechanik at Leibniz Universität Hannover as the second examiner of this thesis, and I am gratefully indebted to his for his very valuable comments on this thesis.

Finally, I must express my very profound gratitude to my family for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you...

# Contents

|   |            |
|---|------------|
| <b>Declaration of Authorship</b>                                    | <b>i</b>   |
| <b>Abstract</b>   | <b>ii</b>  |
| <b>Acknowledgements</b>   | <b>iii</b> |
| <b>1 Introduction</b>   | <b>1</b>   |
| 1.1 Organization of the report . . . . .                            | 3          |
| <b>2 Theory of Finite Element Method</b>                            | <b>4</b>   |
| 2.1 FEM for Continuum Mechanics . . . . .                           | 4          |
| 2.1.1 Weak form formulation . . . . .                               | 7          |
| 2.1.2 Finite Element Formulation . . . . .                          | 7          |
| 2.1.3 2D Linear Triangle Elements . . . . .                         | 8          |
| Assembling the element terms . . . . .                              | 11         |
| 2.1.4 Discretization shapes: Triangular vs. quadrilateral . . . . . | 11         |
| 2.2 Higher Order Elements . . . . .                                 | 15         |
| <b>3 Finite Element Types</b>                                       | <b>20</b>  |
| 3.1 Classification of element types . . . . .                       | 21         |
| 3.1.1 Geometry and selection of element type . . . . .              | 25         |
| 3.2 Finite Element Method Solution Tool . . . . .                   | 25         |
| 3.3 Beam . . . . .  | 26         |

|          |   |           |
|----------|---|-----------|
| 3.3.1    | Bending Analysis . . . . .  | 26        |
|          | First-order Hexahedrons . . . . .                                 | 27        |
|          | First-order tetrahedrons . . . . .                                | 28        |
|          | Second-order Tetrahedrons . . . . .                               | 29        |
| 3.3.2    | Modal Analysis . . . . .  | 30        |
| 3.4      | Stress Concentration . . . . .                                    | 32        |
|          | 3.4.1 Tension . . . . .   | 32        |
|          | 3.4.2 Bending Moment . . . . .                                    | 35        |
| 3.5      | Conclusion . . . . .  | 37        |
| <b>4</b> | <b>Mesh Generation Methods</b>                                    | <b>39</b> |
|          | 4.1 Delaunay Triangulation Criterion . . . . .                    | 40        |
|          | 4.2 Algorithms . . . . .  | 41        |
|          | 4.3 Constrained Delaunay Triangulations . . . . .                 | 42        |
| <b>5</b> | <b>Automation of Mesh generators</b>                              | <b>44</b> |
|          | 5.1 Test and Evaluation . . . . .                                 | 45        |
|          | 5.1.1 Geometry for secondary assessment . . . . .                 | 49        |
|          | 5.1.2 Assessment procedure . . . . .                              | 50        |
|          | 5.1.3 Secondary assessment: Open-source Mesh generation . . . . . | 51        |
|          | Gmsh . . . . .  | 51        |
|          | SALOME . . . . .  | 52        |
|          | 5.1.4 Secondary assessment: Proprietary Mesh generation . . . . . | 53        |
|          | FEMAP . . . . .   | 54        |
|          | ENNOVA . . . . .  | 55        |
|          | 5.1.5 Assessment matrix . . . . .                                 | 56        |
|          | 5.2 Development of SALOME Interface . . . . .                     | 58        |

|          |  |           |
|----------|--|-----------|
| 5.2.1    | Introduction to SALOME . . . . .             | 58        |
| 5.2.2    | Meshing Algorithms and Hypotheses . . . . .  | 59        |
| 5.2.3    | Target Geometry . . . . .                    | 62        |
| 5.2.4    | Automation Process . . . . .                 | 65        |
| 5.2.5    | Interfaces . . . . .                         | 70        |
| 5.3      | Conclusion . . . . .                         | 71        |
| <b>6</b> | <b>Conclusion</b>                            | <b>72</b> |
|          | <b>Bibliography</b>                          | <b>73</b> |
| <b>A</b> | <b>Program for Automatic Meshing</b>         | <b>76</b> |
| <b>B</b> | <b>Format conversion of mesh export file</b> | <b>83</b> |
| <b>C</b> | <b>List of Mesh generation softwares</b>     | <b>87</b> |

# List of Figures

|      |   |    |
|------|---|----|
| 2.1  | A continuum domain, $\Omega_0$ in reference position and $\Omega_t$ in deformed position[14] . . . . .  | 5  |
| 2.2  | A domain discretized using triangles . . . . .  | 8  |
| 2.3  | Triangle element[14] . . . . .  | 9  |
| 2.4  | Shape functions at the nodes of a triangle[14] . . . . .  | 9  |
| 2.5  | Discretization of a domain using Triangles and Quadrilaterals[14] . . .   | 12 |
| 2.6  | Isoparametric mapping concept[14] . . . . .   | 12 |
| 2.7  | Isoparametric mapping and shape functions for triangles, tetrahedrons and hexahedrons[14] . . . . .   | 14 |
| 2.8  | Locked triangular elements[13] . . . . .  | 15 |
| 2.9  | Un-compatible elements (a) Discretization and load (b) Deformed Shape[28]   | 16 |
| 2.10 | (a) Deformation of cantilever beams (b) Rigid body displacement of grey element[28] . . . . .   | 16 |
| 2.11 | The 6-noded quadratic triangle element (a) straight edges and mid-side nodes at midpoints. (b) the isoparametric triangle quadratic element[29] . . . . . | 17 |
| 2.12 | The 6-noded quadratic triangle element coordinates[28] . . . . .  | 17 |
| 2.13 | Shape functions $N_1$ and $N_4$ for the quadratic triangle. The shape function is 1 at node 1 and 4 respectively, and 0 everywhere else.[28] . . . .      | 18 |
| 2.14 | A tetrahedron with quadratic shape functions[28] . . . . .  | 18 |
| 3.1  | Structured and Unstructured meshes[12] . . . . .  | 21 |
| 3.2  | Conformal and Non-conformal meshes[12] . . . . .  | 22 |



|      |  |    |
|------|--|----|
| 3.3  | Triangle and Quadrilateral elements[12]  | 22 |
| 3.4  | Three-dimensional element shapes[12]   | 22 |
| 3.5  | Linear and Quadratic elements  | 23 |
| 3.6  | A Quadrilateral element with 9 nodes   | 24 |
| 3.7  | FEM Analysis results comparison using linear and quadratic elements                            | 24 |
| 3.8  | A High Pressure turbine blade from a GE jet engine[31]   | 25 |
| 3.9  | Simply supported Cantilever Beam with point load   | 26 |
| 3.10 | Boundary conditions for simply supported beam in numerical model                               | 27 |
| 3.11 | Cantilever beam meshed with first-order hexahedrons: 10 elements in depth                      | 27 |
| 3.12 | Beam tip deflection with first-order hexahedrons   | 28 |
| 3.13 | Cantilever beam meshed with first-order tetrahedrons: 10 nodes in depth                        | 28 |
| 3.14 | Beam tip deflection with first-order tetrahedrons  | 29 |
| 3.15 | Beam tip deflection with quadratic tetrahedrons  | 29 |
| 3.16 | Beam tip deflection: Analytical and Numerical results  | 30 |
| 3.17 | Beam for frequency analysis  | 30 |
| 3.18 | Effect of element type in a free-free modal analysis: Mode 1                                   | 31 |
| 3.19 | Effect of element type in a free-free modal analysis: Mode 2                                   | 31 |
| 3.20 | Flow of stress is denser near the hole   | 32 |
| 3.21 | A finite plate with a hole   | 33 |
| 3.22 | Closed-form Stress Concentration [9]   | 33 |
| 3.23 | Boundary conditions for a plate with a hole to study stress concentrations under tensile loads | 34 |
| 3.24 | Tension loading: Stress concentration plots  | 34 |
| 3.25 | A plate with hole subjected to out-of-plane bending moment                                     | 35 |
| 3.26 | Closed form Stress concentration - Plate with a hole   | 36 |

|      |  |    |
|------|--|----|
| 3.27 | Boundary conditions for a plate with a hole to study stress concentrations under bending moment . . . . .  | 36 |
| 3.28 | Bending load: Stress concentrations . . . . .  | 37 |
| 4.1  | Advancing Front Method for mesh generation [16] . . . . .  | 39 |
| 4.2  | A graphical representation of the Quadtree Algorithm . . . . .   | 40 |
| 4.3  | Delaunay triangulation in two-dimensions maintained in (a), not maintained in (b) . . . . .  | 41 |
| 4.4  | (a) creates all the delaunay simplices between the specified points, (b)chooses only the triangles and (c) excludes all the crossing Delaunay edges [26] . . . . . | 41 |
| 5.1  | Automation of the structural mechanical design process for a blade assembly[4] . . . . .   | 46 |
| 5.2  | Test Geometry used to assess the mesh generators[18] . . . . .   | 49 |
| 5.3  | Small details of the turbine blade[18] . . . . .   | 50 |
| 5.4  | Steps followed to perform the secondary assessment for mesh generators . . . . .   | 51 |
| 5.5  | Test Geometry mesh using <b>Gmsh</b> , poor aspect ratio and poor representation of features . . . . .   | 52 |
| 5.6  | Test Geometry mesh using <b>SALOME</b> , stable program with a mesh that captures the details accurately . . . . .   | 53 |
| 5.7  | Test Geometry mesh using Siemens FEMAP, similar details as SALOME but lower node count . . . . .   | 54 |
| 5.8  | Test Geometry mesh using <b>ENNOVA</b> , captures the details properly . . . . .   | 55 |
| 5.9  | NETGEN 2D Hypothesis . . . . .   | 61 |
| 5.10 | NETGEN 3D Hypothesis . . . . .   | 62 |
| 5.11 | A turbine blade with a complex cooling circuit[8] . . . . .  | 62 |
| 5.12 | A typical blade and disk assembly[3] . . . . .   | 63 |
| 5.13 | Mesh requirement on fillets . . . . .  | 64 |

|      |   |    |
|------|---|----|
| 5.14 | Mesh requirement on very small curves . . . . .   | 64 |
| 5.15 | Mesh requirement on cooling air holes . . . . .   | 64 |
| 5.16 | Flowchart of a typical mesh creation job using an imported geometry<br>file . . . . .             | 65 |
| 5.17 | Distribution of HPT blade faces' area . . . . .   | 67 |
| 5.18 | Flowchart of the processes in the geometry module on the HPT blade<br>and disk assembly . . . . . | 67 |
| 5.19 | Flowchart of the processes in the meshing module on the HPT blade<br>and disk assembly . . . . .  | 68 |
| 5.20 | Blade Mesh . . . . .  | 69 |
| 5.21 | Blade Mesh of the internal cooling pipes . . . . .  | 69 |
| 5.22 | Aspect ratio errors in the blade mesh . . . . .   | 70 |
| 5.23 | Flowchart to convert a UNV to a DAT file . . . . .  | 71 |

# List of Tables

|     |  |    |
|-----|--|----|
| 2.1 | A comparison between triangle and quadrilateral elements . . . . .   | 14 |
| 2.2 | A comparison between linear and quadratic triangle/tetrahedron elements . . . . .  | 19 |
| 3.1 | Free Modal analysis: Summary of analytical and FE model results . . .  | 31 |
| 3.2 | Stress concentration on a Plate with a hole subjected to tensile loading: Summary of closed-form and FE model results . . . . .        | 35 |
| 3.3 | Stress concentration on a Plate with a hole subjected to <b>bending moment</b> : Summary of closed-form and FE model results . . . . . | 37 |
| 5.1 | Assessment Matrix of Open-source and Proprietary meshing Programs(1-best score & 4-worst score) . . . . .                              | 57 |

# List of Abbreviations

|            |   |
|------------|---|
| <b>FEM</b> | <b>Finite Element Method</b>              |
| <b>FEA</b> | <b>Finite Element Analysis</b>            |
| <b>PDE</b> | <b>Partial Differential Equation</b>      |
| <b>CDT</b> | <b>Constrained Delaunay Triangulation</b> |
| <b>CAD</b> | <b>Computer Aided Definition</b>          |
| <b>DOF</b> | <b>Degree Of Freedom</b>                  |
| <b>API</b> | <b>Application Programming Interface</b>  |
| <b>HPT</b> | <b>High Pressure Turbine</b>              |

## Chapter 1

# Introduction

One of the most used tools for scientific computing is the tool, Finite Element Method (FEM). FEM is a numerical method to solve any physical problem. Most of the physical problems can be represented using Partial Differential Equations (PDEs). FEM uses numerical computation to solve the PDEs.

But, before the PDEs can be solved, the continuous CAD geometry has to be broken into small and simple pieces or *elements*. An element, for e.g. a Triangle, is easier to compute as it is well known how to perform calculations on such geometries. Each element has a number of nodes associated to it, based on the type of the element. For the most basic triangle element, it is 3 nodes. This discretization is termed as mesh generation and is a necessary tool in the FEM world. Construction of a mesh is considered frequently as a bottleneck in the whole process. It is possible that it takes longer to create a mesh than to actually solve the PDEs.

The automatic mesh generation problem is to divide complex geometries in physical domains into finite parts without any user intervention. Generating a mesh can be challenging to create as it is hard to satisfy all the requirements at the same time, which are contradictory in nature. The mesh must represent the geometry accurately enough. It must also ensure that the governing PDEs are approximated accurately. This is ensured via a set of polynomial functions defined over each element. As the elements become smaller, the numerical solution will converge to the true solution.

This must be ensured while maintaining the element size along with the shape of the element. The FEM simulation converges to the solution of the actual boundary value problem as the element size becomes smaller and smaller. In complex geometries, multitude of elements would be needed to achieve these requirements. Contradictory to this, the number of elements has to be confined to a reasonable number to ensure a certain size of the simulation model. The number of elements directly affects the simulation model size. The simulation model size influences the

speed and the time required to compute the solution to the PDEs or the computational cost. Thus, it is an important factor in generating a mesh.

Once the model requirements are set, next step is to determine the type of the element to be used. Elements can be categorized in many ways. For e.g. according to its shape or its dimension. Higher quality or right shaped elements generally have better numerical properties. A right shaped element's definition changes with the problem and its governing equations. An example of a right shaped element could be a triangle that is equilateral and equiangular.

Based on basic shapes, the most common surface or two-dimensional elements are Triangles and Quadrilaterals. Poor elements in this category can be defined by sharp angles or short edges. Tetrahedrons and Hexahedrons among other three-dimensional elements are the most popular elements in usage. These elements are made with all Triangles or all Quadrilaterals or simply a combination of both. Meshes can also be classified as structured or unstructured. A structured mesh is so defined that basic arithmetic can be used to determine which elements surround a vertex. On the other hand, unstructured meshes entail storing information about each vertices' elements and neighbouring vertices.

In the last few decades, the usage of meshes has also grown outside of the finite element world. The animation industry uses polygon meshes extensively. Economically, the computer animation industry supersedes the finite-element industry in their use of meshes. There are other industries that use meshes actively, such as image processing, population sampling, aerial land surveying etc. Mesh generation is multi-disciplinary now.[26]

Generating meshes can be a cumbersome and time-consuming process, based on the geometry on hand. It affects the results directly but is not a value adding step to the whole FEM process. Thus, depending on the design process involved, it can be quite efficient to setup an automated meshing process to allow the user to proceed directly from the CAD step to the finite element solution.

The mesh generation industry is rife with mesh generation solutions/programs. They can be of two kinds. A program which is fully automatic i.e. no user interaction is required to generate the mesh. The mesh generation process acts as a black box. Only the geometry has to be input and the mesh will be generated. Automatic mesh generation is already being established for specific problems. But, for arbitrary geometries, completely automatic mesh generation, which meets all the requirements, is still an elusive goal.

The other kind of mesh generation programs are those that generate the mesh based on the boundary conditions set by the user. The user has to have some basic understanding of the program to generate a mesh that satisfies all the requirements.

Such solutions are aplenty and work well with any arbitrary geometry.

The thesis on hand describes the development of a completely automated meshing program, which is restricted to a specific problem. It starts with setting the groundwork to determine the preferred element type. This is achieved using FE analyses of simple and known problems. After the determination, the methods and algorithms present to generate this element type are studied. Each of the methods are evaluated for their advantages and disadvantages. Based of these, existing mesh generation programs are studied and evaluated based on a CAD model of turbine blades. They are assessed based on several criteria. One of them is chosen and the automated meshing tool is developed based on it. This tool is a part of a bigger automated analysis package. It is developed with the ability to mesh surfaces based on their sizes without user intervention. It takes the CAD geometry and meshes it automatically. Further, the tool allows to proceed directly to the finite element solution step by making it ready to be analyzed.

## 1.1 Organization of the report

This thesis report is divided into four chapters. First, in *chapter 2* the finite element method is described. A general boundary value problem is introduced, followed by its weak formulation. Afterwards, the actual FE method is explained with focus on the shape functions of elements. Triangles (tetrahedrons in 3D) and Quadrilateral (Hexahedrons in 3D) are compared and their formulations are studied.

To help the reader familiarize oneself with the various element types, *chapter 3* categorizes the element types. Then, it describes a study done to compare different element types, namely first-order hexahedrons, first-order and second-order tetrahedrons. The elements' behaviours are investigated for cases whose analytical solutions or closed form solutions are already present. The chapter concludes on second-order tetrahedrons being the best choice.

*Chapter 4* follows with the techniques and algorithms used to generate tetrahedral elements. It begins with the Delaunay Triangulation Criterion, one of the most popular triangle and tetrahedron meshing techniques. Afterwards, the current algorithms used to apply the Delaunay Triangulation criterion are discussed. Finally, it touches upon Constrained Delaunay Triangulation, a relaxed Delaunay Triangulation criterion.

*Chapter 5* presents the program developed in the course of this work. It starts with the assessment of existing meshing programs. The determination of criterion defining mesh quality generated by these programs follow next. Finally, the automatic meshing program is developed using Python and an open-source platform for numerical simulation, SALOME.



## Chapter 2

# Theory of Finite Element Method

Laws of Physics can express problems as partial differential equations. Majority of these problems can't be solved analytically. Hence, an approximation of the equations are established. The PDEs are approximated, using discretization, into algebraic equations which can be solved using numerical methods. The solution to these equations is an approximation of the real solution of the PDEs. The *finite element method* (FEM) is used to compute this approximation. Typical steps involved in obtaining a solution through FEM involve [15]:

- Creating a mechanical model
- Creating a mathematical model
- Discretization of the domain: subdivision into finite elements
- Building a finite element formulation
- Solving the associated algebraic system
- Post-processing the result

### 2.1 FEM for Continuum Mechanics

The continuum mechanics approach ignores the discrete nature of matter, considers uniform material which is uniformly distributed in space. Hence, material properties are defined as continuous functions of position. For this approach, the underlying fundamental equations are introduced and the weak formulation is obtained. Consider the domain  $\Omega_0$  in Figure 2.1. The deformed domain of this domain is  $\Omega_t$ .

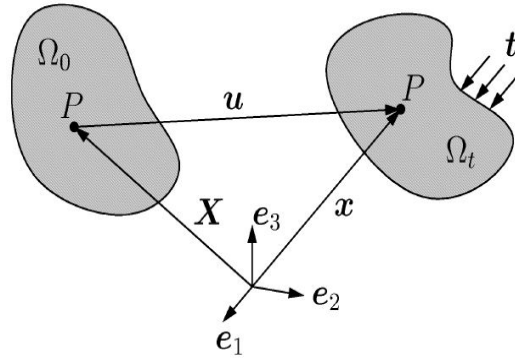


FIGURE 2.1: A continuum domain,  $\Omega_0$  in reference position and  $\Omega_t$  in deformed position[14]

### Kinematics

The displacement can be symbolically denoted as:

$$\mathbf{u} = \mathbf{x} - \mathbf{X} \quad (2.1)$$

Its components are:

$$u_i = x_i - X_i \quad (2.2)$$

Similarly, strain can be written symbolically and in its components as:

$$\boldsymbol{\varepsilon} = \frac{1}{2} \left( \text{grad}(\mathbf{u}) + \text{grad}^T(\mathbf{u}) \right) \quad (2.3)$$

$$\varepsilon_{ij} = \frac{1}{2} (u_{i,j} + u_{j,i}) \quad (2.4)$$

$$(2.5)$$

The components for strain can be written in three-dimensions as:

$$\boldsymbol{\varepsilon}(\mathbf{u}) = \begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ 2\varepsilon_{12} \\ 2\varepsilon_{23} \\ 2\varepsilon_{31} \end{Bmatrix} = \begin{Bmatrix} u_{1,1} \\ u_{2,2} \\ u_{3,3} \\ u_{1,2} + u_{2,1} \\ u_{2,3} + u_{3,2} \\ u_{3,1} + u_{1,3} \end{Bmatrix} \quad (2.6)$$

### Equilibrium

The balance of momentum in 3D can be written as:

$$\int_{\Omega_t} \rho \mathbf{b} \, dv + \int_{\partial\Omega_t} \mathbf{t} \, da = \mathbf{0} \quad (2.7)$$

$\mathbf{b}$  - Volume/acceleration body force vector

$\mathbf{t}$  - surface traction vector

Using the Cauchy theorem,

$$\mathbf{t} = \boldsymbol{\sigma} \cdot \mathbf{n} \quad (2.8)$$

$\boldsymbol{\sigma}$  - Cauchy stress tensor

$\mathbf{n}$  - normal vector

Equation 2.7 can be rewritten as:

$$\int_{\Omega_t} \rho \mathbf{b} \, dv + \int_{\partial\Omega_t} \boldsymbol{\sigma} \cdot \mathbf{n} \, da = \mathbf{0} \quad (2.9)$$

Using the Gauss theorem and since  $\Omega_t$  can be chosen arbitrarily:

$$\begin{aligned} \int_{\Omega_t} [\rho \mathbf{b} + \text{div}(\boldsymbol{\sigma})] \, dv &= \mathbf{0} \\ \Rightarrow \text{div}(\boldsymbol{\sigma}) + \rho \mathbf{b} &= \mathbf{0} \end{aligned} \quad (2.10)$$

Equation 2.10 can be written in components for three-dimensions in voigt notation as:

$$\sigma_{ij,j} + \rho b_i = 0_i \quad (2.11)$$

$$\boldsymbol{\sigma}(\mathbf{u}) = \begin{Bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{23} \\ \sigma_{31} \end{Bmatrix} \quad (2.12)$$

### Constitutive relation

Hooke's law

$$\boldsymbol{\sigma}(\mathbf{u}) = 2\mu \boldsymbol{\varepsilon} + \Lambda \text{tr}(\boldsymbol{\varepsilon}) \mathbf{1} \quad (2.13)$$

$$\Lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \quad (2.14)$$

$$\mu = \frac{E}{2(1+\nu)} \quad (2.15)$$

This stress-strain can be written in matrix notation in three-dimensions in voigt notation:

$$\begin{Bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{23} \\ \sigma_{31} \end{Bmatrix} = \underbrace{\begin{Bmatrix} 2\mu + \Lambda & \Lambda & \Lambda & 0 & 0 & 0 \\ \Lambda & 2\mu + \Lambda & \Lambda & 0 & 0 & 0 \\ \Lambda & \Lambda & 2\mu + \Lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{Bmatrix}}_{\mathcal{C}_{3D}} \cdot \begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ 2\varepsilon_{12} \\ 2\varepsilon_{23} \\ 2\varepsilon_{31} \end{Bmatrix} \quad (2.16)$$

Now, the field equations for a 3D continuum are established, the weak form of the equilibrium would be derived.

### 2.1.1 Weak form formulation

Weak form of equilibrium is also referred as the principle of virtual work and can be derived from the equilibrium conditions. The Differential equation depicting the continuum is multiplied with a test function,  $\eta$ . The test function can be chosen arbitrarily. It is then integrated over the domain and then finally, half of the derivatives are transferred to the test function, reducing the order of the differential equation.

Equation 2.10 when multiplied with the test function,

$$\int_{\Omega_t} (div(\boldsymbol{\sigma}) + \rho \mathbf{b}) \cdot \boldsymbol{\eta} dv = 0 \quad (2.17)$$

As  $div(\boldsymbol{\eta}^T \boldsymbol{\sigma}) = div(\boldsymbol{\sigma}) \cdot \boldsymbol{\eta} + \boldsymbol{\sigma} : grad(\boldsymbol{\eta})$

$$\begin{aligned} \int_{\Omega_t} (div(\boldsymbol{\eta}^T \cdot \boldsymbol{\sigma}) - \boldsymbol{\sigma} : grad(\boldsymbol{\eta}) + \rho \mathbf{b} \cdot \boldsymbol{\eta}) dv &= 0 \\ \int_{\Omega_t} \boldsymbol{\sigma} : grad(\boldsymbol{\eta}) dv - \int_{\Omega_t} \rho \mathbf{b} \cdot \boldsymbol{\eta} dv - \int_{\partial\Omega_t} \mathbf{t} \cdot \boldsymbol{\eta} da &= 0 \end{aligned} \quad (2.18)$$

Since the stress tensor is symmetric,  $\boldsymbol{\sigma} = \boldsymbol{\sigma}^T$ .

And,  $\boldsymbol{\varepsilon}(\boldsymbol{\eta}) = \frac{1}{2}(grad(\boldsymbol{\eta}) + grad^T(\boldsymbol{\eta}))$ .

Hence,  $\boldsymbol{\sigma} : grad(\boldsymbol{\eta}) = \boldsymbol{\sigma} : grad^T(\boldsymbol{\eta}) = \boldsymbol{\sigma} : \boldsymbol{\varepsilon}(\boldsymbol{\eta})$

$$\int_{\Omega_t} \boldsymbol{\sigma} : \boldsymbol{\varepsilon}(\boldsymbol{\eta}) dv - \int_{\Omega_t} \rho \mathbf{b} \cdot \boldsymbol{\eta} dv - \int_{\partial\Omega_t} \mathbf{t} \cdot \boldsymbol{\eta} da = 0 \quad (2.19)$$

Equation 2.19 is the weak form of equilibrium.

### 2.1.2 Finite Element Formulation

The continuum domain in figure 2.1 is discretized into finite elements. The elements have nodes at their vertices. The primary variable is now approximated at the nodes. To transfer the primary variable to the elements, shape functions are used. The continuum domain,  $\Omega$  is divided into  $n_e$  elements. Hence, a discretization error occurs here and  $\Omega_h$  is the approximation of  $\Omega$ .

$$\Omega_h = \bigcup_{e=1}^{n_e} \Omega_e \neq \Omega \quad (2.20)$$

Discretization can be carried out using triangles/tetrahedrons or quadrilaterals/hexahedrons for 2D/3D domains. An example of discretization using triangles can be seen in figure 2.2.

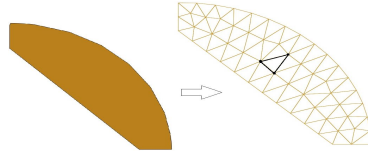


FIGURE 2.2: A domain discretized using triangles

The element quantities, like the area, integrals, derivatives are computed once and then assembled in the global system.

### 2.1.3 2D Linear Triangle Elements

The primary variable is approximated in the domain and then represented in one plane using three constants, figure 2.3 and equation 2.21.

$$u_x(x, y) = c_1 + c_2x + c_3y \quad (2.21)$$

$$= (1 \quad x \quad y) \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \quad (2.22)$$

$$u_x(x_1, y_1) = u_{x1} \quad (2.23)$$

$$u_x(x_2, y_2) = u_{x2} \quad (2.24)$$

$$u_x(x_3, y_3) = u_{x3} \quad (2.25)$$

$$\Rightarrow \begin{pmatrix} u_{x1} \\ u_{x2} \\ u_{x3} \end{pmatrix} = \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \quad (2.26)$$

$$\Rightarrow \mathbf{u}_x = \mathbf{H} \mathbf{c} \quad (2.27)$$

$$u_x(x, y) = (1 \quad x \quad y) \mathbf{c} \quad (2.28)$$

$$= \underbrace{(1 \quad x \quad y)}_{\text{Shape functions}} \mathbf{H}^{-1} \mathbf{u}_x \quad (2.29)$$

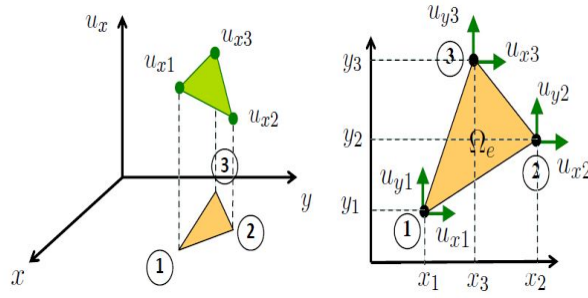


FIGURE 2.3: Triangle element[14]

At this point, shape functions are introduced. The shape functions are so defined that they are 1 at a given node and 0 elsewhere.

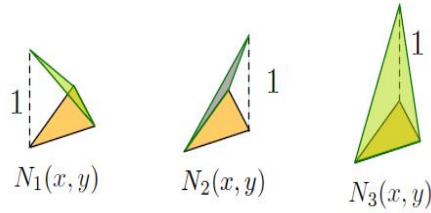


FIGURE 2.4: Shape functions at the nodes of a triangle[14]

$$\mathbf{u}_x(x, y) = \begin{pmatrix} N_1(x, y) & N_2(x, y) & N_3(x, y) \end{pmatrix} \mathbf{u}_x \quad (2.30)$$

The shape functions can be written in general terms as:

$$N_i(x, y) = \frac{1}{2\Omega_e} (f_i + g_i x + h_i y) \quad (2.31)$$

with:

$$\begin{aligned} f_1 &= x_2 y_3 - x_3 y_2 & g_1 &= y_2 - y_3 & h_1 &= x_3 - x_2 \\ f_2 &= x_3 y_1 - x_1 y_3 & g_2 &= y_3 - y_1 & h_2 &= x_1 - x_3 \\ f_3 &= x_1 y_2 - x_2 y_1 & g_3 &= y_1 - y_2 & h_3 &= x_2 - x_1 \end{aligned} \quad (2.32)$$

The area of the element,  $\Omega_e$  in figure 2.3 can be given by the expression,

$$\Omega_e = \frac{x_1 g_1 + x_2 g_2 + x_3 g_3}{2} \quad (2.33)$$

The primary variable can now be written as an approximation in terms of the shape functions. The same approximation function is chosen for the test function.

$$\mathbf{u}_h(x, y) = \begin{Bmatrix} u_{hx}(x, y) \\ u_{hy}(x, y) \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^3 N_i(x, y) u_{xi} \\ \sum_{i=1}^3 N_i(x, y) u_{yi} \end{Bmatrix} \quad (2.34)$$

$$\boldsymbol{\eta}(x, y) = \begin{Bmatrix} \eta_x(x, y) \\ \eta_y(x, y) \end{Bmatrix} = \begin{Bmatrix} \sum_{k=1}^3 N_i(x, y) \eta_{xk} \\ \sum_{k=1}^3 N_i(x, y) \eta_{yk} \end{Bmatrix} \quad (2.35)$$

In the weak form of equilibrium in figure 2.19, the first derivatives of the displacement field ( $\boldsymbol{\varepsilon}(\boldsymbol{\eta})$ ) or the primary variable are obtained. Hence, the derivatives of the shape functions will be calculated and are:

$$\begin{aligned} \frac{\partial N_i}{\partial x} &= \frac{1}{2\Omega_e} g_i \\ \frac{\partial N_i}{\partial y} &= \frac{1}{2\Omega_e} h_i \end{aligned} \quad (2.36)$$

Inserting the approximation for  $\boldsymbol{\eta}(x, y)$  into  $\boldsymbol{\varepsilon}(\boldsymbol{\eta})$ :

$$\begin{aligned} \boldsymbol{\varepsilon}(\boldsymbol{\eta}) &= \begin{Bmatrix} \frac{\partial \eta_x(x, y)}{\partial x} \\ \frac{\partial \eta_y(x, y)}{\partial y} \\ \frac{\partial \eta_x(x, y)}{\partial y} + \frac{\partial \eta_y(x, y)}{\partial x} \end{Bmatrix} \\ &= \begin{Bmatrix} \sum_{k=1}^3 \frac{\partial N_i}{\partial x} \eta_{xk} \\ \sum_{k=1}^3 \frac{\partial N_i}{\partial y} \eta_{yk} \\ \sum_{k=1}^3 \frac{\partial N_i}{\partial y} \eta_{xk} + \sum_{k=1}^3 \frac{\partial N_i}{\partial x} \eta_{yk} \end{Bmatrix} \\ &= \begin{Bmatrix} \sum_{k=1}^3 \frac{1}{2\Omega_e} g_k \eta_{xk} \\ \sum_{k=1}^3 \frac{1}{2\Omega_e} h_k \eta_{yk} \\ \sum_{k=1}^3 \frac{1}{2\Omega_e} (h_k \eta_{xk} + g_k \eta_{yk}) \end{Bmatrix} \\ \Rightarrow \boldsymbol{\varepsilon}(\boldsymbol{\eta}) &= \sum_{k=1}^3 \frac{1}{2\Omega_e} \underbrace{\begin{Bmatrix} g_k & 0 \\ 0 & h_k \\ h_k & g_k \end{Bmatrix}}_{\mathbf{B}_k} \cdot \underbrace{\begin{Bmatrix} \eta_{xk} \\ \eta_{yk} \end{Bmatrix}}_{\boldsymbol{\eta}_k} \end{aligned} \quad (2.37)$$

Similarly, the term  $\boldsymbol{\varepsilon}(\mathbf{u})$  can be written,

$$\boldsymbol{\varepsilon}(\mathbf{u}) = \sum_{i=1}^3 \frac{1}{2\Omega_e} \underbrace{\begin{Bmatrix} g_i & 0 \\ 0 & h_i \\ h_i & g_i \end{Bmatrix}}_{\mathbf{B}_i} \cdot \underbrace{\begin{Bmatrix} u_{xi} \\ u_{yi} \end{Bmatrix}}_{\mathbf{u}_i} \quad (2.38)$$

Inserting these discrete ansatz in the weak form of equilibrium and using equation 2.34:

$$\int_{\Omega_t} \boldsymbol{\varepsilon}(\boldsymbol{\eta}) : \boldsymbol{\sigma}(\mathbf{u}_h) dv = \int_{\Omega_t} \sum_{k=1}^3 \sum_{i=1}^3 \frac{1}{4\Omega_e^2} \boldsymbol{\eta}_k^T \cdot \mathbf{B}_k^T \cdot \mathbf{C}_{2D} \cdot \mathbf{B}_i \cdot \mathbf{u}_i dv \quad (2.39)$$

$$\int_{\Omega_t} \boldsymbol{\eta} \cdot \rho \mathbf{b} dv + \int_{\partial\Omega_t} \boldsymbol{\eta} \cdot \mathbf{t} da = \int_{\Omega_t} \sum_{k=1}^3 \boldsymbol{\eta}_k^T N_k \cdot \rho \mathbf{b} dv + \int_{\partial\Omega_t} \sum_{k=1}^3 \boldsymbol{\eta}_k^T N_k \cdot \mathbf{t} da \quad (2.40)$$

### Assembling the element terms

As the test function can be arbitrarily chosen,  $\boldsymbol{\eta}_k$  can also be arbitrarily chosen. Thus,

$$\underbrace{\sum_{k=1}^3 \sum_{i=1}^3 \int_{\Omega_t} \frac{1}{4\Omega_e^2} \cdot \mathbf{B}_k^T \cdot \mathbf{C}_{2D} \cdot \mathbf{B}_i dv \cdot \mathbf{u}_i}_{\mathbf{K}_{ki}^e} = \int_{\Omega_t} \sum_{k=1}^3 N_k \cdot \rho \mathbf{b} dv + \int_{\partial\Omega_t} \sum_{k=1}^3 N_k \cdot \mathbf{t} da \quad (2.41)$$

For each element 'e', it can be written in general terms in equation 2.42. For the element depicted in figure 2.3, the equation can be written in its components.

$$\mathbf{K}_e \mathbf{u}_e = \mathbf{f}_e \quad (2.42)$$

$$\begin{bmatrix} \mathbf{k}_{11}^e & \mathbf{k}_{12}^e & \mathbf{k}_{13}^e \\ \mathbf{k}_{21}^e & \mathbf{k}_{22}^e & \mathbf{k}_{23}^e \\ \mathbf{k}_{31}^e & \mathbf{k}_{32}^e & \mathbf{k}_{33}^e \end{bmatrix} \cdot \begin{bmatrix} u_{x1} \\ u_{y1} \\ u_{x2} \\ u_{y2} \\ u_{x3} \\ u_{y3} \end{bmatrix} = \begin{bmatrix} f_{x1} \\ f_{y1} \\ f_{x2} \\ f_{y2} \\ f_{x3} \\ f_{y3} \end{bmatrix} \quad (2.43)$$

The local stiffness matrix for an element can be now used to assemble the global stiffness matrix. Then, it will be used to build the global system of equations.

$$\mathbf{K} = \mathbf{A}_{e=1}^{n_{el}} \mathbf{K}_e \quad (2.44)$$

### 2.1.4 Discretization shapes: Triangular vs. quadrilateral

Along with triangular elements, discretization can also be created using quadrilateral elements. A comparison of discretization using triangles and quadrilaterals of the same domain can be seen in figure 2.5.



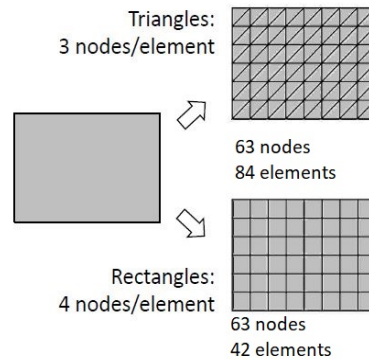


FIGURE 2.5: Discretization of a domain using Triangles and Quadrilaterals[14]

The number of nodes are same in both the meshes but the number of elements is much higher for triangular elements as compared to quadrilateral elements.

$$\mathbf{u} = \underbrace{\mathbf{K}^{-1}}_{63 \times 63 \text{ matrix}} \mathbf{f} \quad (2.45)$$

$$\mathbf{K} = \mathbf{A}_{e=1}^{n_{el}} \mathbf{K}_e \quad (2.46)$$

Hence, the element stiffness matrix are bigger for triangular elements than quadrilateral elements. At the first glance, this can mean that triangles are computationally costlier than quadrilaterals. The disadvantage of quadrilaterals is that they are not flexible enough to easily generate meshes for arbitrary geometries.

The shape functions of a quadratic element are defined with the idea of *isoparametric mapping*, figure 2.6.

1. Definition of a **parent** element. This is a reference square element and the shape functions are defined on it.
2. The parent element and the real quadrilateral element are mapped.

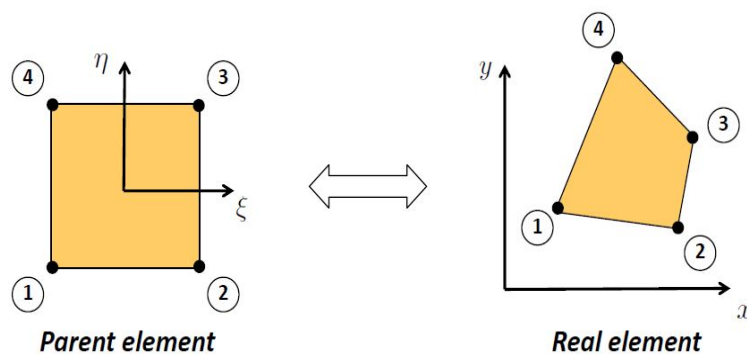


FIGURE 2.6: Isoparametric mapping concept[14]

The shape functions on a parent element can be defined:

$$u_x(\xi, \eta) = \sum_{i=1}^4 N_i(\xi, \eta) u_{xi}$$

$$\text{where, Kronecker } - \delta \text{ property } \begin{cases} N_i(\xi_i, \eta_i) = 1, \\ N_i(\xi_k, \eta_k) = 0, \quad \text{if } i \neq k. \end{cases} \quad (2.47)$$

$$\text{and, partition of unity } \sum_{i=1}^4 N_i(\xi, \eta) = 1 \forall (\xi, \eta) \in [-1, -1] \times [1, 1]$$

From the weak form,

$$\boldsymbol{\varepsilon}(\boldsymbol{\eta}) = \sum_{i=1}^4 \begin{pmatrix} N_{i,x} & 0 \\ 0 & N_{i,y} \\ N_{i,y} & N_{i,x} \end{pmatrix} \begin{pmatrix} \eta_{xi} \\ \eta_{yi} \end{pmatrix} \quad (2.48)$$

$N_{i,x}$  and  $N_{i,y}$  are calculated using the chain rule and result in a Jacobian matrix, which is inverted.

$$\begin{bmatrix} N_{i,x} \\ N_{i,y} \end{bmatrix}^* = \underbrace{\mathbf{J}_e^{-1}}_{\mathbf{B}_i} \cdot \begin{bmatrix} N_{i,\xi} \\ N_{i,\eta} \end{bmatrix}^* \quad *Unknown / *Known \quad (2.49)$$

$$\text{where, } \mathbf{J}_e^{-1} = \begin{bmatrix} x_{,\xi} & y_{,\xi} \\ x_{,\eta} & y_{,\eta} \end{bmatrix} \text{ is the transpose of the Jacobian matrix.}$$

$$\boldsymbol{\varepsilon}(\boldsymbol{\eta}) = \sum_{i=1}^4 \mathbf{B}_i \cdot \boldsymbol{\eta}_i = \mathbf{B}_e \quad (2.50)$$

$$\boldsymbol{\sigma}(\mathbf{u}) = \mathbf{C}_{2D} \cdot \mathbf{B}_e \cdot \mathbf{u}_e \quad (2.51)$$

$$\text{Thus, } \int_{\Omega_t} \boldsymbol{\varepsilon}(\boldsymbol{\eta}) : \boldsymbol{\sigma}(\mathbf{u}_h) dv = \boldsymbol{\eta}_e^T \int_{\Omega_t} \mathbf{B}_e^T \cdot \mathbf{C}_{2D} \cdot \mathbf{B}_e dv \mathbf{u}_e = \boldsymbol{\eta}_e^T \cdot \mathbf{K}_e \cdot \mathbf{u}_e \quad (2.52)$$

The next steps are similar to the formulation of the triangle element.

The isoparametric concept can also be extended to triangle/tetrahedral and Hexahedral elements, figure 2.7.

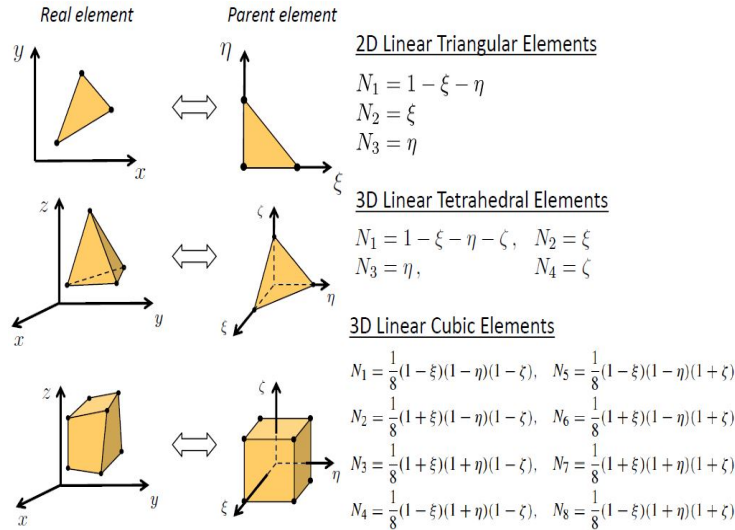


FIGURE 2.7: Isoparametric mapping and shape functions for triangles, tetrahedrons and hexahedrons[14]

In conclusion, table 2.1 compares triangle and quadrilateral elements. The shape functions are compared along with their effect on the elements' performance.

TABLE 2.1: A comparison between triangle and quadrilateral elements

| Triangle element  | Quadrilateral element  |
|---|--|
|   |  |
| $N_{1,\xi} = -1$ $N_{1,\eta} = -1$<br>$N_{2,\xi} = 1$ $N_{2,\eta} = 0$<br>$N_{3,\xi} = 0$ $N_{3,\eta} = 1$<br>$\mathbf{J}_e, \mathbf{B}_e = \text{constant.}$<br>$\Rightarrow \boldsymbol{\varepsilon}(\boldsymbol{\eta}) = \text{constant} \ \& \ \mathbf{K}_e = \int_{\Omega} \text{const}$   | $N_{1,\xi} = \frac{1}{4}(\eta - 1)$ $N_{1,\eta} = \frac{1}{4}(\xi - 1)$<br>$N_{2,\xi} = \frac{1}{4}(1 - \eta)$ $N_{2,\eta} = -\frac{1}{4}(1 + \xi)$<br>$N_{3,\xi} = \frac{1}{4}(1 + \eta)$ $N_{3,\eta} = \frac{1}{4}(1 + \xi)$<br>$N_{4,\xi} = -\frac{1}{4}(1 + \eta)$ $N_{4,\eta} = \frac{1}{4}(1 - \xi)$<br>$\mathbf{J}_e, \mathbf{B}_e \neq \text{constant}$                                    |
| <p><b>[-]</b> Strain field is constant in the element. This may be a poor representation of strains in some physical problems.</p> <p><b>[+]</b> The stiffness matrix is found by integrating a constant. The computation is simple.</p> <p><b>[-]</b> Larger number of elements when discretizing with triangles.</p> <p><b>[+]</b> Arbitrary domains can be discretized easily.</p> | <p><b>[+]</b> Strain field is not constant in the element. This will represent strains with more accuracy.</p> <p><b>[-]</b> Because the stiffness matrix is to be calculated by integrating a non-constant variable, the computation is more complex.</p> <p><b>[+]</b> Smaller number of elements.</p> <p><b>[-]</b> Not flexible enough to easily generate meshes for arbitrary geometries.</p> |

## 2.2 Higher Order Elements

Quadrilaterals and hexahedrons aren't the simplest shapes to work with when creating finite meshes. They require significant effort from the user to fill up spaces. This is especially true, if the geometry is complex and complicated. Triangles and tetrahedrons offer an alternative to overcome this disadvantage of quadrilaterals and hexahedrons. Creating meshes is a simpler task with triangles and tetrahedrons. They are also called simplices because of their simplicity.

But, a substantial problem with using triangles and tetrahedrons is that the strain field is constant. Hence, no variation of strain across the element can be calculated. Results using triangular elements can be erroneous under combinations of loads, geometry and meshing.

The linear shape functions on elements can undergo a phenomenon called *shear locking*. This behaviour is very prominent in cases, where bending is present. The linear elements can't model the actual curvature, under bending loads. Hence, a spurious shear stress is introduced and the material shears instead of bending. This causes the element to reach equilibrium at a smaller displacement. Thus, the model appear to be stiffer than it actually is. This deterioration can be more pronounced on the triangles' spatial counterpart, the 4-node tetrahedron elements. This is because shear effects are more prominent in three-dimensions.

When the material is incompressible, *pressure locking* can occur. The behaviour is similar to shear locking i.e. the element acts stiffer than it actually is. For Tetrahedrons, this phenomena is termed as volumetric locking.

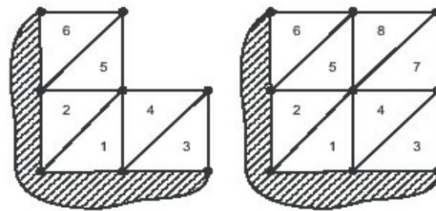


FIGURE 2.8: Locked triangular elements[13]

Shear locking can be avoided to an extent in certain cases by using a sufficiently fine mesh. However, with a fine mesh, there is always the trade-off with the computational cost. It may be lucrative to have a lower computational cost FE model.

Shear locking can also be avoided by using more sophisticated shape functions on the elements. An element, whose edges are able to curve can successfully evade shear locking. Before introducing such shape functions, it would benefit to study the requirements from shape functions. The requirements are motivated by mesh convergence.

- **Compatibility:** The interpolation has to be such that the field of the unknown quantity( e.g displacement) is:
  - continual and derivable inside the element
  - continuous across the element boundary

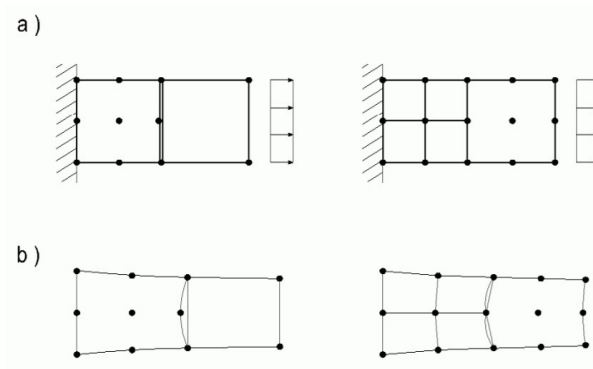


FIGURE 2.9: Un-compatible elements (a) Discretization and load (b) Deformed Shape[28]

- **Completeness:** The interpolation must be able to represent:
  - the rigid body displacements
  - constant strain state

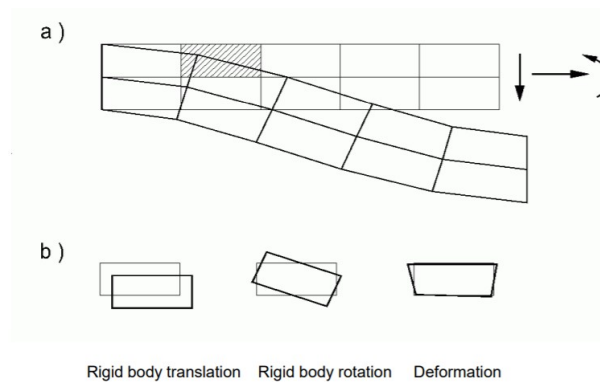


FIGURE 2.10: (a) Deformation of cantilever beams (b) Rigid body displacement of grey element[28]

The triangle and tetrahedron element with linear shape functions can be observed in figure 2.7. Triangular and tetrahedron elements with quadratic, cubic or higher-order shape functions are higher-order elements.

A quadratically interpolated triangle is defined by 6 nodes, 3 at the vertices and 3 at the middle of each edge. The edges may be defined by a straight or quadratic line. Figure 2.11 indicates a 6-node quadratic triangle element.

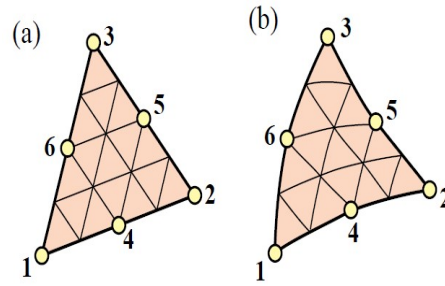


FIGURE 2.11: The 6-noded quadratic triangle element (a) straight edges and midside nodes at midpoints. (b) the isoparametric triangle quadratic element[29]

Shape functions for quadratic triangles can be expressed as products of linear functions:

$$N^i = c_i L_1 L_2 \dots L_n$$

where,  $L_j = 0, j = 1, \dots, n$

(2.53)

are the homogeneous equation of lines on which  $N^i$  vanishes and  $c_i$  is a normalisation coefficient.

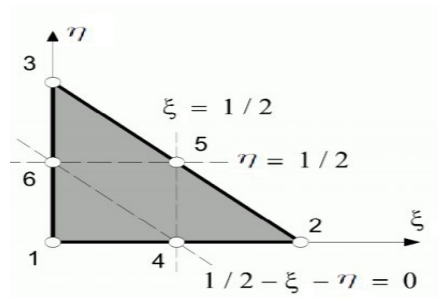


FIGURE 2.12: The 6-noded quadratic triangle element coordinates[28]

The shape functions can thus be written as:

$$\begin{aligned}
 N_1 &= (1 - \zeta - \eta)(1 - 2\zeta - 2\eta) \\
 N_2 &= \zeta(2\zeta - 1) \\
 N_3 &= \eta(2\eta - 1) \\
 N_4 &= 4\zeta(1 - \zeta - \eta) \\
 N_5 &= \zeta\eta \\
 N_6 &= 4\eta(1 - \zeta - \eta)
 \end{aligned}$$

(2.54)

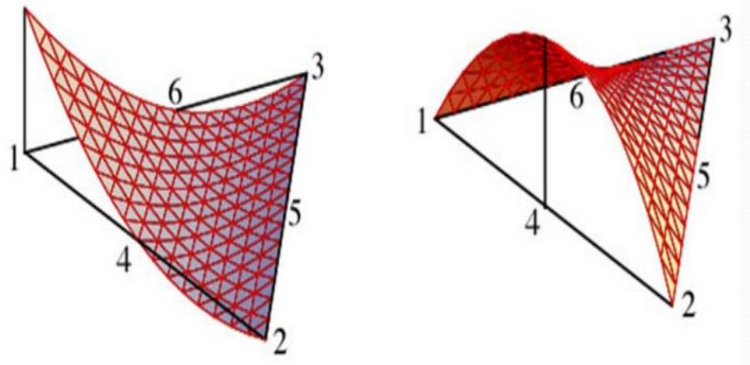


FIGURE 2.13: Shape functions  $N_1$  and  $N_4$  for the quadratic triangle. The shape function is 1 at node 1 and 4 respectively, and 0 everywhere else.[28]

Similarly, a tetrahedron with quadratic shape functions can also be defined.

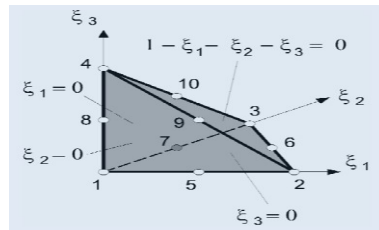


FIGURE 2.14: A tetrahedron with quadratic shape functions[28]

$\zeta_1$ ,  $\zeta_2$  and  $\zeta_3$  are the coordinates of the tetrahedral element, Thus, the shape functions are:

For the corner nodes 1 to 4:

$$\begin{aligned}
 N_1 &= 1 - \zeta_1 - \zeta_2 - \zeta_3 \\
 N_2 &= \zeta_1 \\
 N_3 &= \zeta_2 \\
 N_4 &= \zeta_3
 \end{aligned} \tag{2.55}$$

Shape functions of nodes 5 and 10 on the edges:

$$\begin{aligned}
 N_5 &= 4\zeta_1(1 - \zeta_1 - \zeta_2 - \zeta_3) \\
 N_6 &= 4\zeta_1\zeta_2 \\
 N_7 &= 4\zeta_2(1 - \zeta_1 - \zeta_2 - \zeta_3) \\
 N_8 &= 4\zeta_3(1 - \zeta_1 - \zeta_2 - \zeta_3) \\
 N_9 &= 4\zeta_1\zeta_3 \\
 N_{10} &= 4\zeta_2\zeta_3
 \end{aligned} \tag{2.56}$$

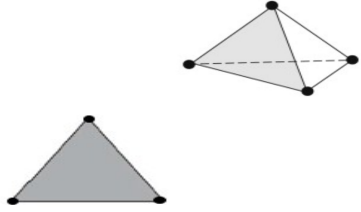
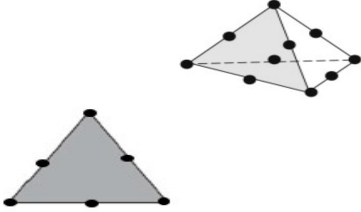
The shape functions at the corner must be corrected to ensure that the shape functions are 1 at the corresponding node and 0 elsewhere.

$$\begin{aligned}
 N_1 &\leftarrow N_1 - 0.5(N_5 + N_7 + N_8) \\
 N_2 &\leftarrow N_2 - 0.5(N_5 + N_6 + N_9) \\
 N_3 &\leftarrow N_3 - 0.5(N_6 + N_7 + N_{10}) \\
 N_4 &\leftarrow N_4 - 0.5(N_8 + N_9 + N_{10})
 \end{aligned}
 \tag{2.57}$$

So, now the shape functions are quadratic and when they are differentiated as in equation 2.37, the result is not a constant. Because of its variation, this can represent strain measures in a more accurate way as compared to the linear triangle or tetrahedron element.

In Conclusion, a comparison between the linear and quadratic triangle/tetrahedron elements can be made.

TABLE 2.2: A comparison between linear and quadratic triangle/tetrahedron elements

| Linear shape functions  | Quadratic shape function   |
|---|--|
|    |    |
| <p>[+]Fewer Gauss nodes<br/>                 [+]Smaller stiffness matrix<br/>                 [+]These two lead to less computational time.</p> | <p>[-]Several Gauss nodes<br/>                 [-]Larger stiffness matrix<br/>                 [-]Higher computational time is required.</p> |
| <p>[-]The strain gradients are approximated by constants.<br/>                 [-]Hence, the accuracy is low.</p>                               | <p>[+]The strain gradients are approximated by non-constants.<br/>                 [+]Hence, the accuracy is higher.</p>                     |

Hence, it is generally recommended that the polynomial order of the shape functions be higher in regions of large gradients to capture them accurately. Also, triangles/tetrahedrons elements are easier to work with, specially in complex geometries. Thus, the recommendation is to use second-order triangles/tetrahedrons.



## Chapter 3

# Finite Element Types

After the Finite element method has been introduced briefly in the last chapter, *Chapter 2*, the next step is to understand the factors affecting the accuracy of the simulation results. With the present day finite element tools' capability, it is easy to compute results as colour plots. But, the most important aspect of a finite element simulation is the precision of the solution. There are many aspects at play here. Does the element size capture the geometry to a satisfactory level? Are there any distorted, thin, bad elements? How are the boundary conditions setup? It is also important to provide correct material properties to ensure a precise solution. The element types also play a major role in the accuracy. This is evident in this Chapter when different element types are used to evaluate well-known textbook problems with analytical/closed-form solutions. Different results are arrived at for each element type.

A little bit of history is important at this point to understand the motivation behind using the different element types. The Finite-Element-Method was developed well before the first electronic Computers. Some of the first Computers used to solve the first finite element problems in the area of structural mechanics were with very less memory. Thus, first-order elements were the preferred choice to save memory and clock cycles. First-order Bricks or 8-noded Hexahedrons give better results than first-order Tetrahedrons. Due to this legacy, brick elements were preferred by some engineers [6].

Brick/Hexahedral or its 2D counterpart, Quadrilateral elements ensure a model that is computationally inexpensive. They are more accurate because of how they are computed. Although, it can require significant user effort to generate the mesh. A Tetrahedral mesh can be generated easily without any user interaction but can be computationally expensive, based on the geometry.

This chapter is subdivided into two parts, where in the first part classifications of element types are described. Classification is done based on several criteria. Then, classical problems with known analytical solutions are compared with the solutions

from numerical models of the same. Models using first a beam and then a finite plate are studied and evaluated.

### 3.1 Classification of element types

Element types can be categorized based on several criterion [12]. These can be based on:

#### 1. Number of adjacent elements to each inner node

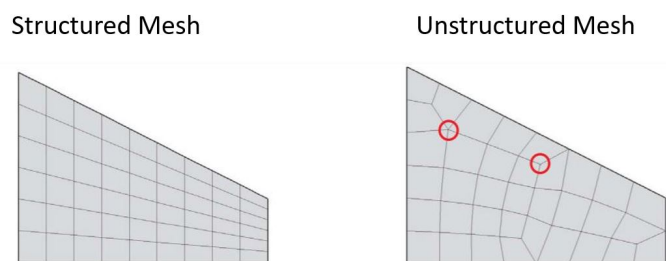


FIGURE 3.1: Structured and Unstructured meshes[12]

#### Structured mesh

- Constant number of elements adjacent to any inner node
- Basic arithmetic is sufficient to determine which elements surround a vertex
- More suited to domains which satisfy some pre-defined constraints

#### Unstructured mesh

- Can have different number of elements around any inner node
- Information about elements around a vertex must be stored
- Can be easily used for arbitrary domains

#### 2. Intersection between neighbouring elements

#### Conformal mesh

- There are no hanging nodes
- They are more common in the industry
- Restrictive in dealing with non-uniform element size

#### Non-conformal mesh

- There are hanging nodes
- They are less usual in the industry
- Flexible in dealing with non-uniform element size

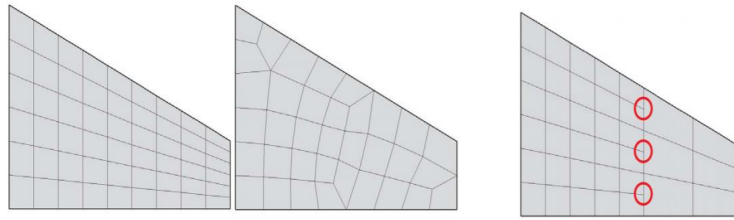


FIGURE 3.2: Conformal and Non-conformal meshes[12]

### 3. Two-dimensional elements' shape



FIGURE 3.3: Triangle and Quadrilateral elements[12]

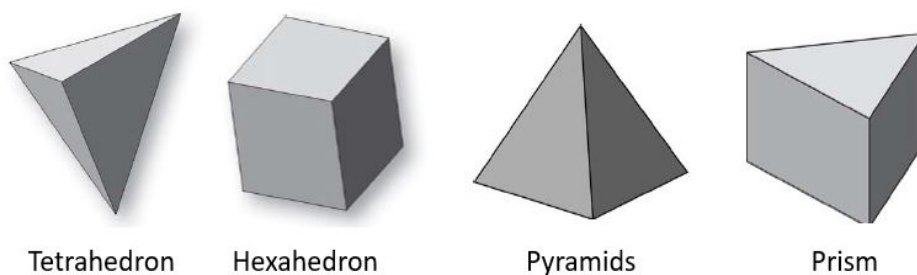
#### Triangle element

- Can be easily used for any domain
- Requires lesser user interaction to generate
- Less accurate due to the way shape functions(linear) are built for linear elements

#### Quadrilateral element

- Can be used easily for specific domains only
- Requires more user interaction to generate
- More accurate due to the way shape functions(bilinear) are built for linear elements

### 4. Three-dimensional elements' shape



Tetrahedron

Hexahedron

Pyramids

Prism

FIGURE 3.4: Three-dimensional element shapes[12]

- |  |  |  |   |
|--|--|--|---|
| <ul style="list-style-type: none"> <li>• Composed of 4 triangles</li> <li>• Most common choice for any analysis</li> <li>• Second-order TET elements generally give the most accurate results</li> </ul> | <ul style="list-style-type: none"> <li>• Composed of 6 quadrilaterals</li> <li>• More accurate results among first-order elements</li> </ul> | <ul style="list-style-type: none"> <li>• Composed of 4 triangles and 1 quadrilateral</li> <li>• Can be used as an interface where both tetrahedrons and hexahedrons are present</li> </ul> | <ul style="list-style-type: none"> <li>• Composed of 2 triangles and 3 quadrilaterals</li> <li>• Can be used as an interface between tetrahedrons and hexahedron</li> </ul> |
|--|--|--|---|

### 5. Element order

Element order refers to the polynomial order of the element's shape functions. Since in FEM, the results are only calculated on the nodes, a mechanism is needed to carry the results from the nodes to the elements. The shape functions accomplish exactly this and can be defined as a mathematical function that define the shape of the element results. The accuracy of the FEM analysis is dependent on how close the element shape functions agree with the real solution. An element can be linear or quadratic. Naturally, higher orders exist but these two are the most common elements.

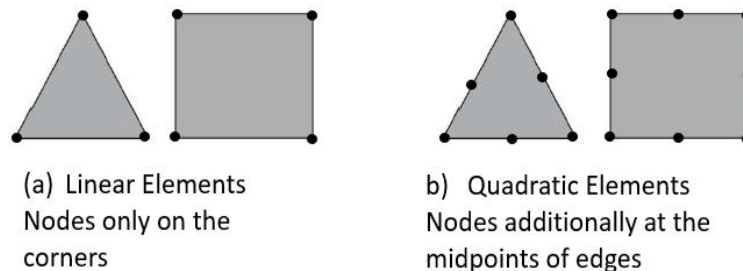


FIGURE 3.5: Linear and Quadratic elements

- Linear elements can support only linear variation in the unknown quantity, e.g. displacement. The gradient of the unknown quantity, so strain and thus stress are constant (specifically for triangles) in a single element.
- They are more suited to analyses where the intent is to only output the nominal stress results.
- A large number of elements are generally required to output an acceptable level of resolution of stress gradient.
- Highly sensitive to distortion.
- Quadratic elements can support quadratic variation in the displacement and hence a linear variation in the strain and stress in a single element.
- Analyses with quadratic elements can output highly accurate stresses.

- In many cases, they can output better results than linear elements with fewer DOFs.
- Curved edges and surfaces are represented more accurately. They aren't as sensitive to element distortion.

There is a further class of such elements, which have an inner mid node. For e.g: a 9-noded quadratic element in figure 3.6. They aren't commonly used in the industry.

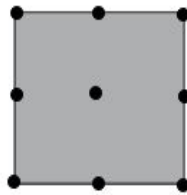


FIGURE 3.6: A Quadrilateral element with 9 nodes

A further comparison between the results obtaining these two elements, linear and quadratic can be seen in figure 3.7. The worst results are obtained using the linear elements. Multiple linear elements ensure a better result, but still there is an error from the actual quadratic distribution of the DOFs. The closest results are obtained using quadratic elements.

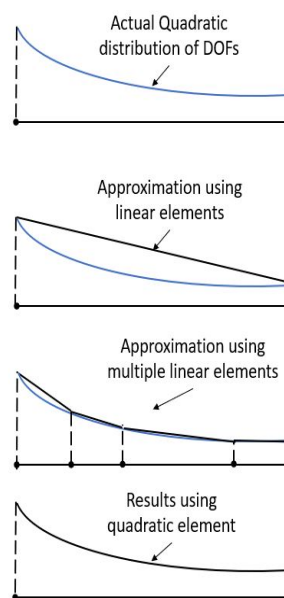


FIGURE 3.7: FEM Analysis results comparison using linear and quadratic elements

### 3.1.1 Geometry and selection of element type

The group of geometry of interest in this thesis are turbine blades. A typical turbine blade by GE can be seen in figure 3.8. Such a blade has a complex solid geometry made of several cooling holes and complex air flow paths. Thus, only three-dimensional elements are of interest, mainly Tetrahedrons and Hexahedrons.



FIGURE 3.8: A High Pressure turbine blade from a GE jet engine[31]

So, in the next sections, a study between element types is done. Simple models are meshed using Tetrahedrons and Hexahedrons. Then, boundary conditions are applied. Afterwards, the numerical simulation is performed. Finally, the results from the simulation is compared to that of the analytical/closed-form solution. Based on the comparison, eventually, a conclusion is drawn at the end of study as to which is the preferred element type closest to the analytical solution.

## 3.2 Finite Element Method Solution Tool

The tool used to complete the analyses in the next sections is *BasicFEA*. It is a commercial tool available with Hyperworks package. *The objective for BasicFEA is to define a user profile that will allow users from the novice to expert level to run a broad range of simple analyses. BasicFEA aims to bring the power of the solver to a wider user base using a streamlined process based approach. The interface was designed for those who might not use FEA everyday, but still want a simple way to set up a basic analysis*[10].

### 3.3 Beam

To evaluate the performance of the element types in bending dominated problems, a simply supported cantilever beam under bending load is analyzed. The behaviour of the element types in the frequency domain is also studied. This is established through a frequency/Modal analysis. Three beam models for the 2 cases are created using these element types:

- First-order/8-noded/Linear Hexahedrons
- First-order/4-noded/linear Tetrahedrons
- Second-order/10-noded/Quadratic Tetrahedrons

Then, each of model is analysed and finally, the finite element solutions are compared with the known analytical solutions of a beam.

#### 3.3.1 Bending Analysis

A simply supported 3-dimensional beam, figure 3.9 under pure bending is considered. It is loaded with a point load at one end. The analytical solution to this problem can be easily calculated using beam theory.

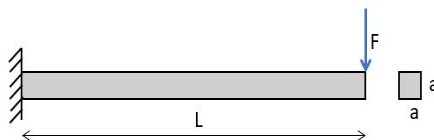


FIGURE 3.9: Simply supported Cantilever Beam with point load

The dimensions and properties of the beam are,

$$\begin{aligned}
 \text{Length, } L &= 100 \text{ mm} \\
 \text{Width, } a &= 10 \text{ mm} \\
 \text{Cross section area, } A &= a^2 = 100 \text{ mm}^2 \\
 \text{Moment of inertia, } I &= \frac{1}{12}a^4 = 833.33 \text{ mm}^4 \\
 \text{Young's Modulus, } E &= 210 \text{ GPa} \\
 \text{Poisson's Ratio, } \nu &= 0.3 \\
 \text{Point load, } F &= 1000 \text{ N} \\
 \text{Beam Tip Deflection, } \delta &= \frac{FL^3}{3EI} = 1.9048 \text{ mm}
 \end{aligned} \tag{3.1}$$

Thus, the defined beam under pure bending deflects by  $1.905 \text{ mm}$ . The next step is to create the numerical model with the same dimensions and materials as in the analytical model. Three models are created: First-order Hexahedrons, First-order Tetrahedrons and Second-order Tetrahedrons. Each of the model is fixed on the left end and loaded with a point load,  $F = 1000 \text{ N}$  at the right end.

The boundary conditions can be studied in figure 3.10. The boundary conditions are always applied to a surface, which is associated with elements and nodes. Hence, the boundary conditions are transferred to the nodes from the surface. The model is linear and no non-linearity are assumed. The load doesn't follow the beam deflection.

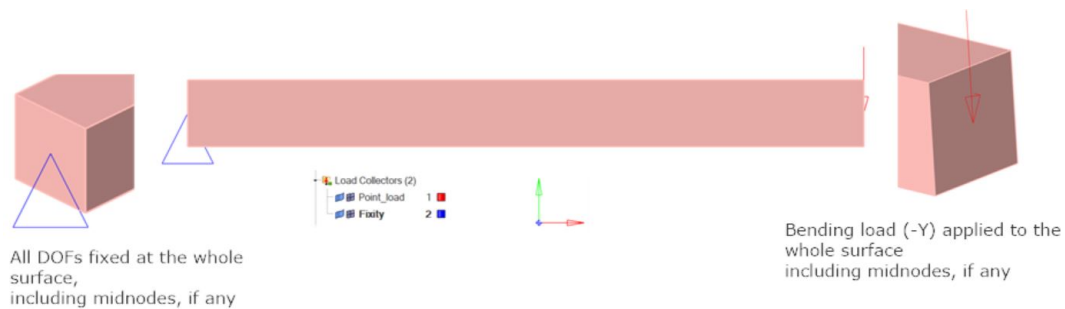


FIGURE 3.10: Boundary conditions for simply supported beam in numerical model

### First-order Hexahedrons

First, the FE model meshed with first-order hexahedrons is studied. The beam is meshed with 10 elements in the beam's depth, figure 3.11. The boundary conditions are applied and the simulation is completed.

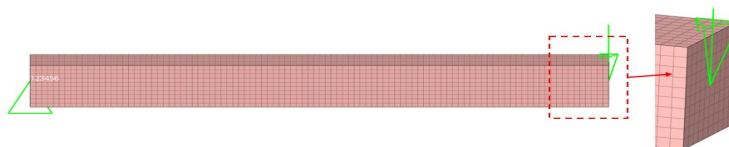


FIGURE 3.11: Cantilever beam meshed with first-order hexahedrons: 10 elements in depth



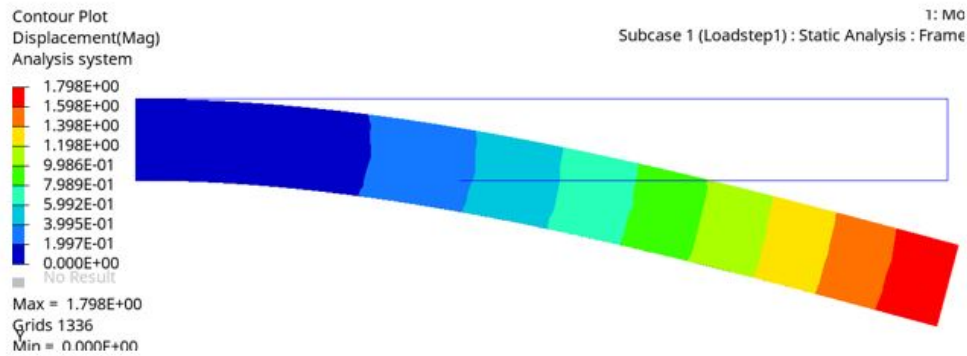


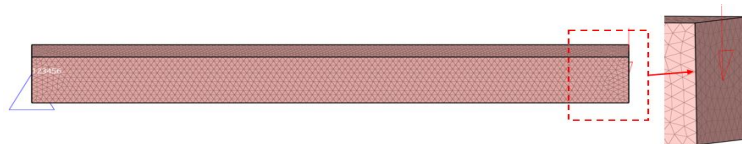
FIGURE 3.12: Beam tip deflection with first-order hexahedrons

Max beam tip deflection of FE model with first-order hexahedrons =  $1.798 \text{ mm}$

The tip deflection in this model is farthest from the analytical solution, figure 3.12. The stiff linear shape functions can be attributed to this behaviour.

### First-order tetrahedrons

Here, first-order tetrahedrons beam model is evaluated. 20 nodes are created in the beam's depth, figure 3.13. Similar to the first-order hexahedrons, the beam is fixed on the one side in all DOFs and a point load,  $F = 1000 \text{ N}$  is applied on the other end.

FIGURE 3.13: Cantilever beam meshed with first-order tetrahedrons:  
10 nodes in depth

First-order tetrahedrons tend to be too stiff in bending and thus, the beam tip deflection differs from the analytical solution, figure 3.14. The linear shape functions of linear tetrahedrons account for the element being too stiff.

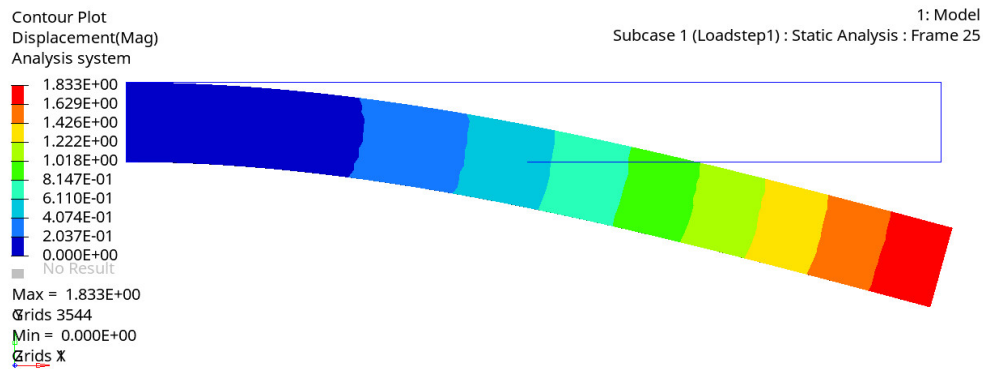


FIGURE 3.14: Beam tip deflection with first-order tetrahedrons

Max beam tip deflection of FE model with first-order tetrahedrons = 1.833 mm.

### Second-order Tetrahedrons

Finally, the cantilever beam model is created using second-order tetrahedrons. These tetrahedrons use quadratic shape functions (section 2.2). The previous model with first-order Tetrahedrons is used and mid-nodes are added to each element to make it second-order. 40 nodes are created in the beam's depth. Each element has a second order shape function. The tetrahedrons with quadratic shape functions show deflection results closest to the exact analytical solution for pure bending dominated problem as seen in figure 3.15.

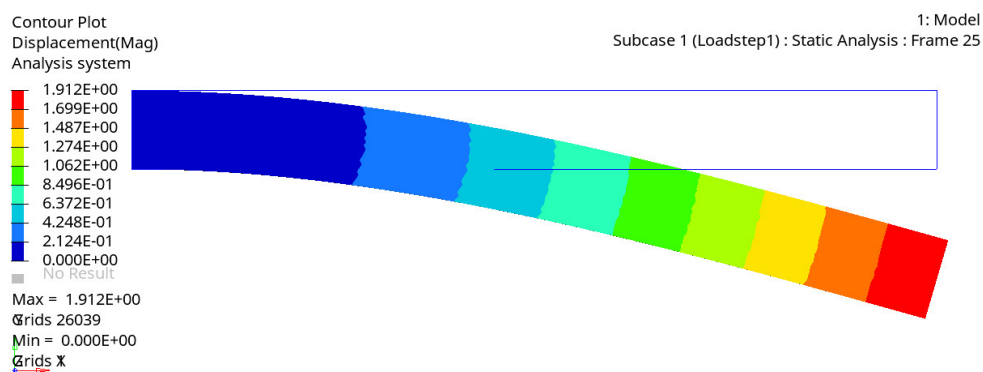


FIGURE 3.15: Beam tip deflection with quadratic tetrahedrons

Max beam tip deflection of FE model with second-order tetrahedrons = 1.912 mm.

Figure 3.16 summarizes the beam bending problem solved analytically and numerically with the different element types. It is clear that the linear hexahedrons and first-order tetrahedrons aren't suited to bending dominated problems. The cause to

this behaviour can be attributed to *shear locking* (Section 2.2). Second-order tetrahedrons are the best choice for bending problems due to its results matching closest with the analytical one. The quadratic shape functions in second-order tetrahedrons ensure that the element edges curves and hence, depicts accurate stiffness.

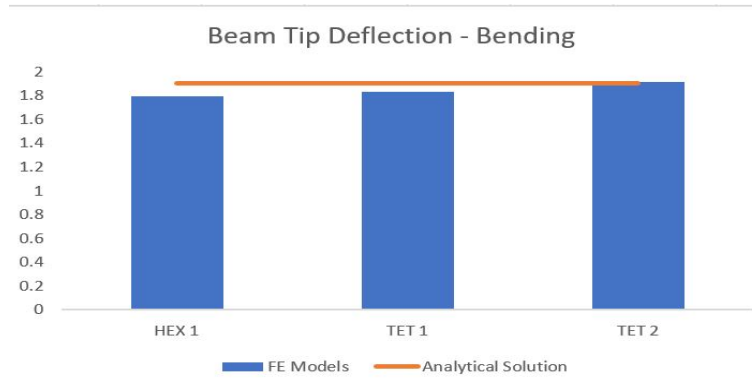


FIGURE 3.16: Beam tip deflection: Analytical and Numerical results

### 3.3.2 Modal Analysis

To study the effect on structural stiffness due to the use of different element types, a free-free modal analysis of a beam is performed. To begin with, the first two eigenfrequencies are calculated analytically with the beam described in figure 3.17.

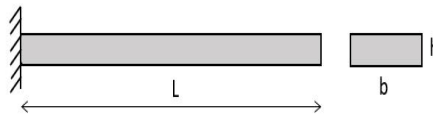


FIGURE 3.17: Beam for frequency analysis

Dimensions of the beam:

$$\begin{aligned}
 L &= 100 \text{ mm} & b &= 10 \text{ mm} & h &= 1 \text{ mm} \\
 \text{Young's Modulus, } E &= 119.5 \text{ GPa} \\
 \text{Poisson's Ratio, } \nu &= 0.29 \\
 \text{Density, } \rho &= 8.05e^{-9} \text{ tonne/mm}^3 \\
 \text{Moment of inertia, } I &= 0.833 \text{ mm}^4
 \end{aligned} \tag{3.2}$$

Uniform load per unit length(including weight),  $w = 7.9e^{-6} \text{ N/mm}$

$$\text{Natural frequency, } f_n = \frac{K_n}{2\pi} \sqrt{\frac{EIg}{wL^4}}$$

$K_n$  is a constant which refers to each mode. (3.3)

$$K_1 = 3.52, K_2 = 22.0$$

$$\text{Therefore, } f_1 = 82.643 \text{ Hz, } f_2 = 516.517 \text{ Hz}$$

Then, three FE models are created using first-order hexahedrons, first-order tetrahedrons and second-order tetrahedrons. For each model, the first two natural Eigen-frequencies and the mode shapes are computed. Figure 3.18 and 3.19 compare the three FE models' first and second mode shapes and eigen-frequencies respectively.

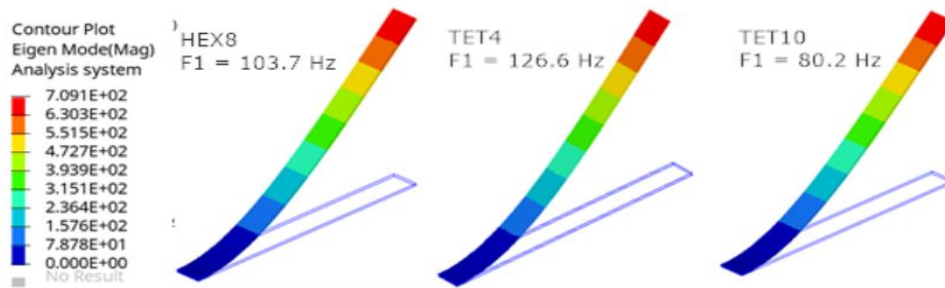


FIGURE 3.18: Effect of element type in a free-free modal analysis: Mode 1

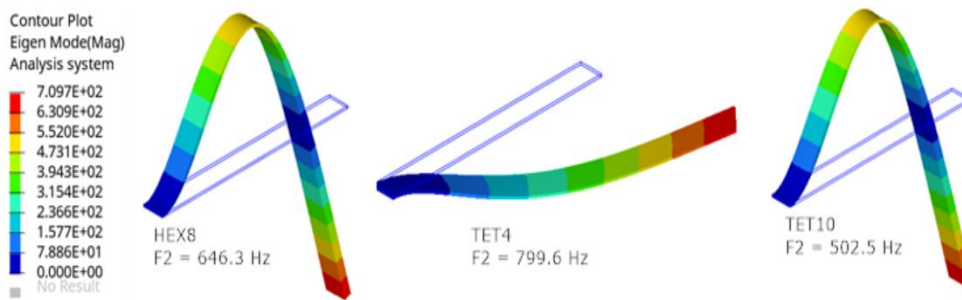


FIGURE 3.19: Effect of element type in a free-free modal analysis: Mode 2

TABLE 3.1: Free Modal analysis: Summary of analytical and FE model results

| Model         | 1 <sup>st</sup> Mode Freq. (Hz) | 2 <sup>nd</sup> Mode Freq. (Hz) | Error: first mode | Error: second mode |
|---------------|---------------------------------|---------------------------------|-------------------|--------------------|
| Analytical    | 82.6                            | 516.5                           | 0.00%             | 0.00%              |
| FE with HEX8  | 103.7                           | 646.3                           | 25.48%            | 25.13%             |
| FE with TET4  | 126.6                           | 799.6                           | 53.19%            | 54.81%             |
| FE with TET10 | 80.2                            | 502.5                           | 2.95%             | 2.71%              |

The linear tetrahedron perform the worst when compared to the analytical solution, showing an error of almost 55%. The mode shape is also different for the second mode for the first-order tetrahedrons. On the other hand, Hexahedrons perform slightly better than linear Tetrahedrons with an error of about 25%. The model with the quadratic Tetrahedrons are quite close to the analytical solution and thus, are also a good choice for modal analyses.

### 3.4 Stress Concentration

Stress concentrations occur when there are irregularities in a component that cause an interruption to the flow of stress, figure 3.20.

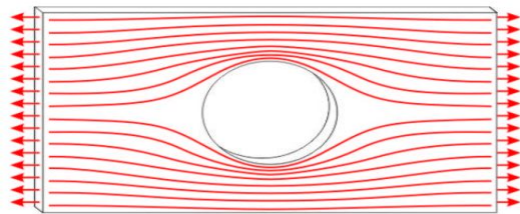


FIGURE 3.20: Flow of stress is denser near the hole

As it is impossible to have components without irregularities, stress concentrations are important. Thus, the effect of element types on stress concentrations are studied in this section. A simple finite plate with a notch is used for this study. The notch is a circular hole at the centre of the plate. There are two types of loading applied on the plate and its effect studied:

- Tension
- Bending Moment

Similar to the last section, the study starts with known closed-form solutions. Charts from Peterson's Stress Concentration Handbook[9] are used to determine the Stress concentration factor. Then, three numerical models are created with first-order hexahedrons, first-order and second-order tetrahedrons. Appropriate boundary conditions are applied and the results are calculated. The numerical results are then compared to the closed-form solutions.

#### 3.4.1 Tension

The plate with a circular hole is loaded with tensile loading on both ends, figure 3.21. Introducing a hole in the plate disturbs the uniform stress distribution near the

hole, resulting in a significantly higher than average stress. The stress concentration factor is a function of the hole diameter to the plate width.

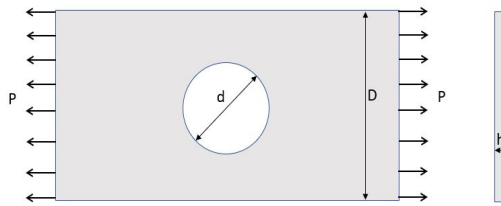


FIGURE 3.21: A finite plate with a hole

Dimensions of the Plate are:

$$\begin{aligned}
 D &= 50 \text{ mm} & d &= 20 \text{ mm} & h &= 10 \text{ mm} \\
 \frac{d}{D} &= 0.4 & & & & \\
 \text{Tensile load, } P &= 1000 \text{ N} & & & & \\
 \text{Nominal Stress in the plate, } \sigma_0 &= \frac{P}{(D-d)h} = 3.33 \text{ MPa} & & & & 
 \end{aligned} \tag{3.4}$$

Peterson's Stress Concentration Handbook [9] is used to read the closed-form Stress concentration of the plate with hole under tension. It can be referred in figure 3.22 and is **2.23**.

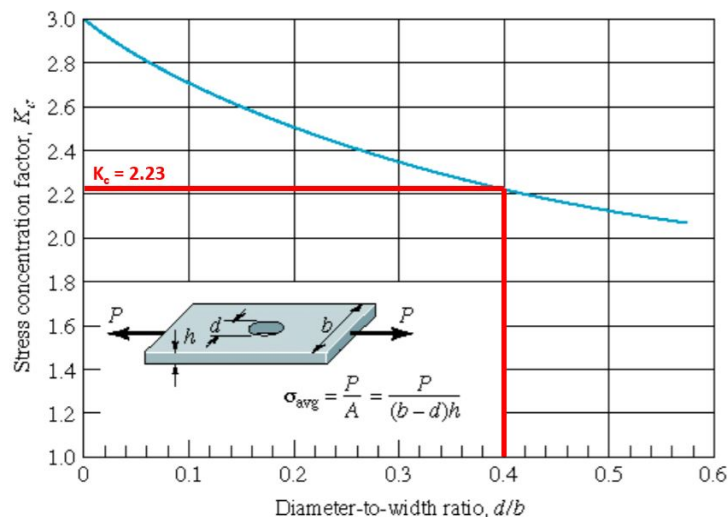


FIGURE 3.22: Closed-form Stress Concentration [9]

The same plate is now used to create numerical models. Three Finite Element models are created using the 3 element types. A tensile loading of 1000 N is applied at both ends of the FE models. The boundary conditions application can be studied in figure 3.23. The tension load is applied to the surface of the plate. The load is

transferred to all nodes. Mid side nodes are included in the load application, for the model with second-order tetrahedrons.

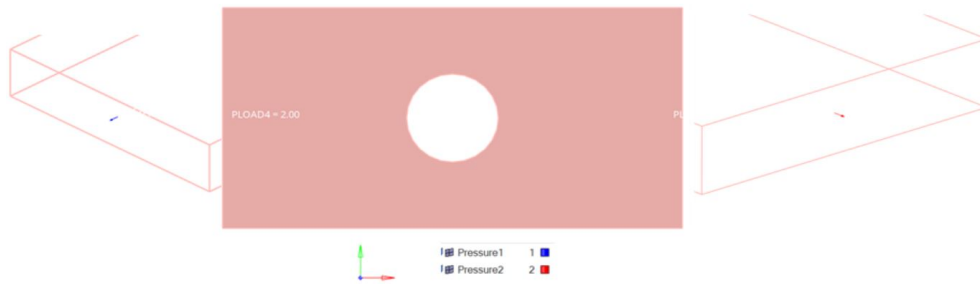


FIGURE 3.23: Boundary conditions for a plate with a hole to study stress concentrations under tensile loads

The results from the models is compared to each other and against the closed-form solution in figure 3.24 and table 3.2 respectively.

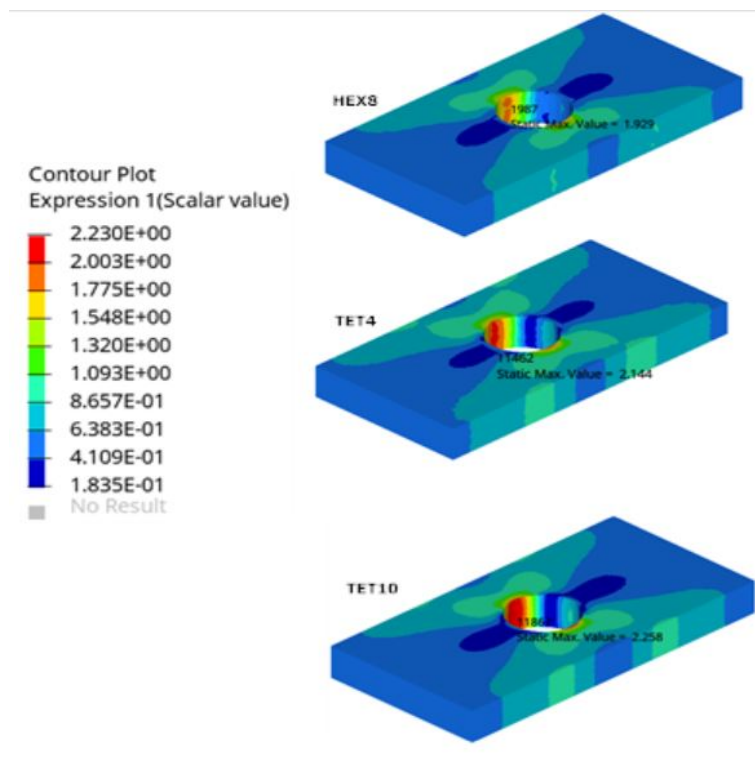


FIGURE 3.24: Tension loading: Stress concentration plots

TABLE 3.2: Stress concentration on a Plate with a hole subjected to tensile loading: Summary of closed-form and FE model results

| Model         | Stress Concentration | % Error in comparison to closed-form solution |
|---------------|----------------------|---|
| Closed-form   | 2.23                 | 0.00%   |
| FE with HEX8  | 1.93                 | 13.45%  |
| FE with TET4  | 2.14                 | 4.04%   |
| FE with TET10 | 2.26                 | 1.35%   |

Again, results using second-order tetrahedrons are closest to the closed-form solution than the one using hexahedrons or first-order tetrahedrons. Hence, second-order tetrahedrons are suited to problems with stress concentrations under tensile loads.

### 3.4.2 Bending Moment

The stress concentration factor for a plate with rectangular cross-section with a central circular hole is analysed for out-of-plane bending. The plate, figure 3.25, has an out-of-plane moment applied.

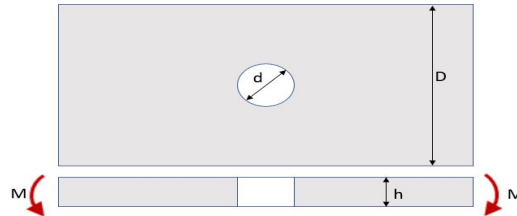


FIGURE 3.25: A plate with hole subjected to out-of-plane bending moment

The dimensions of the plate are:

$$\begin{aligned}
 L &= 50 \text{ mm} & d &= 10 \text{ mm} & D &= 50 \text{ mm} & h &= 10 \text{ mm} \\
 \frac{d}{D} &= 0.2 \\
 \text{Bending moment, } M &= 1 \text{ kNmm} \\
 \text{Nominal Stress in the plate, } \sigma_o &= \frac{6M}{t^2(D-d)} = 1.5 \text{ MPa}
 \end{aligned} \tag{3.5}$$

The closed-form solution is also present for this case. From the Peterson's Stress Concentration Handbook[9], figure 3.26 is used to get the analytical stress concentration and is 1.85.



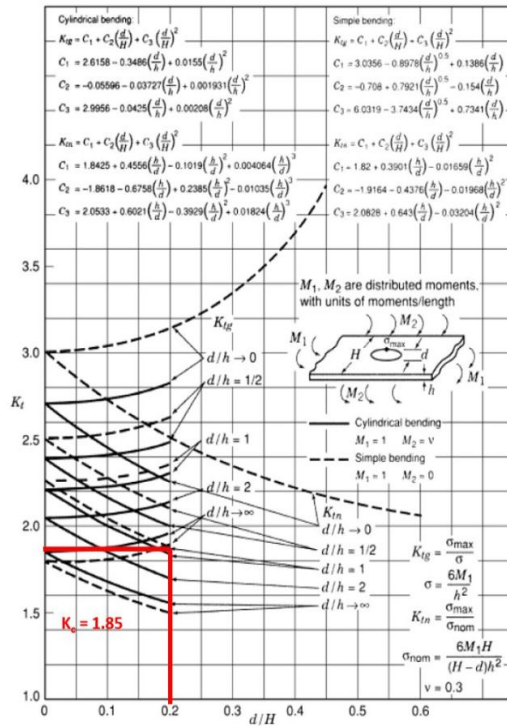


FIGURE 3.26: Closed form Stress concentration - Plate with a hole

Thereafter, the numerical models are built and the appropriate boundary conditions, similar to the analytical model are applied. They are depicted in figure 3.27. Bending moment is applied through rigid body couplings. All the nodes (mid side nodes also for quadratic tetrahedrons) on the surface are coupled to one master node. The moment is applied on the master node, then distributed to the slave nodes.

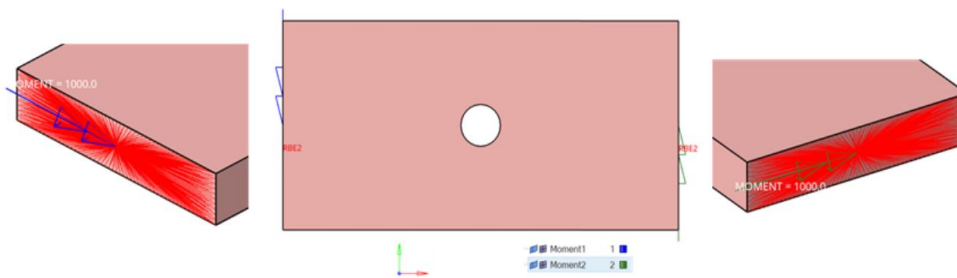


FIGURE 3.27: Boundary conditions for a plate with a hole to study stress concentrations under bending moment

A comparison between the models built using different element types can be seen in figure 3.28 and in table 3.3.

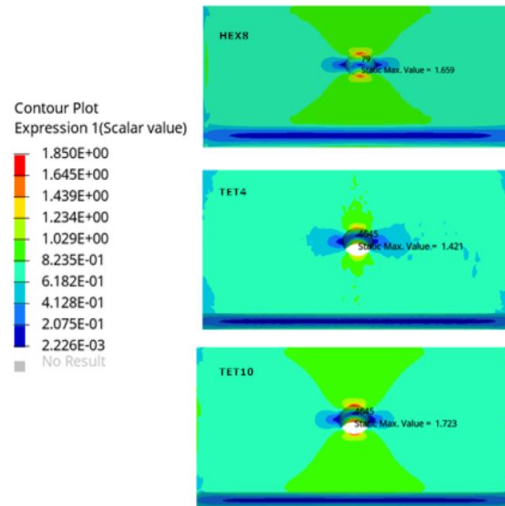


FIGURE 3.28: Bending load: Stress concentrations

TABLE 3.3: Stress concentration on a Plate with a hole subjected to **bending moment**: Summary of closed-form and FE model results

| Model                | Stress Con-<br>centration | % Error |
|----------------------|---------------------------|---------|
| <b>Closed-form</b>   | 1.85                      | 0.00%   |
| <b>FE with HEX8</b>  | 1.66                      | 10.27%  |
| <b>FE with TET4</b>  | 1.42                      | 23.24%  |
| <b>FE with TET10</b> | 1.72                      | 7.03%   |

The quadratic Tetrahedrons results are closer to the closed-form solution when compared to the Hexahedrons or linear Tetrahedrons.

### 3.5 Conclusion

At the end, based on the tests performed in the last few sections, it can be concluded that quadratic/second-order/10-noded Tetrahedrons are a better choice all the cases evaluated here. These elements aid in a more accurate result. The way the shape functions are defined can be attributed for this behaviour. Linear elements behave stiffer than they actually are, the shear lock affect. It is the behaviour when the elements don't model the actual curvature accurately due to their linear shape functions. Linear hexahedrons are better than linear tetrahedrons because the strains in linear tetrahedrons are constant. This is due to the way the element shape functions are defined. Second-order tetrahedrons can follow the actual curvature because of the quadratic shape functions defined on them. The strains are linear and thus can

represent problems better than linear tetrahedrons. This can be studied in more detail and mathematically in chapter 2.

Another advantage of tetrahedrons over hexahedrons is that they are generally easier to generate. Complex, arbitrary models can be generated easily using tetrahedrons. Tetrahedrons also lend themselves well to be automated when compared to hexahedrons. The methods and algorithms used to generate tetrahedrons are studied in chapter 4.

A disadvantage of tetrahedrons can be attributed to the increased number of elements, leading to bigger element stiffness matrices. This could mean potentially that tetrahedrons are computationally costlier than hexahedrons.

## Chapter 4

# Mesh Generation Methods

Mesh generation is a necessary tool in the computational simulation world of physical phenomena. Mathematics forms the base of the mesh generation processes. The generation process in itself is not unique and there are no inherent laws to generate a mesh. There are however optimization criteria that can help establish a mesh generation process. The world of mesh generation has reached a point that several, commercial and open-source programs are available. These programs use several well-known and established methods/algorithms to conceive the elements and, consecutively the mesh.

In Chapter 3, it was established that second-order tetrahedrons are the preferred element type for most analyses. Subsequently, this Chapter talks about the methods/algorithms used for generating tetrahedrons only.

Based on the present-day methods available, mesh generation algorithms for tetrahedrons can be described in three classes:

- *Advancing front method*, wherein elements are generated one by one, starting from the boundary of the domain to its center. Where the elements meet the unmeshed domain is called the front. Thus, the elements are generated till the front disappears.

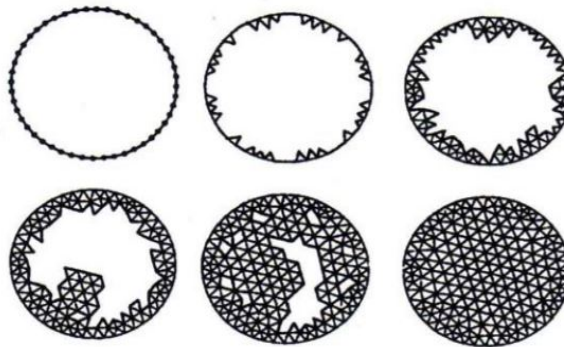


FIGURE 4.1: Advancing Front Method for mesh generation [16]

This method generates high-quality elements at the boundary. The method fails when the front collides with itself and ensuring element quality is difficult. These methods have been more successful in the fluid mechanics field where good elements are more necessary on the boundaries.

- *Quadtree/Octree algorithms*, which subdivide the domain by laying a structured background grid. Usually by warping the grid, element quality is ensured so no short edges are present. Quadtree algorithms, Figure 4.2, are suitable for 2D domains. Octree algorithms work on the same principle, but divide 3D domains.

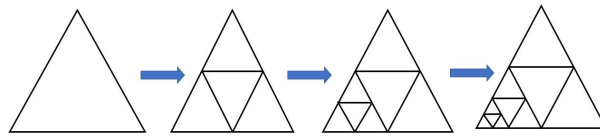


FIGURE 4.2: A graphical representation of the Quadtree Algorithm

Element quality at the interior is good but not on the boundaries. One disadvantage is the tendency of mesh edges to be aligned in a few preferred directions. Their speed, ease of parallelism and their robustness of meshing unclean CAD geometry are some of the advantages.

- *Delaunay refinement algorithms*, which construct meshes based on delaunay triangulation criterion. These are the most popular algorithms when it comes to unstructured mesh.

## 4.1 Delaunay Triangulation Criterion

The Delaunay triangulation criterion implies that no vertex of a triangulation lies in the interior of the any triangle's circumcircle, the unique circle that passes through each of a triangle's vertices. Correspondingly, in three-dimensions, no vertex is enclosed by any tetrahedron's circumsphere.

**Definition.** Let  $S$  be a set of points in the plane. A triangulation  $T$  can be said to follow the Delaunay criterion of  $S$  if for each edge  $e$  of  $T$ , there exists a circle  $C$  such that:

- the endpoints of  $e$  are on the circumference of  $C$ , and
- no other vertex of  $S$  is in the interior of  $C$ .

The Delaunay triangulation would be unique if no four points are co-cyclic. If co-cyclic points exist, all the triangulations would be Delaunay.[1]

Delaunay refinement algorithms start by constructing a triangulation that is delaunay. It is then refined by adding new vertices which eliminates very thin or large elements, always while maintaining the delaunay criterion. The delaunay triangulation influences the placement of new vertices so that no sliver elements or short edges are formed. In contrast to the front advancing algorithms, delaunay algorithms create high quality elements in the interior and worst elements on the domain boundary. Delaunay methods also assure a mathematical guarantee, they will always produce a valid mesh and in most cases, no sliver elements. [26]

Figure 4.3 shows an example of delaunay and not-delaunay triangulation.[17]

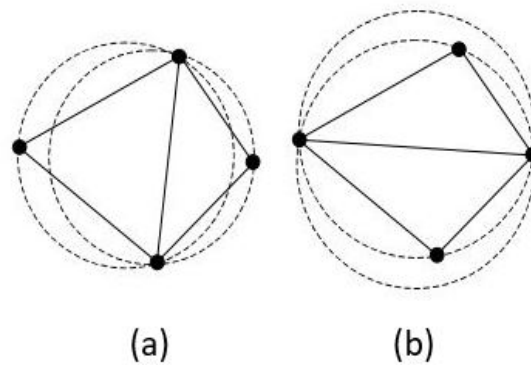


FIGURE 4.3: Delaunay triangulation in two-dimensions maintained in (a), not maintained in (b)

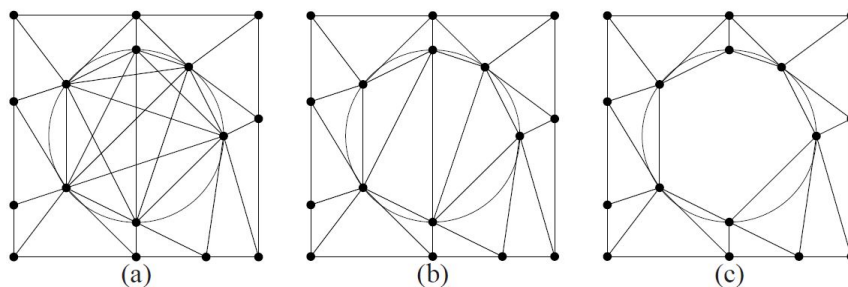


FIGURE 4.4: (a) creates all the delaunay simplices between the specified points, (b) chooses only the triangles and (c) excludes all the crossing Delaunay edges [26]

## 4.2 Algorithms

The first published algorithm implementing the Delaunay triangulation was in 1967 by Bernal and Finney. They developed the method and the program, which can also be termed as the brute force algorithm. It will test every possible tetrahedron to check which one satisfies the delaunay criterion. Thus, it takes  $O(n^5)$  time. In multi-dimensions, the time is  $O(n^{d+2})$ , where  $d$  is the number of dimensions.[30]

Besides this algorithm, the delaunay algorithms can be classically divided in three types:

- *Gift Wrapping* or incremental search, pivoting and graph traversal algorithms generate one Delaunay triangle at a time, all the while using the previous element as a seed for the next element. These algorithms are easily extended to higher dimensions but the bottleneck is identifying new triangles. Thus, the different algorithms are differentiated by the methods used for constructing triangles or the vertex search strategies.
- *Divide-and-Conquer*[25] algorithm is the first Delaunay triangulation algorithm to run in optimal time,  $O(n \log n)$ . It divides the set of points in two, divided by a line, and then both the halves are simultaneously triangulated finally merging them in one. This algorithm is fast but not enough in the three-dimensional world.
- *Incremental insertion* algorithms doesn't use a already present set of points to create the triangulation. Rather, it places vertices one by one, always maintaining the delaunay triangulation before a new vertex is inserted. The fastest 3-dimensional meshes are generated using this class of algorithms.

All the three algorithms can be extended on Constrained Delaunay triangulation, introduced in the next section. Gift wrapping and Divide-and-Conquer algorithms are complicated and thus difficult to implement. Thus, they are rarely used. The most commonly used algorithm is the Incremental insertion algorithm.

### 4.3 Constrained Delaunay Triangulations

A problem with Delaunay triangulation is the possibility of the triangulation not respecting the domain's boundary. An alternative to this problem is to use the Constrained Delaunay triangulation or CDT. A CDT is defined by the vertices and segments that mark the domain boundary. Every segment of the domain should be a segment of the CDT. It is not mandatory for the triangles defined to be delaunay. Instead they must satisfy the Constrained Delaunay criterion, which somewhat relaxes the empty circumsphere condition.

**Definition.** Let  $G$  be a planar graph which is a straight line. A Triangulation  $T$  would be a constrained Delaunay Triangulations of  $G$  if each edge of  $G$  is an edge of  $T$ . For each of remaining edges,  $e$  of  $T$ , there exists a circle  $C$  with the following properties:

- The endpoints of the edge  $e$  is on the boundary of  $G$ , and

- if any vertex  $v$  of  $G$  is inside  $C$ , then it should be invisible to at least one of the endpoints of  $e$  i.e. if a line is drawn from  $v$  to all the endpoints of  $e$ , then at least one line segment crosses an edge of  $G$ .

It can be concluded that if  $G$  has no edges, then a constrained Delaunay triangulation is the same as unconstrained Delaunay triangulation. Intuitively, both the triangulations are similar except that, the CDT ignores the portion of the circles where edges cut the circumcircle.[1]

CDT doesn't need extra vertices to be added to maintain the arbitrary segments. Another big advantage of CDT is that it inherits all the Delaunay triangulation advantages: it maximizes the minimum angle, minimizes the largest circumcircle, and minimizes the min-containment angle.



## Chapter 5

# Automation of Mesh generators

Designing components typically can take longer than the production and delivery of the component put together. The evolution of a new component is an iterative process. It involves several steps and calculations on the component to design it. Because it is a new product, several changes in the input or the design process steps can be expected. The loading conditions can change or the geometry is modified. With each small or big change, many changes follow in the whole design process.

Finite Element modeling is a standard approach used in the industry to evaluate new designs. It is a cheaper alternative to costly physical testing. A big advantage of finite element modeling is that the overall lead time in product and process design is reduced significantly. An important step in this modeling is discretizing the domain. Building the FEM models by discretizing the domains for analysis still requires a significant amount of the design engineer's time. Also, with each change, however big or small, a new FEM model has to be generated. This is considerable effort.

At the *Department for component design and manufacturing technologies, Institute for the structures and design, Deutsche Zentrum für Luft- und Raumfahrt e.V.(DLR)*, an automated process is being developed. This process automates the structural development of a new component, here a turbine blade assembly. Figure 5.1 indicates the flowchart for this process. The flowchart is an iterative procedure, which starts with an input CAD/parametric geometry and loads. Then, a structured mesh for the parametric geometry is created. The CAD geometry is the input to generate unstructured mesh. Afterwards, the model is built with the materials, boundary conditions, analysis options and pressure/temperature mapping. Next, the model is made ready for the solver(s) and the analysis performed. The results generated are used in two ways. They are used, naturally for the result visualisation. They are also instrumental in providing feedback to manipulate the geometry based on critical locations in the results. Every step is automated i.e. no user interaction is present. The user has to only input the original geometry and the loads. He/she will get the components' optimum behaviour out, like the eigen frequencies, stresses without

any intervention in between.

This thesis's aim is to establish the automation of the steps which are coloured blue in figure 5.1. This chapter establishes the creation of unstructured meshes from the CAD model. The mesh would be converted to a standard format too.

As concluded in Chapter 3, Quadratic Tetrahedrons are the best choice to capture the physical behaviour of the component. The algorithms discussed in Chapter 4 for generating Tetrahedrons would be used in this chapter to create the unstructured mesh. In the first part, section 5.1, the test and evaluation of the existing automated meshing softwares for tetrahedrons is described. The second part, section 5.2, develops on an existing software to create an automated program suited to the geometry on hand, a turbine blade assembly.

## 5.1 Test and Evaluation

An automatic mesh generator can be defined as a unit that uses some boundary data as an input, then creates the interior and external nodes and elements automatically based on the input. Since the mesh generator assumes little or no information on the structure of the domain, almost any structure can be modelled with these class of mesh generators[11]. The field of automatic mesh generation is a relatively new field. Tremendous advances have been made since its inception. Generation of tetrahedrons is easier to automate than generation of hexahedrons.

The first step is to explore the existing automatic mesh generators. There are several mesh generators already present to be examined. They can be open source or closed/proprietary generators. A open-source mesh generator is a software, which is generally free of cost, in which the copyright owner lets the user study, change and distribute the software to anyone. In contrast, to use a closed/proprietary mesh generator, a user must buy a license to use the software and cannot freely distribute the software. The intellectual and copyright rights lie with the software's publisher[24].

Other advantages of using a open-source mesh generator is that it is easily available and the source code is accessible. It is believed to be better-designed since it typically has many independent programmers testing and fixing the software. They can be efficiently used to develop interfaces. Disadvantages of such a software can be that the development process might not be well defined or the documentation is ignored.

On the other hand, proprietary softwares come with full documentation, support, training and guidance. A well defined development and upgradation plan for

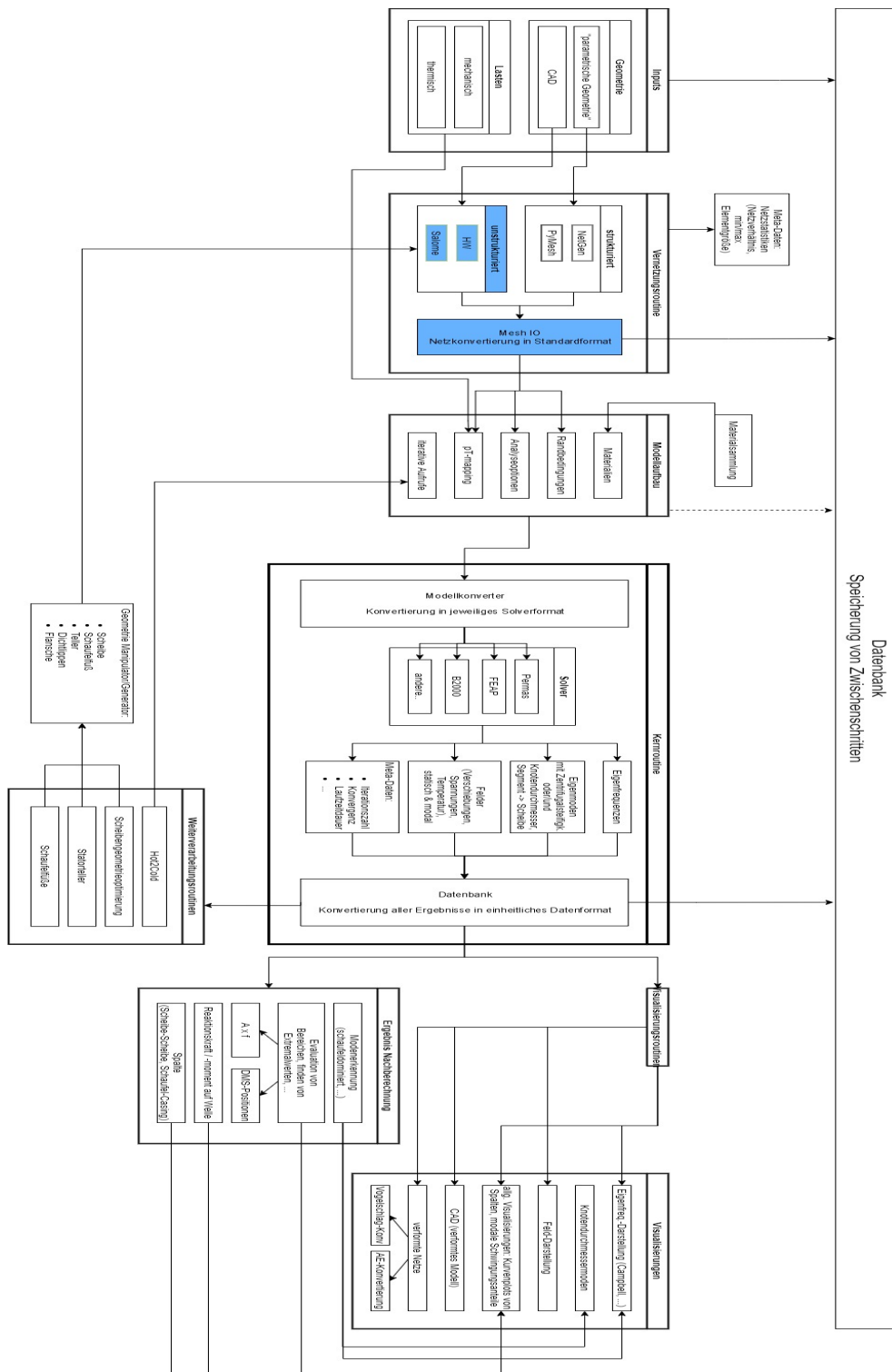


FIGURE 5.1: Automation of the structural mechanical design process for a blade assembly[4]

the software is available. Aside from the fact that a license is needed, another disadvantage of a proprietary mesh generator could be to not have the ability to tailor the source code to the user's requirement.

To explore the mesh generation software market, several mesh generators are assessed. The assessment is done in two steps:

- A **preliminary** assessment, where each of the mesh generators are gauged for their ease of installation, documentation availability etc. For the commercial softwares, an extra criteria is if a test/demo version is accessible.
- A **secondary** assessment where two open-source and proprietary mesh generators each are studied in more detail. They are selected based on the initial assessment of all the softwares.

In the initial assessment, some of the mesh generation softwares[22] surveyed are:

#### 1. Open-Source Mesh generators

- Gmsh: A Delaunay algorithm based mesh generator, generates adapted meshes. The software is easy to download and compile. There are several options to create meshes.
- DeIPSC: It can produce a weighted Delaunay algorithm based mesh. It guarantees all triangles and tetrahedrons have bounded radius-edge ratio. The software isn't easy to build.
- SALOME: It is a free software providing a pre- and post-processing solutions. It has several meshing algorithms to mesh the available lines/surfaces/solids. The mesh generator is easy to download and use.
- Tetgen: It generates exact Delaunay tetrahedrons. This mesh generator requires a two-dimensional mesh to be created beforehand.
- LBIE-Mesher: Level Set Boundary Interior and Exterior mesher. It is an unstable software, crashing easily.
- MeshGenC++: A software package for generating unstructured mesh. It is easy to download but cumbersome to install and build.
- DistMesh: A MATLAB code to generate unstructured mesh. It is easy to compile but works well only for simple geometries.
- FELICITY: Generation of unstructured mesh with angle bounds is guaranteed. It is easier to use as MATLAB is already available. But, a specific version of MATLAB is required and many other plug-ins, making it difficult to compile.

- CGAL mesh generation: Various packages for triangle and tetrahedron generation are available. It is easy to download but difficult to compile.

## 2. Closed/Proprietary Mesh generators

- BOXERmesh: A mesh generation software to generate tetrahedrons. It is better suited to CFD applications. The test version for this software is not available.
- GID: Structured and unstructured mesh can be generated using this generator. This mesher works well when a bigger element size is used. But, switching to a finer element size presents a lot of problems while meshing.
- CM2MeshTools: A Delaunay mesher for triangles, quadrangle and tetrahedrons. A trial version is available. The compilation is tricky for this software.
- Castnet: An automated hybrid mesher for all kinds of three-dimensional elements. A test version is not easily available.
- NISA-Display IV: An automatic and mapped mesh generation for structured and unstructured meshes. Trial version is easily installed. But, the mesh is not easy to generate.
- ENNOVA: A structured and unstructured surface and volume meshing along with hybrid and boundary layer meshing using prism. A trial version is obtained and tested. This mesh generator has a lot of potential.
- FEMAP: A finite element preprocessor for generating 2D and 3D meshes. A trial version is easily available. There are many options available to create a good mesh and is stable.
- CADfix: A volume and surface mesh generator. It is more suited to medical object technology. No access to a test version is available.
- DIANA(FEMGV): A general pre-and post-processing software suitable for use with FEA, CFD and Finite difference method. Accessing a test version is not possible.

At the end of this initial assessment, the mesh generation softwares that exhibit the most potential are chosen. The generators which are easy to download along with their ease to install and compile are preferred. Stable programs are desirable. A mesh generation program that has several options for creation of meshes is also a better choice.

An additional desirable feature in a mesh generation software is that it lends itself well to being programmed i.e. API. An application programming interface (API) is a computing interface to a software, that defines how other interfaces can

access or use it. Because in the whole automated process (in figure 5.1), the automated meshing module is called by its upstream and downstream interfaces, an API is an important feature in the mesh generator. Hence, the thesis at hand requires customization of the mesh generator to suit the geometries of interest. It is important that the mesh generator has an API that is easy to access and program.

A secondary and final assessment is done on these mesh generators. The selected mesh generators are evaluated in detail to assess with these features like their ease to download and compile along with being stable. A test geometry is used to complete this assessment.

### 5.1.1 Geometry for secondary assessment

The selected mesh generators are subjected to closer scrutiny using a test geometry. This geometry depicts the actual geometry to some degree. It is a rotating component of an aircraft engine. Specifically, the blade of a turbine. The geometry of a blade is complicated as it has an airfoil shape with lots of small and long holes, which are meant for cooling of the blade evident in figure 5.2.

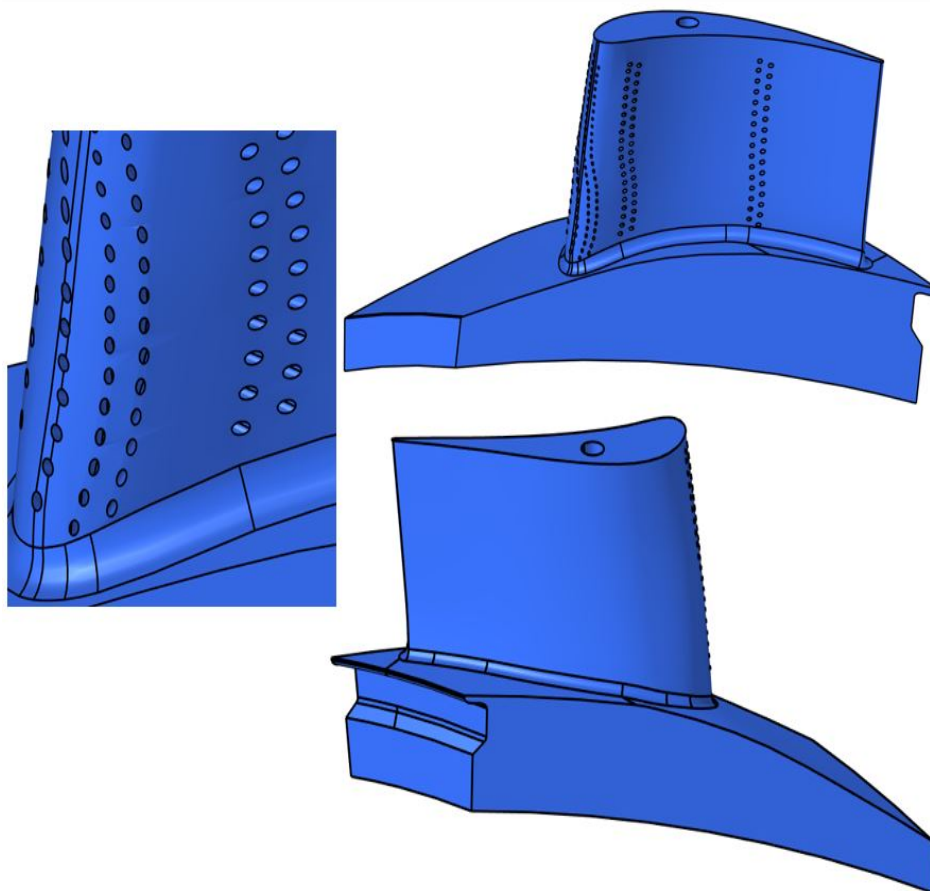


FIGURE 5.2: Test Geometry used to assess the mesh generators[18]

Figure 5.3 depicts the small areas that could prove to be difficult to mesh due to their complexity. The cooling holes are small in diameter and long in length. They are illustrated by hiding the top aerodynamic surfaces of the blade.

Another test geometry is used which is also a model of a turbine blade with similar complex details. But, the CAD model is not created cleanly. An example of unclean geometry can be geometries with mating surfaces having gaps between the surfaces. Or, the mating surfaces are overlapping.

So, this geometry tests the mesh generator's ability to deal with bad input geometry. It is an added advantage if the generator has options to deal with unclean geometry before it meshes the geometry. This geometry cannot be depicted in this report due to it being protected information.

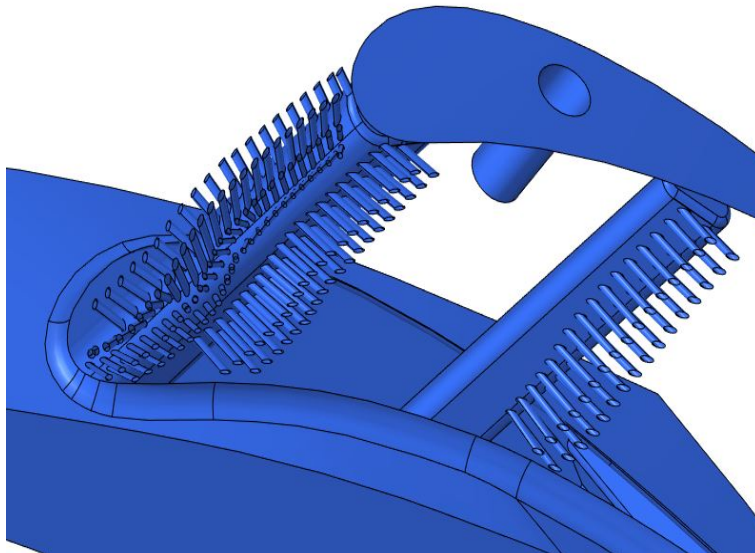


FIGURE 5.3: Small details of the turbine blade[18]

### 5.1.2 Assessment procedure

The secondary assessment is done for the selected mesh generators using the steps illustrated in figure 5.4. The typical steps of generating a mesh are followed: installation/compilation of the mesh generator, import the geometry, selection of meshing options/algorithms and finally, meshing. The mesh generator is evaluated meticulously for each of the steps.

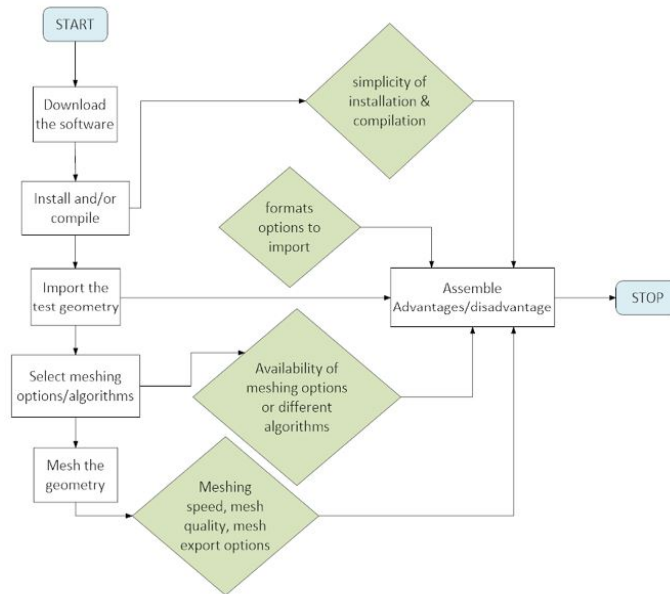


FIGURE 5.4: Steps followed to perform the secondary assessment for mesh generators

### 5.1.3 Secondary assessment: Open-source Mesh generation

In the preliminary assessment of the open-source generators, the most important factors emerged as ease of installation, stability and the available meshing options. The ease of using the API is also evaluated. Two mesh generators, Gmsh and Salome fared well on these criteria in the initial assessment and thus, are selected to evaluate further.

#### Gmsh

*Gmsh is an open-source three-dimensional finite element grid generator with a build-in CAD engine and post-processor. Its design goal is to provide a fast, light and user-friendly meshing tool with parametric input and advanced visualization capabilities.*[7]

Installation of Gmsh is simple and straightforward. Compilation is uncomplicated as the download package had an 'exe' file incorporated which can be simply called to launch the application. The GUI of Gmsh is simple but the options available to generate meshes are limited.

The test geometry in figure 5.2 is imported to Gmsh to mesh. The delaunay criterion algorithm is used to create the mesh for the test geometry. The element size suggested automatically by Gmsh is used. The mesh is created in an bottom-to-top fashion i.e. first the edges are discretized, then the faces are meshed and finally the 3D elements are created. With this algorithm, the produced mesh captures the small



details of the test geometry to a satisfactory level. But, many tetrahedron elements fail on the aspect ratio (equal to 10) quality check. The mesh of the test geometry can be referred in figure 5.5.

A particular important disadvantage is that program is unstable and crashes easily. For e.g. if a finer element size is chosen, Gmsh crashes. Gmsh API programming is not simple to use. This is a clear disadvantage as API is a crucial feature to automate the meshing process.

Additionally, Gmsh is also used to mesh the second test geometry with the unclean surfaces. Gmsh isn't able to work with the bad geometry and crashed promptly.

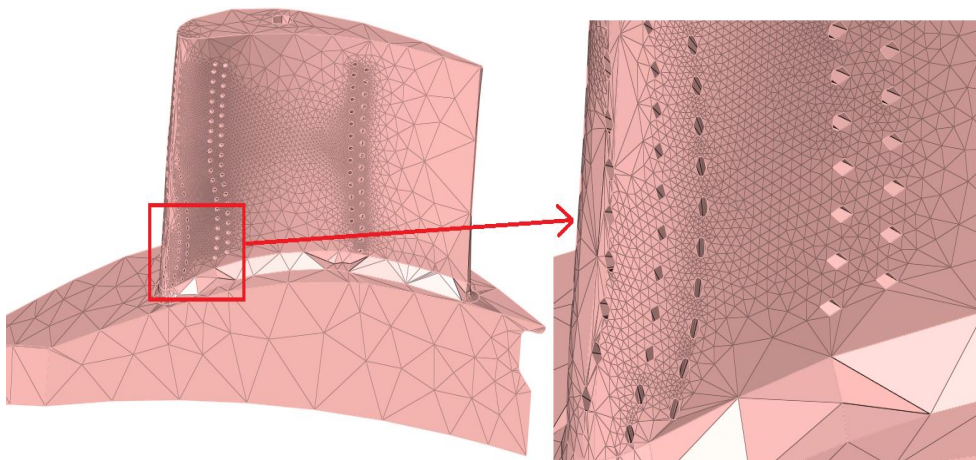


FIGURE 5.5: Test Geometry mesh using **Gmsh**, poor aspect ratio and poor representation of features

## SALOME

*SALOME is an open-source software that provides a generic Pre- and Post-Processing platform for numerical simulation. It is based on an open and flexible architecture made of reusable components.[19]*

The download package for SALOME is easily downloaded and could be used straightaway without any need of installation or compilation. A 'bat' file is already present in the package which directly launches the SALOME application.

SALOME has a very comprehensive GUI. There are lots of modules on offer. The geometry module has the ability to create and modify geometries. The mesh module has several options to create a mesh based on different algorithms. Apart from the pre-processor modules, there are various modules for post-processing too.

The geometry module of SALOME is used to import the test geometry. Then, there are various options to make groups in the geometry. At this point, no groups

are created for the test geometry. This is because only the meshing capability is evaluated. Next, the mesh module is activated. The NETGEN algorithm is selected to create the mesh for the test geometry. This algorithm incorporates all the algorithms for discretizing edges, faces and solids together. Using this algorithm requires very little interaction from the user in the mesh generation process. The element size suggested by the program is used. The generated mesh captures all the features of the blade adequately. The mesh, figure 5.6 also fares well on the aspect ratio quality check.

SALOME is quite stable and has a superior performance as compared to Gmsh. The API of SALOME is through python. Python is a high-level, general-purpose programming language. It is simple to use to automate the meshing module. This is an immense advantage.

SALOME is also tested for the poorly generated test geometry. It can handle geometries which aren't very unclean. But, a geometry with several problematic surfaces doesn't fare well with SALOME. Thus, it might need an additional step to cleanup the geometry beforehand for poor geometries.

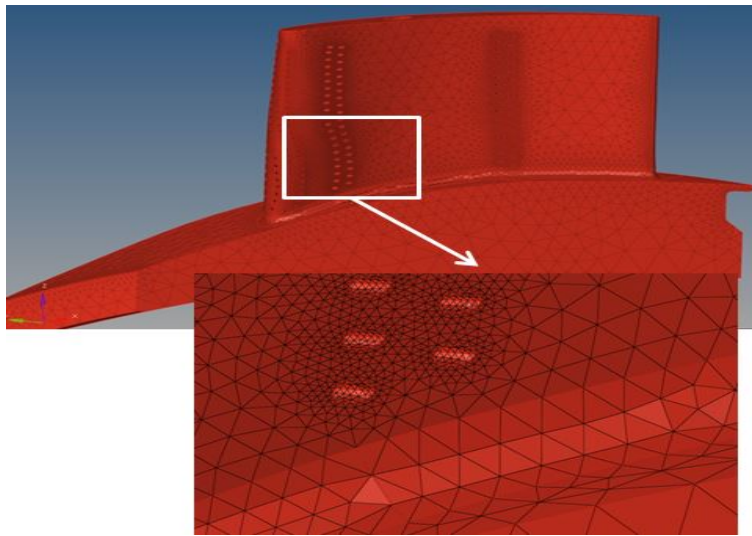


FIGURE 5.6: Test Geometry mesh using **SALOME**, stable program with a mesh that captures the details accurately

#### 5.1.4 Secondary assessment: Proprietary Mesh generation

In contrast to Open-source mesh generation softwares, proprietary mesh generators are easier to download and install/compile. Documentation and support for the proprietary generators are abundant and easily available. Here too, two mesh generation programs are chosen: FEMAP and ENNOVA. They fared well on the preliminary assessment.

## FEMAP

*FEMAP is an advanced engineering simulation application for creating, editing and importing, re-using mesh-centric finite element analysis models of complex products or systems. It can be used to model components, assemblies or systems and to determine the behavioral response for a given operating environment[27]. FEMAP is a pre- and post-processing software by Siemens.*

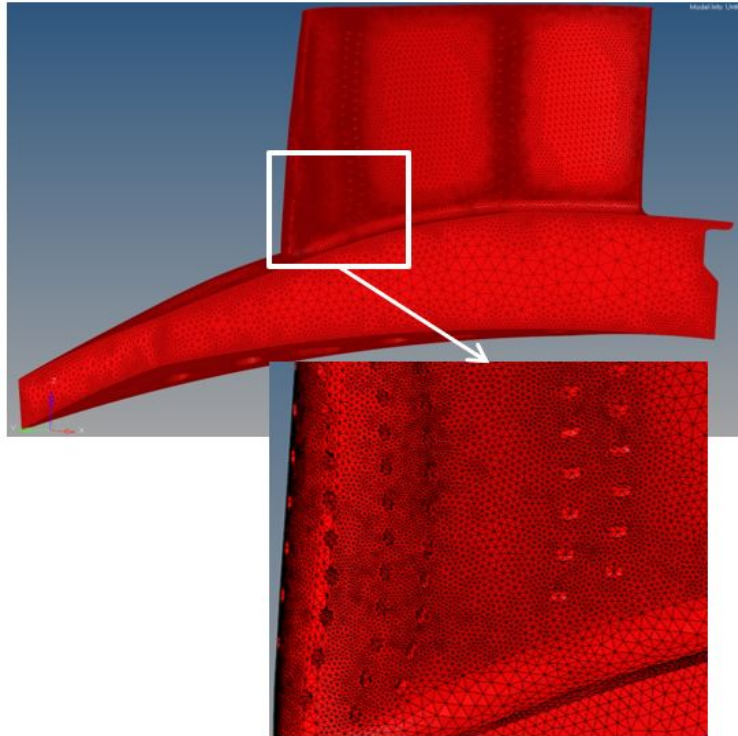


FIGURE 5.7: Test Geometry mesh using Siemens FEMAP, similar details as SALOME but lower node count

A test version of FEMAP is easily available and downloadable. It comes with easy to follow instructions to install. As FEMAP is a software from SIEMENS, it can be advantageous to create interfaces with CAD geometry creation from SIEMENS (NX) or with SIEMENS' FE solvers.

The GUI of FEMAP is very user friendly. There are several options to create/modify geometries. The creation of meshes is a smooth process with numerous meshing options. FEMAP can also be used to post-process analysis results.

The geometry is imported in FEMAP. Here too, there are several options to create groups of surfaces or edges to have better control on the meshing process. No groups are created to generate the test mesh. Basic settings, suggested by FEMAP are used for the mesh generation. The quality of the mesh generated through FEMAP is similar to that through SALOME. But, the size of the model i.e. number of nodes,

generated is smaller in FEMAP. The number of elements which have an aspect ratio less than 10 is also similar in both FEMAP and SALOME. FEMAP has a built in API. It can be easily used to call FEMAP from within FEMAP or other external programs.

FEMAP is a stable software and fares well with the test geometry. Some stability issues are observed when the unclean test geometry is used to generate meshes. It has limited capability to deal with bad geometries and crashes easily. Another drawback with FEMAP is that as it is a paid commercial software and hence, the license has to be bought.

## ENNOVA

*ENNOVA meshing software provides a scalable client / server architecture solution for CFD and FEA meshing. Using a novel Hands Free approach. ENNOVA can generate a wide range of meshes from fully structured to unstructured, with tetra, prism, and polygonal cell types. Hybrid meshes with regions of structured and unstructured blocks can be constructed with minimal user input, which is ideally suited for rapid design cycles and optimization [5].*

The mesh generator ENNOVA'S test version is straightforward to download and install. An academic license is shared by ENNOVA. It's documentation is limited and hence poses some challenges.

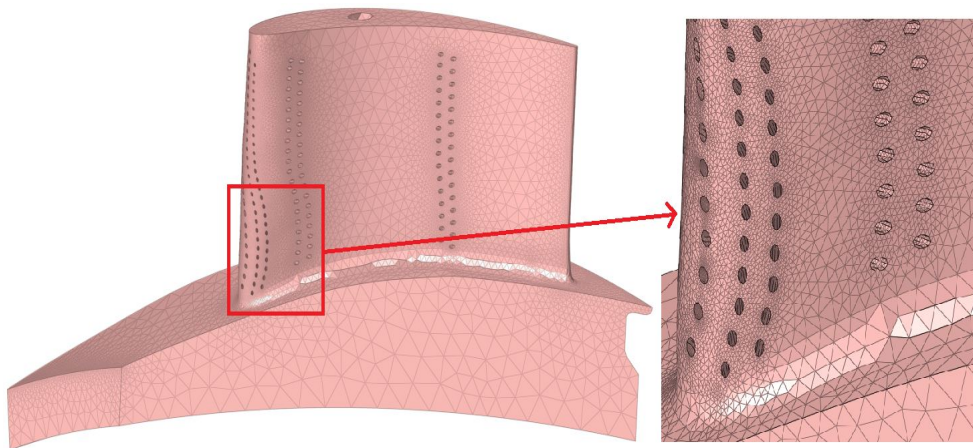


FIGURE 5.8: Test Geometry mesh using ENNOVA, captures the details properly

The test geometry is imported in ENNOVA. Unlike SALOME and FEMAP, there aren't numerous options to create meshes. ENNOVA can suggest mesh size settings based on the geometry and they are used to create the mesh for the test geometry. Topology based meshing is used, which captures all the essentials of the geometry. A satisfactory test mesh is created with very less effort, figure 5.8. The creation of

the mesh is quite fast with ENNOVA. ENNOVA has a built-in API but due to limited documentation it is unclear how to call it from an external program.

On the stability front, ENNOVA is right at the lead. It is also tested with the unclean test geometry. ENNOVA has a powerful geometry cleanup option which can deal with bad geometries. It can then easily create the mesh.

### 5.1.5 Assessment matrix

The secondary and final assessment preceded by the preliminary assessment led to four different mesh generation softwares. There are two open-source and proprietary mesh generators each which are of the most interest because of the advantages they offer. The chosen four programs are used to generate the mesh for the test geometry, as explained in the last section. Apart from the factors already explained like the downloading/installation ease, program stability, API there are a few other criteria to be studied. For e.g. meshing speed, automatic creation of groups etc. The four mesh generators only are judged on these criteria. The criteria of most importance are:

- **Geometry handling:** The ability of the mesh generation program to handle unclean geometries. It is expected that small problems in the geometry be dealt successfully by the program.
- **Program Stability:** It is important that the program is stable i.e. doesn't crash or becomes unresponsive when something demanding is expected. E.g. when a too fine mesh size is used to generate the mesh.
- **Mesh Control:** Several meshing algorithms options availability in a mesh generation software is a useful advantage. Different algorithms ensure better control on the creation of mesh. A location on the geometry could be meshed with a finer setting with a different algorithm, another could use different settings to generate the mesh.
- **Meshing speed:** It is imperative that the mesh generation process is finished in a reasonable amount of time. For the same quality, if one mesh generator is faster than the other, it would benefit to choose the first one.
- **FE-Model size:** The size of the FE-model ( i.e. the total number of elements and nodes generated) created with the same element quality directly affects the time required to compute the unknown quantity by solving the FEM equations.
- **Bad elements:** The shape and quality of an element has a large effect on the accuracy of the analysis results. Hence, certain quality checks are required to

check the shape and quality. E.g. the aspect ratio of element or the skewness or the jacobian ratio.

- Group creation: Creation of groups or sets, whether from elements or nodes aids in boundary condition applications. The creation of groups also helps in making a part of the geometry finer than the global mesh.
- API: How effortless is the API of the mesh generation program is another important aspect to measure.
- Cost: The cost of buying a software and licenses also weighs in. The inclination is towards open-source but only if they satisfy the required criteria.

Table 5.1 discusses these criteria and how the mesh generators perform against them.

TABLE 5.1: Assessment Matrix of Open-source and Proprietary meshing Programs(1-best score & 4-worst score)

| Criteria                 | Gmsh | SALOME | FEMAP | ENNOVA |
|--------------------------|------|--------|-------|--------|
| Geometry Handling        | 4    | 2      | 2     | 1      |
| Program Stability        | 4    | 2      | 2     | 1      |
| Mesh Control             | 4    | 1      | 1     | 3      |
| Meshing Speed            | 3    | 2      | 1     | 1      |
| FE Model size            | 4    | 3      | 1     | 1      |
| Bad elements             | 4    | 2      | 3     | 1      |
| Automatic group creation | 4    | 1      | 2     | 2      |
| API                      | 4    | 1      | 1     | 4      |
| Cost                     | 1    | 1      | 4     | 4      |

Gmsh doesn't meet most of the evaluation criteria and isn't a good fit to be used further. On the other hand, FEMAP is a good fit from the perspective of mesh control, meshing speed, FE model size and API use. ENNOVA performs well on many criteria, the handling of unclean geometry, program stability, meshing speed, FE model size and least bad elements. The biggest drawback of ENNOVA is its API. No documentation or support is found to call the API from external programs. SALOME fares the best in the options for mesh control, automatic group creation, API. As it is an open-source program, it is also a positive on the cost aspect. It is also never the worst in any of the criteria.

Based on these takeaways, SALOME is the logical choice for further development on the meshing automation process. In the next section, development on a SALOME interface to build the automated meshing module of figure 5.1 is established.

## 5.2 Development of SALOME Interface

### 5.2.1 Introduction to SALOME

SALOME has two modes to work in: GUI mode and Batch mode. The GUI mode accesses the functionalities of SALOME via SALOME session server. It has a desktop window, menus, dialogs and Viewers & object browser. The Batch mode accesses the SALOME functionalities from the command terminal without GUI. The batch mode has a python console for directing accessing SALOME. Python scripts can be run through this mode. SALOME can be used on windows and Linux both. Though, the program was originally aimed at Linux systems and later, a windows version was released. The source code is available to any programmer who wants to use/develop it. SALOME is distributed under a GNU General public license. It means that everyone is permitted to copy and distribute copies of the software, but changing it is not allowed.

SALOME has several built-in modules which can be used for various pre/post analysis processes. One of them is the mesh module. The mesh module can[19]:

- create meshes in different ways: meshing a geometry, mesh editing or building 3D mesh from a 2D mesh
- import/export meshes
- build node/elements groups automatically based on criterion
- offer different algorithms and hypotheses for meshing
- generate meshes using Python(API) scripting

Along with the mesh module, SALOME also contains a computer-aided design (CAD) module that can read several geometry input files, for e.g. step format. An instrumental tool in the CAD module is the creation of groups based on filtering. Filtering can be used to group Sub-shapes together automatically. The sub-shapes' property is instrumental in this automatic selection. The sub-shapes' property which is calculated according to the topology can be extracted for each sub-shape:

- length for edges/wires
- area for faces/shells
- volume for solids

The property can be used to define the range of edges/faces or solids to be grouped together in a *geometry set*. This geometry set can be later used to create finer mesh areas or to create boundary conditions.

## 5.2.2 Meshing Algorithms and Hypotheses

Mesh generation is performed in a bottoms-up fashion, nodes on vertices are created first, then edges are discretized using the vertices' nodes, these nodes on the edges are used to segment the faces and finally the faces' nodes create the solid mesh. This approach ensures the mesh is conforming.

SALOME has several algorithms available to generate meshes based on element types, different techniques or dimensions. As established in Chapter 3, quadratic tetrahedrons are the preferred element type. Hence, to build the tetrahedrons, first a two-dimensional mesh is created which is made of triangles. Before that, the edges and vertices have to be discretized. Thus, algorithms and hypotheses for each of them must be selected.

Hence, in this section, algorithms in SALOME relevant to triangles and tetrahedrons generation are addressed. The generic definitions of triangle and tetrahedron generation algorithms can be found in chapter 4. A mesh generation algorithm is a process or set of rules to follow to generate a mesh. Hypotheses in SALOME are the boundary conditions for the meshing algorithms. They manage the level of detail on the mesh by specifying the mesh parameters.

### 1. Meshing 1D entities(edges)

#### (a) One-dimensional Mesh Generation Algorithms[20]

- *Wire Discretization*: This algorithm splits an edge into a number of mesh segments based on a 1D hypothesis.
- *Composite Side Discretization*: A whole face of the geometry can be discretized together even if it is made of several edges provided the edges form a C1 curve.

#### (b) One-dimensional Mesh Generation Hypotheses[20]

- *Number of Segments* hypothesis estimates edges by a defined number of mesh segments. Different node distributions can be defined like an equidistant (all the segments will have the same length) or a analytical function (a formula that will rule the change of length).
- *Local Length* hypothesis sets the length of segments to discretize a edge. It also requires a precision parameter, which rounds up the number of segments. The number of segments is calculated by dividing the edge length by the defined length of segment. The precision parameter determines if the number of segments goes to the lower or upper integer.
- *Arithmetic & Geometric Progression* hypotheses divide an edge into segments using Arithmetic or Geometric progressions.



- *Start & end length* hypothesis makes segments on an edge which has the first and last segment of the specified length. The segments in between change based on an automatically chosen geometric progression.
- *Adaptive* splits an edge into segments based on the curvature of the edges and faces. The maximum, minimum size and the deflection of a segment from the curvature as user-input limit the hypothesis.
- The *Deflection* hypothesis divides curvilinear edges based only on chord error or the deflection from the curve. The more curved the surface is the shorter the elements will be. Hence, creating a finer mesh at curved locations.

## 2. Meshing 2D entities(faces)[20]

### (a) Two-dimensional Triangle generation algorithms

- *Triangle:Mefisto* splits faces into triangular elements. It is only a 2D algorithm and needs a 1D algorithm additionally.
- *NETGEN 1D-2D* combines the generation algorithms of 1D and 2D elements in one algorithm. The surface mesh generation algorithm is based on the advancing front method algorithm class. Mesh creation based on the delaunay criterion can also be selected instead of the advancing front method.[23]
- *NETGEN 2D* defines only the 2D discretization into triangles and is similar to the NETGEN 1D-2D in the surface mesh creation. it requires a 1D algorithm additionally to work.

### (b) Two-dimensional Triangle generation Hypotheses

- *Max Element Area* hypothesis limits the maximum area of each 2D element.
- *Length from Edges* hypothesis defines the maximum linear size of mesh faces. An average length of the mesh faces approximates the meshed face boundary.
- *NETGEN 2D Parameters* hypothesis defines the maximum and minimum element size allowed. It also controls many features, like the fineness of the mesh, or which algorithm to use: Delaunay criterion or Advancing Front method. The input to control mesh size based on the curvature of the geometry can also be chosen here. [21]

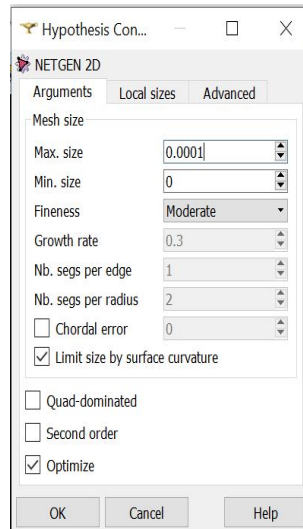


FIGURE 5.9: NETGEN 2D Hypothesis

### 3. Meshing 3D entities(solids)

#### (a) Three-dimensional Tetrahedron generation Algorithms

- *NETGEN 1D-2D-3D* algorithm doesn't require any lower level algorithms and hypotheses to be defined. Based on the hypothesis chosen, the mesh can be generated using Delaunay criterion or the advancing front method.
- *NETGEN 3D* algorithm is similar to the *NETGEN 1D-2D-3D* algorithm but here there is a flexibility to use different 2D and 1D mesh generation algorithms and hypotheses.

#### (b) Three-dimensional Tetrahedron generation Hypotheses

- *Max Element Volume* hypothesis limits the maximum volume a 3D mesh element can have.
- *NETGEN 3D Parameters* hypothesis presents several controls on creating a mesh. Figure 5.10 depicts these controls. The max and min size of the elements can be defined along with the fineness of the mesh. The fineness can be system defined or user defined, giving even more control. It can be chosen if the delaunay algorithm or the advancing front algorithm will be used to generate the mesh.

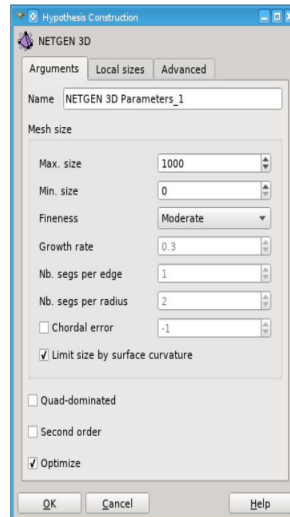


FIGURE 5.10: NETGEN 3D Hypothesis

### 5.2.3 Target Geometry

The aim of the automated mesh generation process is to automate the meshing of a high pressure turbine (HPT) blade and disk with a complex cooling circuit. The actual assembly is a protected geometry and its images can't be shared.

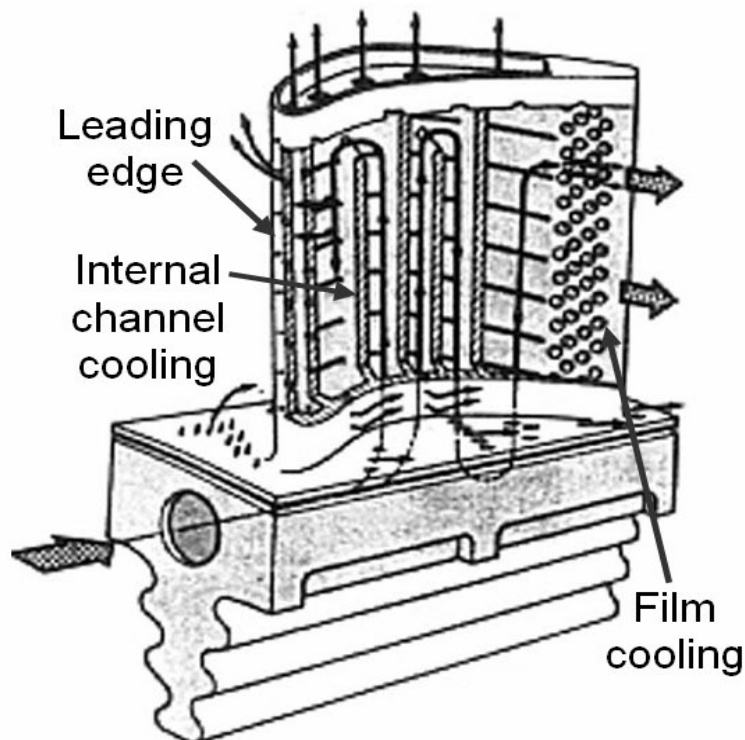


FIGURE 5.11: A turbine blade with a complex cooling circuit[8]

Instead, figure 5.11 depicts a similar turbine blade which is free to share. The

complex geometry for the cooling circuit can be noted. Figure 5.12 illustrates a typical turbine blade and disk assembly.

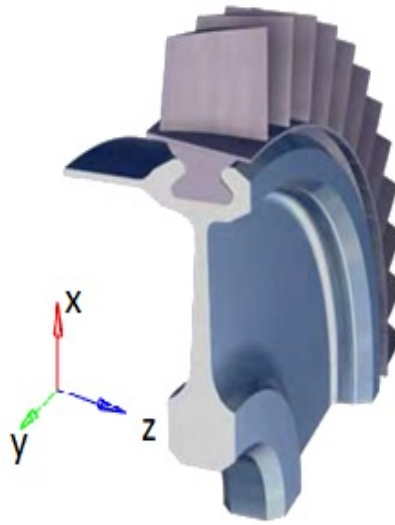


FIGURE 5.12: A typical blade and disk assembly[3]

The mesh produced through this task will be used in a quasi static structural mechanical analysis. The dimensions of the actual blade and disk assembly are:

- Length  $x = 305$  mm
- Length  $y = 39$  mm
- Length  $z = 123$  mm
- Mass = 1.1 Kg

Some best practices in creating the mesh are already laid out by the HPT component team. These are shown using the test geometry (from figure 5.2):

- Maximum element size = 1 mm.
- Thin walls must have 2 elements in thickness.
- Fillets, such as in figure 5.13 must have 8 or more elements

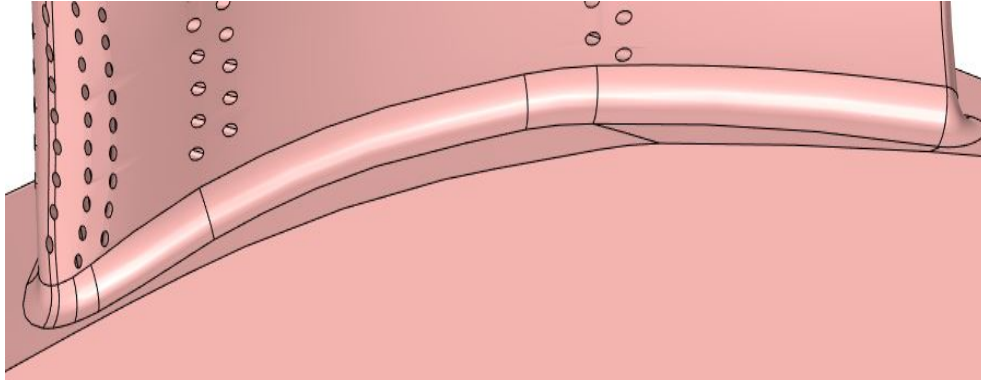


FIGURE 5.13: Mesh requirement on fillets

- Very small curves such as in figure 5.14 should have 1 or more elements in the direction of the biggest curvature.

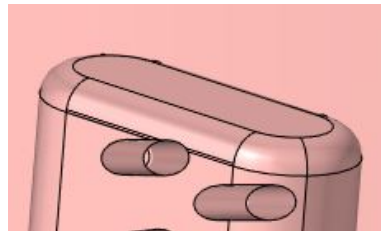


FIGURE 5.14: Mesh requirement on very small curves

- Cooling air holes such as in figure 5.15 should have 8 or more elements on the circumference.

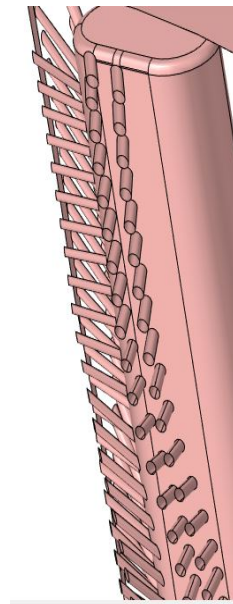


FIGURE 5.15: Mesh requirement on cooling air holes

Apart from these requirements on the blade, there are some requirements on the turbine disk too. The element size on the disk should be smaller than 2 mm. The

surfaces coming in contact with the blade should have 6 or more elements in the short edge direction.

### 5.2.4 Automation Process

The algorithms and hypotheses offered by SALOME are now identified. The geometry in figure 5.2 is used extensively to gain an understanding of SALOME. A typical meshing job in SALOME is shown in the flowchart in figure 5.16. The first step is to activate the geometry module of SALOME. Here, either a geometry can be built or one imported.

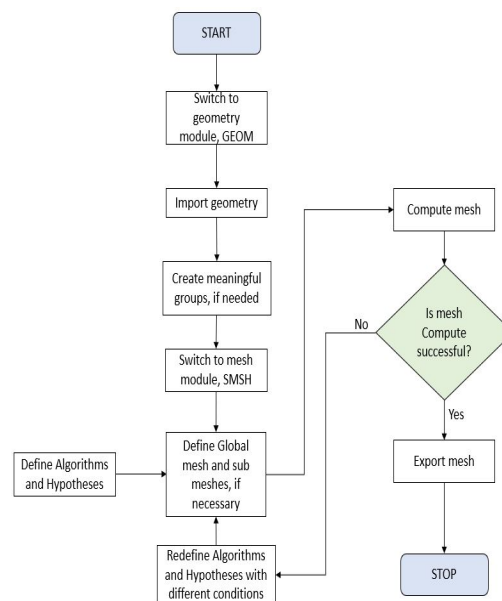


FIGURE 5.16: Flowchart of a typical mesh creation job using an imported geometry file

Several options are at disposal to create a geometry. As the geometry of interest is already present, the import geometry option is more of interest. A few file formats can be read, *STEP* (Standard for the Exchange of Product Data) is a common file. A step file is a 3D model file formatted in STEP, an ISO standard exchange format. Also available are *BREP*, *STL*, *VTK* or *IGES*.

Once, the geometry is input, it is checked for any issues. SALOME can aid in correcting small problems in the geometry. It can for e.g. sew small gaps in mating surfaces. Or, it can help in defeaturing the geometry by using suppress options.

Next, groups are built on the geometry. Groups can be built using vertices, lines, faces or volumes. The IDs of these entities can be used to create groups manually. In the GUI, it is also possible to select manually the entities using the mouse. A very useful way to create groups is by using the parameters of the entities. SALOME can

be programmed to automatically select only the entities within a parameter range. This feature is very convenient in creating varying element sizes based on the entity size.

In the meshing module, a global mesh must be created if a 3D mesh for the whole component is desired. Algorithm(s) must be selected defining the 1D, 2D and 3D mesh generation. Supporting hypotheses should be built too. If at some location, local refinement is required, a submesh can be defined. A submesh can be defined by manually selecting an entity or on a group. Here too, supporting Algorithms and Hypotheses are required.

After the definition of the global mesh and submesh(es), the mesh is computed i.e. the mesh is generated. If the mesh is computed, it can be exported in a few different formats like the *UNV* format. Other mesh export formats are *MED*, *STL* or *CGNS*. But, if the mesh computation is unsuccessful, an error message is generated. That should be corrected by reassigning algorithms and hypotheses. The mesh has to be recomputed and checked again for success.

This flowchart is now applied to the HPT blade and disk assembly explained in section 5.2.3 taking care to meet the mesh requirements set by the component team. All the steps explained in figure 5.16 can be achieved using the GUI or TUI mode of SALOME. TUI is also known as the python interface of SALOME. The python interface module of SALOME has several library commands defined which can be used to write python scripts. So, the next step is to understand the TUI commands available in SALOME. Along with that, how to work in python with SALOME is also important to study. The programming can be divided into two modules:

- **GEOM:** This module deals with operations related to the CAD modeling. The HPT blade and disk is available as a STEP file and is imported in the geom module using the `'geompy.ImportSTEP'` command. Geompy is a new instance of the GEOM module and `ImportSTEP` imports the step file. Then, the geometry is divided into two solids, the blade and the disk. The blade is more complex than the disk and hence, they should be treated in dissimilar fashion. The blade is then divided into face groups and 1 edge group. The edge group is for small edges, smaller than  $1e-4$ . It is evident from figure 5.17 depicting the distribution of the faces' area, that the most of the faces have very small areas. Hence, the faces are divided using a geometric progression,  $ar^n$ . 'a' is the first term in the progression and is equal to the minimum face area in the blade. The number of terms required is 'n'. It is chosen in a way that each and every face is accounted for in a group.

The group of the solid disk is kept as is and no more surface/edge groups are created. This is so because the disk is not a complicated geometry and thus, it is easier to generate its mesh. The mesh requirements on the disk are not

as stringent as the ones on the blade. Figure 5.18 depicts this workflow in the geometry, GEOM module program.

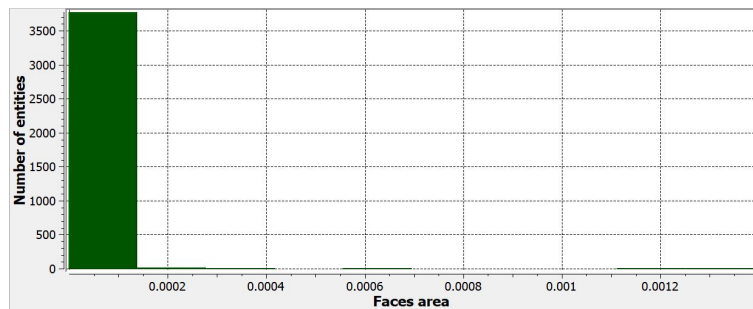


FIGURE 5.17: Distribution of HPT blade faces' area

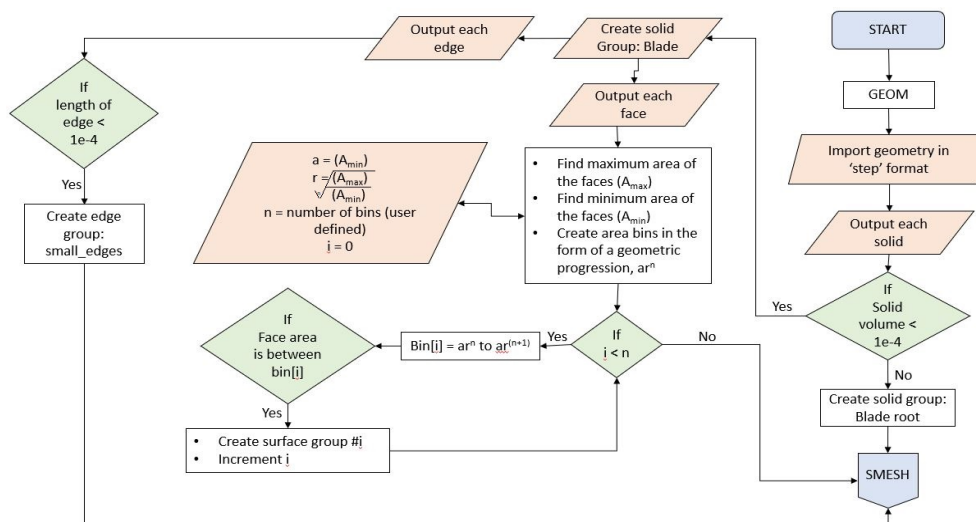


FIGURE 5.18: Flowchart of the processes in the geometry module on the HPT blade and disk assembly

- **SMESH:** All the processes related to meshes are included in this module. To define global meshes, *smesh.Mesh* is the command. The Blade and Disk global meshes are created with this command. The blade global mesh is assigned the NETGEN-1D-2D-3D algorithm. NETGEN 3D Parameters is used as the hypothesis with following values:

- Maximum element size = 0.001
- Minimum element size = 0
- Optimize the mesh
- Create a moderate mesh (Growth rate = 0.3, Segments per Edge = 1, segments per radius = 2)
- Limit the element size by the curvature



The disk global mesh is also assigned a similar algorithm and hypothesis i.e. NETGEN-1D-2D-3D and NETGEN 3D Parameters respectively. The hypothesis has the following inputs:

- Maximum element size = 0.002
- Minimum element size = 0
- Optimize the mesh
- Create a coarse mesh (Growth rate = 0.5, Segments per Edge = 0.5, segments per radius = 1.5)
- Limit the element size by the curvature

The blade is divided into several face groups in the GEOM module. These groups are used to create sub meshes. The sub meshes ensure finer mesh at small faces and coarser mesh at large faces. The sub mesh with the smallest faces is called  $submesh_0$  and so on till  $submesh_n$ . The first few groups with very small faces are meshed using the MEFISTO algorithm with local hypothesis. These faces have very thin, almost sliver like features. MEFISTO creates a mesh whereas NETGEN fails to generate a mesh. The MEFISTO algorithm can deal with sliver faces better than NETGEN. The next few groups are meshed using the NETGEN 2D algorithm with Local length hypothesis. The groups with the larger faces are meshed with the NETGEN-1D-2D algorithm with the NETGEN 2D hypothesis.

The submeshes create only 2D mesh for the blade geometry. Then the global mesh is invoked and a 3D mesh is generated for the blade. Finally, the 3D mesh for disk is generated. Figure 5.19 depicts the workflow in the meshing, SMESH module program.

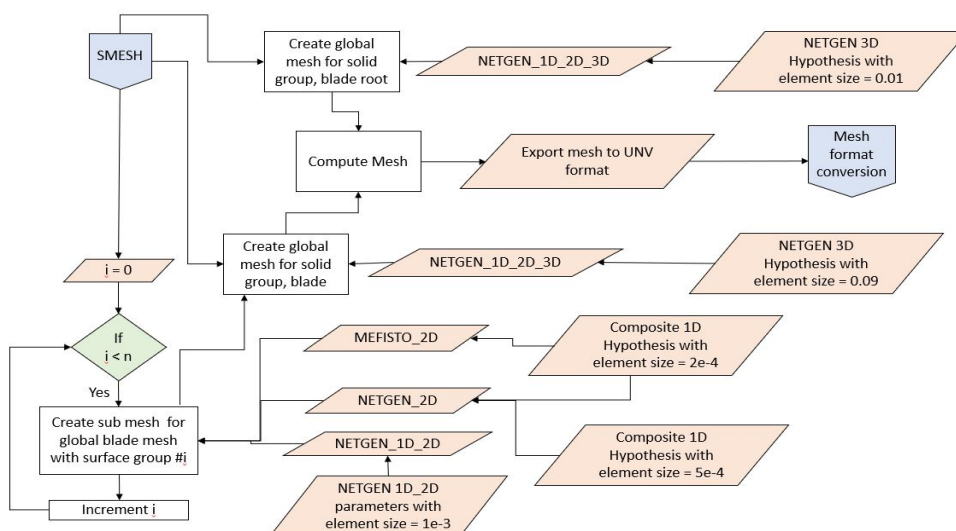


FIGURE 5.19: Flowchart of the processes in the meshing module on the HPT blade and disk assembly

As the generated global mesh for the original target geometry of the HPT blade and disk cannot be shown due to it being protected, the same program is applied on the test geometry in figure 5.2.

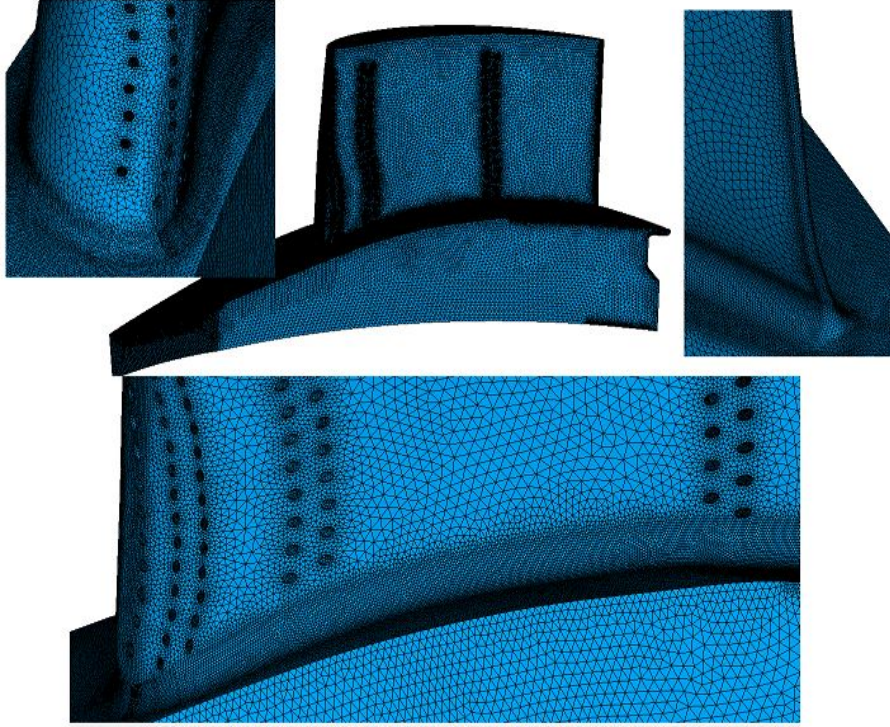


FIGURE 5.20: Blade Mesh

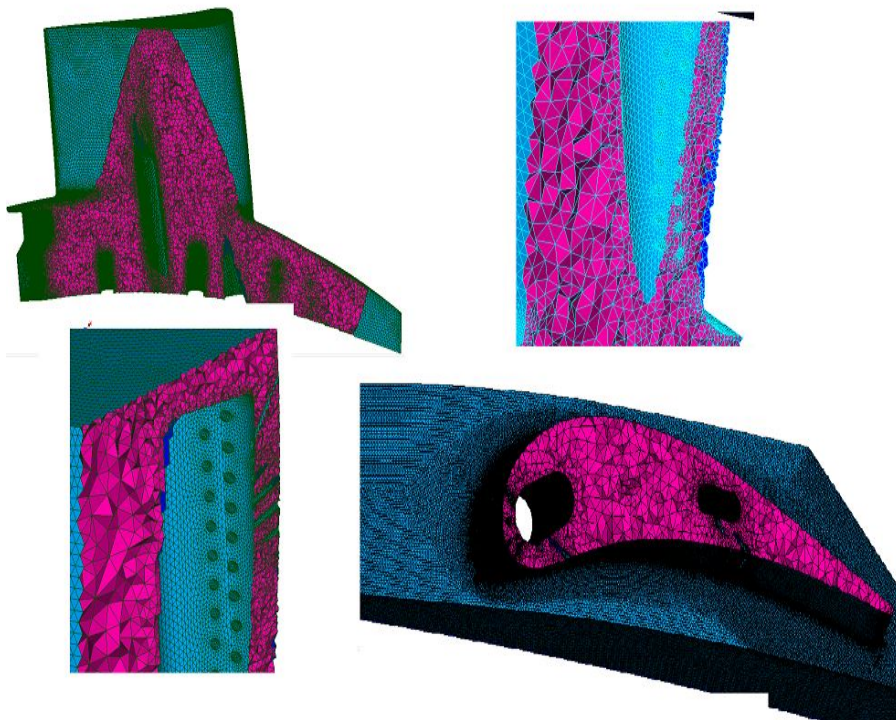


FIGURE 5.21: Blade Mesh of the internal cooling pipes

Hence, the mesh generated for the test blade is shown in figure 5.20. The internal elements are depicted in figure 5.21 with cut-sections. The mesh meets all the criteria specified in section 5.2.3. The quality criteria are also optimum for the mesh. A plot for the aspect ratio can be studied in figure 5.22. The aspect ratio of 10 was used to create this plot in Hypermesh.

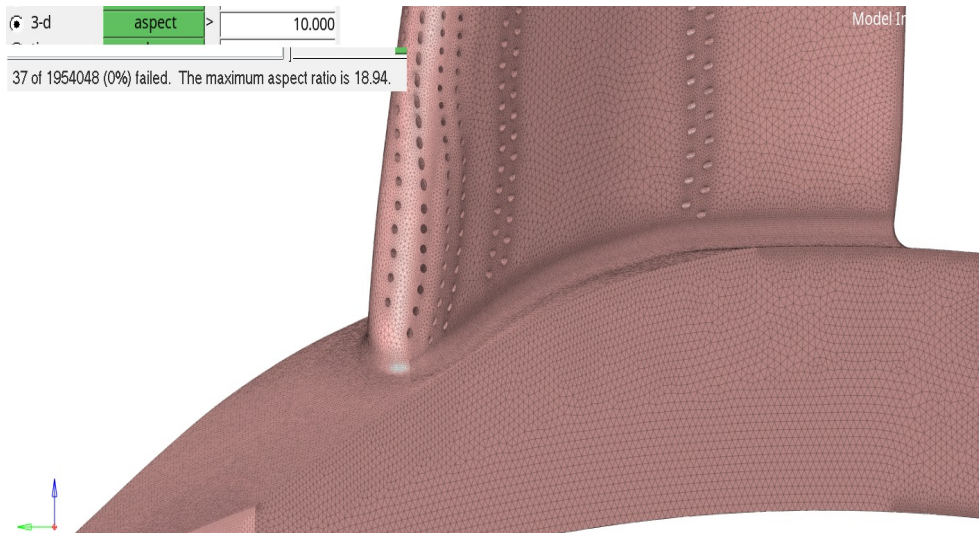


FIGURE 5.22: Aspect ratio errors in the blade mesh

The python program created in this section for the automated mesh generation of the HPT turbine and disk assembly is depicted in Appendix A.

### 5.2.5 Interfaces

The output from the mesh generation process outputs the mesh as a 'UNV' file format. UNV or universal files are ASCII data files that store information from a model file, to interface with programs or to transfer information between different types of computer [2].

A UNV file is built up with blocks of information called datasets. Each block is identified with a -1, in column 5 & 6. The remainder of the line is blank. This is a dataset delimiter. The next line is the dataset number. It is followed by data dependent on the dataset number. The relevant dataset numbers are:

- 2411: Nodes - Double Precision
- 2412: Elements
- 2467: Permanent Groups

The mesh generated through this process will be used to perform the structural-mechanical analysis of the HPT blade. Hence, the desired output from the automatic meshing tool should be readable in PERMAS. PERMAS is the Finite Element solver that would be used to perform the pseudo structural analysis of the component in section 5.2.3. Hence, a 'DAT' file is desired at this point. Hence, the UNV file generated before is to be converted to a DAT. The dataset delimiter and the dataset numbers are used to write a script to do the conversion. Figure 5.23 portrays the flowchart of the conversion. The script can be studied in Appendix B.

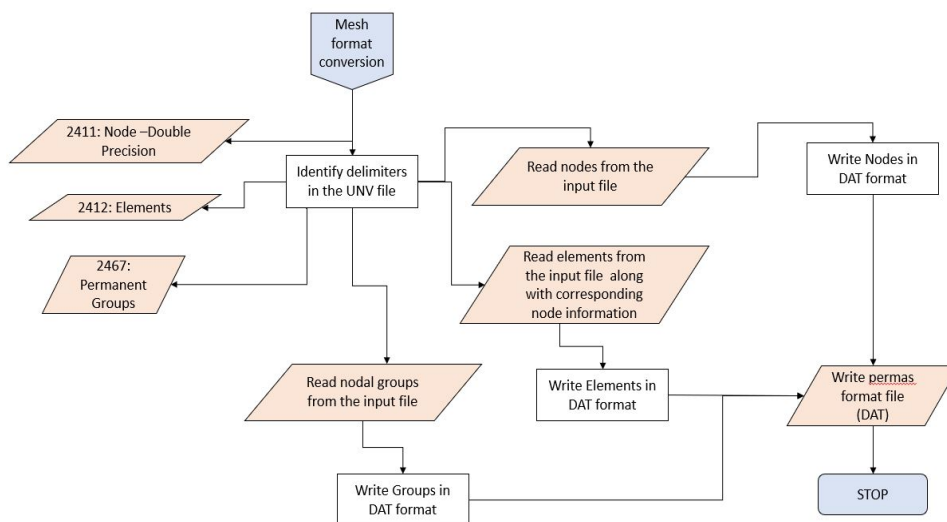


FIGURE 5.23: Flowchart to convert a UNV to a DAT file

The output is a DAT file with the node information, information about Tetrahedrons with quadratic shape functions and the nodal groups.

### 5.3 Conclusion

A rigorous evaluation of existing open-source and proprietary mesh generation programs is accomplished. SALOME is selected as the preferred mesh generation program based on several important criteria. The meshing capability of SALOME is extended and customized to automatically mesh the target geometry (HPT blade and disk assembly), meeting the mesh requirements by the HPT component team. Further, a file format conversion tool is established to ensure that the mesh generated is readable in the desired format.

## Chapter 6

# Conclusion

The aim of the work "Development of an Automatic TET10 Meshing program for rotating components" was an automation of the mesh generation process for a High Pressure Turbine blade and disk assembly. Three important goals were pursued during the completion of the individual tasks.

First, the establishment of the appropriate element type was done. This was achieved through a element study with hexahedrons and tetrahedrons. Tetrahedrons with quadratic shape function were the preferred element. The algorithms suited to the generation of tetrahedrons were then studied. Algorithms using Delaunay triangulation are the most common. These kind of algorithms were also used extensively in this work to generate meshes. Another popular algorithm, Advanced Front Method was also used.

Second, the investigation of mesh generation programs already present and suited to the task at hand (generation of a mesh on a turbine blade) was established. Many open-source and proprietary programs were assessed. SALOME was determined to be the best fit by satisfying most of the important evaluation criteria.

As a final point, the automation of the task of creating a mesh for a turbine blade was executed. A tool was created which has the geometry as input, which generated groups based on the size of entities, automatically selected appropriate algorithms and element sizes. Further, this tool outputs the mesh of the blade and disk with relevant nodal groups. The output of this tool is then converted to a desired format using a small program written in python.

# Bibliography

- [1] L. Paul Chew. “Constrained Delaunay Triangulations”. In: *Algorithmica* 4.1 (1989), pp. 97–108.
- [2] University of Cincinnati. “Universal File Datasets Summary”. In: (). URL: <http://sdrl.uc.edu/sdrl/referenceinfo/universalfileformats/file-format-storehouse/universal-file-datasets-summary>.
- [3] Canadian Neutron Beam Centre (CNBC). “Developing Technology For Repairing Advanced Jet Engines”. In: (). URL: <https://cins.ca/2017/02/01/aero-4/>.
- [4] Institute for the structures DLR, department for component design design, and manufacturing technologies. “Automation of the structural mechanical design process”. In: ().
- [5] ENNOVA. “Meshing Solutions”. In: (). URL: <http://ennova-cfd.com/services.html>.
- [6] Walter Frei. “Meshing Your Geometry: When to Use the Various Element Types”. In: (Nov. 2013). URL: <https://www.comsol.com/blogs/meshing-your-geometry-various-element-types/>.
- [7] Christophe Geuzaine and Jean-François Remacle. “Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities”. In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), pp. 1309–1331. DOI: 10.1002/nme.2579. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2579>.
- [8] Mohammad Hamdan and M. Al-Nimr. “Thermal Augmentation in Internal Cooling Passage by Converting Impingement Jet to Induced Swirl Flow”. In: Jan. 2009.
- [9] “Holes”. In: *Peterson’s Stress Concentration Factors*. John Wiley & Sons, Ltd, 2008. Chap. 4, pp. 176–400. ISBN: 9780470211106. DOI: 10.1002/9780470211106.ch4. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470211106.ch4>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470211106.ch4>.
- [10] Altair Hyperworks. “BasicFEA Introduction”. In: (). URL: <https://connect.altair.com/CP/kb-view.html?kb=156263>.

- [11] Nishant Kumar Jha and Sharad Agarwal; IIT Kanpur. "AUTOMATIC MESH GENERATION (2-D)". In: (). URL: [http://home.iitk.ac.in/~amit/courses/751/97/Mesh\\_Generation\\_2D/proj.html](http://home.iitk.ac.in/~amit/courses/751/97/Mesh_Generation_2D/proj.html).
- [12] Universitat Politècnica de Catalunya Josep sarrate. *A brief introduction to mesh generation*.
- [13] LS-Dyna. "Element Locking". In: (). URL: <https://www.dynasupport.com/tutorial/element-locking/>.
- [14] Michele Marino. *Finite Elements 1*. Oct. 2017.
- [15] Michele Marino. *Finite Elements 2*. Apr. 2018.
- [16] MIT OpenCourseWare. *Numerical Fluid Mechanics - Lecture 22*. Apr. 2015.
- [17] Steven J. Owen. "A Survey of Unstructured Mesh Generation Technology". In: *7th International Meshing Roundtable 3* (May 2000).
- [18] FACTOR NGV DLR Institut of Propulsion Technology Department Turbine. "High Pressure Turbine blade". In: ().
- [19] Andre Ribes and Christian Caremoli. "Salome platform component model for numerical simulation". In: vol. 2. Aug. 2007, pp. 553–564. ISBN: 0-7695-2870-8. DOI: 10.1109/COMPSAC.2007.185.
- [20] SALOME. "Introduction to MESH". In: (). URL: <https://docs.salome-platform.org/7/gui/SMESH/index.html>.
- [21] SALOME. "Introduction to NETGENPLUGIN". In: (). URL: <https://docs.salome-platform.org/latest/gui/NETGENPLUGIN/index.html>.
- [22] Robert Schneider. "List of public domain and commercial mesh generators". In: (). URL: <http://www.robertschneiders.de/meshgeneration/software.html>.
- [23] Joachim Schoeberl. "NETGEN An advancing front 2D/3D-mesh generator based on abstract rules". In: *Computing and Visualization in Science 1* (July 1997), pp. 41–52. DOI: 10.1007/s007910050004.
- [24] Gargi Shah. "Open source software Vs. Commercial software". In: (). URL: <https://www.infostretch.com/blog/open-source-software-vs-commercial-software/>.
- [25] Michael Ian Shamos and Dan Hoey. "Closest-Point Problems". In: *16th Annual Symposium on Foundations of Computer Science (Berkeley, California)* (Oct. 1975), pp. 151–162.
- [26] Jonathan Richard Shewchuk. "Lecture Notes on Delaunay Mesh Generation". In: (Feb. 2012).
- [27] Siemens. "Minimize costly prototyping & bring your products to market faster". In: (). URL: <https://www.plm.automation.siemens.com/global/en/products/simcenter/femap.html>.

- 
- [28] Ruhr-Universität Bochum Institute for Structural Mechanics. "Shape Function Generation and Requirements". In: (). URL: [http://www.sd.ruhr-uni-bochum.de/downloads/Generation\\_Requirements\\_shape\\_funct.pdf](http://www.sd.ruhr-uni-bochum.de/downloads/Generation_Requirements_shape_funct.pdf).
- [29] Politechnika Swietokrzyska. "The Isoparametric Representation". In: (). URL: [http://kis.tu.kielce.pl/mo/COLORADO\\_FEM/colorado/IFEM.Ch16.pdf](http://kis.tu.kielce.pl/mo/COLORADO_FEM/colorado/IFEM.Ch16.pdf).
- [30] Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill. *Handbook of Grid Generation*. USA: CRC Press LLC., 1999.
- [31] Tomas Kellner. *Bringing Back the Bling: New Process Recovers Precious Platinum from "Smut"*. [Online; accessed 17-April-2020 ]. 2015. URL: <https://www.ge.com/reports/post/115132114280/bringing-back-the-bling-new-process-recovers/2>.



## Appendix A

# Program for Automatic Meshing

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar  5 10:50:20 2020

@author: kusu_ch
"""

import salome
salome.salome_init()
from salome.geom import geomBuilder
geompy = geomBuilder.New()

import SMESH
from salome.smesh import smeshBuilder
smesh = smeshBuilder.New()

import math

##Geometry

HPT = geompy.ImportSTEP("HPT_Rotorsegment_NUR_DLR_INTERN.stp", False, False)

#Selection of Solids
selectedSolids = []
Vol = 0.000100
for solid in geompy.ExtractShapes(HPT, geompy.ShapeType["SOLID"], True):
    Volume = geompy.BasicProperties(solid)[2]
    #print(solid)
    if Volume < Vol:
        Blade = geompy.CreateGroup(HPT, geompy.ShapeType["SOLID"])
```

```

        geompy.UnionList(Blade, [solid])
    else:
        Root = geompy.CreateGroup(HPT, geompy.ShapeType["SOLID"])
        geompy.UnionList(Root, [solid])

geompy.addToStudy(HPT, 'HPT')
geompy.addToStudyInFather(HPT, Root, 'Root')
geompy.addToStudyInFather(HPT, Blade, 'Blade')

#identifying small edges
small_edges=geompy.CreateGroup(Blade, geompy.ShapeType["EDGE"])
for edges in geompy.ExtractShapes(Blade, geompy.ShapeType["EDGE"], False):
    if geompy.BasicProperties(edges)[0] < 1e-4:
        geompy.UnionList(small_edges, [edges])
geompy.addToStudyInFather(Blade, small_edges, "small_edges")
small_edges.SetName("small_edges")

#grouping faces by size
faceArea = []
for face in geompy.ExtractShapes(Blade, geompy.ShapeType["FACE"], False):
    faceArea.append(geompy.BasicProperties(face)[1])
maxArea = max(faceArea)
minArea = min(faceArea)
n = 10
r = math.pow((maxArea/minArea), (1/n))

i=0
bins=[]
while i < n+1:
    bins.append(minArea*(math.pow(r,i)))
    i+=1
bins[-1]=maxArea

i=0
grp=[[[] for a in range(len(bins))]]
biggest=[]
while i < len(bins)-1:
    for face in geompy.ExtractShapes(Blade, geompy.ShapeType["FACE"], False):
        area = geompy.BasicProperties(face)[1]
        if area > bins[i] and area < bins[i+1]:
            grp[i].append(face)
        if area == bins[-1]:

```

```
        biggest.append(face)
    i+=1
geompy.addToStudyInFather( Blade, biggest, 'biggest')

name_grp=[]
j = 0
for counter in grp:
    i = 0
    if len(counter) != 0:
        name_grp.append(geompy.CreateGroup(Blade, geompy.ShapeType["FACE"]))
        while i < len(counter):
            geompy.UnionList(name_grp[j], [counter[i]])
            i+=1
        name = 'grp_%02d' % j
        geompy.addToStudyInFather( Blade, name_grp[j], name)
        j+=1
big_faces = geompy.CreateGroup(Blade, geompy.ShapeType["FACE"])
for face in biggest:
    geompy.UnionList(big_faces,[face])
geompy.addToStudyInFather( Blade, big_faces, 'big_faces')

tot_grps = j

##Meshing module

import functions

blade_mesh = smesh.Mesh(Blade, "blade_mesh")

NETGEN_1D_2D = blade_mesh.Triangle(algo=smeshBuilder.NETGEN_1D2D)
NETGEN_2D_Parameters = NETGEN_1D_2D.Parameters()
NETGEN_2D_Parameters.SetMaxSize( 0.001 )
NETGEN_2D_Parameters.SetMinSize( 0 )
NETGEN_2D_Parameters.SetSecondOrder( 0 )
NETGEN_2D_Parameters.SetOptimize( 1 )
NETGEN_2D_Parameters.SetFineness( 3 )
NETGEN_2D_Parameters.SetChordalError( -1 )
NETGEN_2D_Parameters.SetChordalErrorEnabled( 0 )
NETGEN_2D_Parameters.SetUseSurfaceCurvature( 1 )
NETGEN_2D_Parameters.SetFuseEdges( 1 )
NETGEN_2D_Parameters.SetUseDelauney( 1 )
NETGEN_2D_Parameters.SetQuadAllowed( 0 )
```

```
NETGEN_2D_Parameters.SetWorstElemMeasure( 0 )
NETGEN_2D_Parameters.SetCheckChartBoundary( 168 )

NETGEN_3D = blade_mesh.Tetrahedron()
NETGEN_3D_Parameters = NETGEN_3D.Parameters()
NETGEN_3D_Parameters.SetMaxSize( 0.001 )
NETGEN_3D_Parameters.SetMinSize( 0 )
NETGEN_3D_Parameters.SetOptimize( 1 )
NETGEN_3D_Parameters.SetFineness( 2 )
NETGEN_3D_Parameters.SetCheckOverlapping( 0 )
NETGEN_3D_Parameters.SetElemSizeWeight( 1.23594e-311 )
NETGEN_3D_Parameters.SetCheckChartBoundary( 168 )

root_mesh = smesh.Mesh(Root, "root_mesh")
NETGEN_3D_root = root_mesh.Tetrahedron(algo=smeshBuilder.NETGEN_1D2D3D)
a3D_param = NETGEN_3D_root.Parameters()
a3D_param.SetMaxSize( 0.002 )
a3D_param.SetMinSize( 0 )
a3D_param.SetSecondOrder( 0 )
a3D_param.SetOptimize( 1 )
a3D_param.SetFineness( 2 )
a3D_param.SetChordalError( 0 )
a3D_param.SetChordalErrorEnabled( 0 )
a3D_param.SetUseSurfaceCurvature( 1 )
a3D_param.SetFuseEdges( 1 )
a3D_param.SetQuadAllowed( 0 )
a3D_param.SetCheckChartBoundary( 152 )

i=0
name_sm=[]
size = 0.001
#name_sm.append(functions.Composite_1D_algo(blade_mesh, small_edges,size).GetSubMesh())

while i < tot_grps:
    if i == 0:
        size = 0.001
        functions.Composite_1D_algo(blade_mesh, name_grp[i],size)
        name_sm.append(functions.MEFISTO_2D_algo(blade_mesh, name_grp[i]))
    elif i>0 and i<4:
        size = 2e-4
        functions.Composite_1D_algo(blade_mesh, name_grp[i],size)
        name_sm.append(functions.MEFISTO_2D_algo(blade_mesh, name_grp[i]))
```

```

elif i>3 and i<7:
    size = 2e-4
    functions.Composite_1D_algo(blade_mesh, name_grp[i],size)
    name_sm.append(functions.NETGEN_2D_only(blade_mesh, name_grp[i]))
elif i>6 and i<9:
    size = 5e-4
    functions.Composite_1D_algo(blade_mesh, name_grp[i],size)
    name_sm.append(functions.NETGEN_2D_only(blade_mesh, name_grp[i]))
else:
    size = 8e-4
    name_sm.append(functions.NETGEN_1D2D_algo(blade_mesh, name_grp[i],size))
i+=1

name_sm.append(functions.NETGEN_1D2D_algo(blade_mesh, big_faces,size))

order = blade_mesh.SetMeshOrder( [name_sm])

isDone_global = blade_mesh.Compute()
isDone_root = root_mesh.Compute()

i=0
while i < len(name_sm):
    name_sb = 'submesh_%02d' %i
    smesh.SetName(name_sm[i], name_sb)
    i+=1

smesh.SetName(blade_mesh, 'blade_mesh')
smesh.SetName(root_mesh, 'root_mesh')

blade_mesh.ConvertToQuadratic()
blade_mesh.ExportUNV(r'blade_mesh.unv')
root_mesh.ConvertToQuadratic()
root_mesh.ExportUNV(r'root_mesh.unv')

HPT_mesh = smesh.Concatenate([blade_mesh.GetMesh(), root_mesh.GetMesh()], 1,0,1e-5, True)
[Grblade_mesh_Nodes, Grblade_mesh_Edges, Grblade_mesh_Faces, Grblade_mesh_Volumes, Grroot_m
HPT_mesh.ExportUNV(r'HPT_mesh.unv')

Program Functions.py
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 5 13:43:04 2020

```

```
@author: kusu_ch
"""
import SMESH
from salome.smesh import smeshBuilder
smesh = smeshBuilder.New()

def StartCancelClock( theMesh, theMeshingTimeLimit ):
    """
    Start a timer to cancel meshing in some time
    """
    def cancelMeshing( theMesh ):
        print('in cancel now')
        smesh = theMesh.GetEngine()
        mesh = theMesh.GetMesh()
        smesh.CancelCompute( mesh, None )
        return

    from threading import Timer
    timer = Timer( theMeshingTimeLimit, cancelMeshing, args=( theMesh, ))
    timer.start()
    return timer

def Composite_1D_algo(blade_mesh, name, size):
    Composite1d = blade_mesh.Segment(algo=smeshBuilder.COMPOSITE,geom=name)
    Composite1d.LocalLength(size)
    return Composite1d

def Deflection_1D_algo(blade_mesh, name,defl):
    Composite1d = blade_mesh.Segment(algo=smeshBuilder.COMPOSITE,geom=name)
    Deflection_1 = Composite1d.Deflection1D(defl)
    submesh = Composite1d.GetSubMesh()
    return submesh

def NETGEN_2D_only(blade_mesh, name):
    NETGEN_2D = blade_mesh.Triangle(algo=smeshBuilder.NETGEN_2D,geom=name)
    NETGEN_2D.LengthFromEdges()
    submesh = NETGEN_2D.GetSubMesh()
    return submesh

def MEFISTO_2D_algo(blade_mesh, name):
    MEFISTO_2D = blade_mesh.Triangle(algo=smeshBuilder.MEFISTO, geom=name)
```

```
    submesh = MEFISTO_2D.GetSubMesh()
    return submesh

def NETGEN_1D2D_algo(blade_mesh, name, size):
    NETGEN_1D2D = blade_mesh.Triangle(algo=smeshBuilder.NETGEN_1D2D, geom = name)
    NG_param = NETGEN_1D2D.Parameters()
    NG_param.SetMaxSize( size )
    NG_param.SetMinSize( 0 )
    NG_param.SetOptimize( 1 )
    NG_param.SetFineness( 2 )
    NG_param.SetUseSurfaceCurvature( 1 )
    NG_param.SetQuadAllowed( 0 )
    NG_param.SetUseDelauney( 1 )
    submesh = NETGEN_1D2D.GetSubMesh()
    return submesh
```

## Appendix B

# Format conversion of mesh export file

```
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 12 15:04:22 2019

@author: kusu_ch
"""

delimiter = '    -1\n'
firstnode = 18
x=[]
y=[]
z=[]
num=[]
enum=[]
nodes=[]
nodes1=[]
grp_no=[]
grp_name=[]
no_entity=[]
counter=[]
grp_type=[]
flag = 0
k = 0

with open("HPT_mesh.unv", 'rt') as fp:
    #read node info
    for i, line in enumerate(fp):
        if i > firstnode:
```



```

    if line != delimiter:
        if i % 2 == 0:
            #print(i,line)
            x.append(line.split()[0])
            y.append(line.split()[1])
            z.append(line.split()[2])
        else:
            num.append(line.split()[0])
    else:
        print(line)
        firstelem = i+2; break
node_info = {1:num, 2:x, 3:y, 4:z}

#read element info
fp.seek(0)
for i, line in enumerate(fp):
    if i > firstelem:
        if line != delimiter:
            if line.split()[-1] == str(10) and len(line.split()) == 6:
                flag = 1
                first3d = i
                break
            else:
                continue
        else:
            firstgrp = i+2
            break

if flag == 1:
    fp.seek(0)
    for i, line in enumerate(fp):
        if i > first3d:
            if line != delimiter:
                if len(line.split()) == 6:
                    enum.append(line.split()[0])
                else:
                    if len(line.split()) == 8:
                        nodes.append(line.split())
                    else:
                        nodes1.append(line.split())
            else:
                firstgrp = i+2

```

```

        break
    elem_info = {1:enum, 2:nodes, 3:nodes1}
#extraction of groups
fp.seek(0)
for i, line in enumerate(fp):
    if i > firstgrp:
        if line != delimiter:
            if line.split()[-1] != str(0) and (len(line.split())) != 1:
                grp_no.append(line.split()[0])
                no_entity.append(line.split()[-1])
                counter.append(i)
            elif line.split()[-1] != str(0) and (len(line.split())) == 1:
                grp_name.append(line.split()[0])
            else:
                continue

nset=[[[] for a in range(len(counter))]
eset=[[[] for a in range(len(counter))]

while k < len(counter):
    fp.seek(0)
    for i, line in enumerate(fp):
        if i > counter[k] + 1 and i < (counter[k] + int(no_entity[k])/2)+2 :
            if line.split()[0] == str(7):
                nset[k].append(line.split()[1])
                nset[k].append(line.split()[5])
            if line.split()[0] == str(8):
                eset[k].append(line.split()[1])
                eset[k].append(line.split()[5])
        k+=1

#writing information to DAT file
with open("HPT_mesh.dat", "w") as dat:
    dat.write('\n $STRUCTURE\n')
    dat.write('\n $COORD\n')
    #write nodes
    i=0
    while i < len(num):
        dat.write('\n          ')
        dat.write(str(num[i])+'\t'+str(x[i])+'\t'+str(y[i])+'\t'+str(z[i]))
        i+=1
    #print(i)2044148

```

```

#write elements
dat.write('\n\n\n      $ELEMENT   TYPE = TET10   ESET = auto1')
i=0
while i < len(enum):
    dat.write('\n          '+str(enum[i])+'\t'+str(nodes[i][0])+'\t'+str(nodes[i][1])
    dat.write('      &          '+str(nodes[i][5])+'\t'+str(nodes[i][6])+'\t'+str
    i+=1
#write groups
i=0
while i < len(grp_name):
    print(i)
    if eset[i] == []:
        print(grp_name[i])
        dat.write('\n$NSET  NAME = '+ grp_name[i])
        j=0
        try:
            while j <= int(no_entity[i]):
                dat.write('\n')
                k=0
                while k < 8:
                    dat.write('\t' + nset[i][j+k])
                    k+=1
                j+=8
        except IndexError:
            pass
    if nset[i] == []:
        print(grp_name[i])
        dat.write('\n$ESET  NAME = '+ grp_name[i])
        j=0
        try:
            while j <= int(no_entity[i]):
                dat.write('\n')
                k=0
                while k < 8:
                    dat.write('\t' + eset[i][j+k])
                    k+=1
                j+=8
        except IndexError:
            pass
    i+=1

```

## Appendix C

# List of Mesh generation softwares

### 1. Open-Source Mesh generators

- Gmsh - <https://gmsh.info/>
- DeIPSC
- SALOME - <https://www.salome-platform.org/>
- Tetgen - <http://wias-berlin.de/software/tetgen/>
- LBIE-Mesher - <https://www.cs.utexas.edu/bajaj/cvc/software/LBIE.shtml>
- MeshGenC++ - <http://www.dogpack-code.org/MeshGenC++/>
- DistMesh - <http://persson.berkeley.edu/distmesh/>
- FELICITY -  
[https://www.math.lsu.edu/~walker/pdfs/Walker2018\\_FELICITY\\_Matlab\\_CPP\\_Toolbox.pdf](https://www.math.lsu.edu/~walker/pdfs/Walker2018_FELICITY_Matlab_CPP_Toolbox.pdf)
- CGAL mesh generation -  
[https://doc.cgal.org/latest/Mesh\\_3/index.html](https://doc.cgal.org/latest/Mesh_3/index.html)

### 2. Closed/Proprietary Mesh generators

- BOXERmesh - <https://www.cfd-online.com/Wiki/BOXERMesh>
- GID - <https://www.gidhome.com/>
- CM2MeshTools - <https://www.computing-objects.com/cm2-meshtools-suite/>
- Castnet - <http://www.dhcae-tools.com/CastNet.html>
- NISA-Display IV - <https://www.nisasoftware.com/software/nisa-mechanical/display-iv>
- ENNOVA - <http://ennova-cfd.com/>
- FEMAP - <https://www.plm.automation.siemens.com/global/en/products/simcenter/femap.f>
- CADfix - <https://www.cadinterop.com/en/your-needs/cad-data-reuse-for-cae/cadfix-tartan-meshing.html>
- DIANA(FEMGV) - <https://dianafea.com/femgv>